# Językowe Przedszkole

Plan na to ćwiczenie jest prosty – chcemy (częściowo) powtórzyć sukces zespołu odpowiedzialnego za DeepSeek R1 i wykorzystać zaproponowaną przez nich metodę (GRPO – Group Relative Policy Optimisation) jako „środek wychowawczy" służący do zmiany zachowania istniejącego już, wytrenowanego modelu językowego.

GRPO to prosty algorytm…ale diabeł tkwi w szczegółach (patrz: problemy z siostrzaną metodą PPO https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/ ). Z tego powodu (tym razem) zamiast implementować wszystko „ręcznie" skorzystamy z (umiarkowanie) sprawdzonych rozwiązań bibliotecznych.

Co chcemy zrobić? Na początek coś prostego – zmusimy generatywny model językowy by wypowiadał się z wykorzystaniem przede wszystkim czteroliterowych słów. Dlaczego? Załóżmy, że są krótkie i estetyczne. ;]

Jakie kroki są przed nami?

- W czasie treningu LLM będzie musiał kontynuować wypowiedzi zawarte w podanych na start promptach. Przygotujmy ich małą bazę.
    - Stwórzymy nowy plik tekstowy, wpiszmy do niego (w kolejnych liniach) kilka-kilkanaście głupich (e.g. „What does the ox sing?") pytań w języku angielskim.
    - Na jego bazie stwórzmy obsługiwany przez resztę bibliotek zbiór danych (przyda się biblioteka **datasets** https://huggingface.co/docs/datasets/en/index).
    - [na później] Jak już wszystko zadziała – wróć do tego punktu i powiększ listę promptów tak, by było ich minimum kilkaset. Skąd brać takie ich ilości (podpowiedź: może przyda się inny LLM?)? Jak większy zbiór promptów wpłynął na przebieg treningu?
- Następnie przygotujmy funkcję nagrody – powinna przyjmować listę promptów „prompts" oraz listę odpowiedzi „completions", a zwracać listę nagród (lub kar).
    - Tutaj wystarczy „czysty" Python.
- Potem przygotujmy trzon mechanizmu treningowego.
    - Skorzystajmy z biblioteki **trl** i klasy dedykowanej dla GRPO (https://huggingface.co/docs/trl/main/en/grpo_trainer).
    - Jako bazowy model wykorzystajmy stare, klasyczne GPT2 (
        - [na później] jeżeli mamy dostęp do dużych ilości VRAM, to możemy się szarpnąć na coś ciekawszego, np. wymieniony w dokumentacji Qwen2-0.5B-Instruct
    - Zadbajmy też o odpowiednią konfigurację treningu. To de facto główna sprawa w tym ćwiczeniu.

        - Podpowiedź: sprawdź jak na trening wpływają takie parametry jak „max_completion_length”, „gradient_accumulation_steps”, „learning_rate”,  “beta”, czy “epsilon”.
            - Najpierw postaraj się znaleźć zestaw, który „działa”. Utrzymaj jak najwięcej wartości domyślny, eksperymentuj z jednym parametrem „na raz”.
            - [na później] Spróbuj „zepsuć” trening wyłączając (lub nadmiernie wzmacniając) kluczowe dla GRPO mechanizmy. Czy zachowanie było zgodne z oczekiwanym.
- Na koniec przygotowujemy pętlę treningową i monitorujemy efekty.
    - Pamiętajmy, by poza obserwowaniem metryk, sprawdzić też „na oko” zachowanie modelu (dostępnego jako trainer.model) przed i po treningu. Tu przyda się biblioteka **transformers** (a w niej klasa AutoTokenizer oraz moduł pipeline).
- [na później] Na tym etapie powinno udać się zmusić model do bycia zwolennikiem słów czteroliterowych. Plusem GRPO jest to, że reaguje na nagrody, a one mogą być przyznawane w dowolny sposób. Zaprojektuj własną funkcję nagrody (a nawet kilka) – im bardziej szalone pomysły tym lepiej (ale warto, by trening nie trwał kilku dni  by mogły sobie z nimi poradzić mniejsze modele). **Przetestuj przynajmniej jeden alternatywny cel treningu (schemat nagród)!** Pamiętaj, że możesz użyć kilku równocześnie. Dobrej zabawy!

Całość ćwiczenia jest intencjonalnie otwarta. Do zdobycia maksymalnej ilości punktów wystarczy:

- znalezienie konfiguracji, która „działa” (zmienia zachowanie LLMa, zarazem nie „psując” go kompletnie);
- porównanie przebiegu i efektów treningu z trzema innymi konfiguracjami (niekoniecznie „działającymi”);
- sprawdzenie efektów wprowadzenia własnej funkcji nagrody (nagradzającej za coś innego, niż słowa czteroliterowe).