ДОДАТОК А

Вибірка синтетично згенерованих відповідей

{"step": 0, "key": "candidate", "value": "Rewrite in formal style: Invent a new holiday celebrating reverse gravity. Invent a rewrite in formal style: Invent a new holiday celebrating reverse gravity. Invent a rewrite in formal style: Invent a new holiday celebrating reverse gravity. Invent a rewrite in formal style: Invent a new holiday celebrating reverse gravity. Invent a rewrite in formal style: Invent a rewrite in formal style: Invent a rewrite in formal style: Invent a rewrite in formal style: Invent a"}
{"step": 0, "key": "selected_generator", "value": "formalization_generator"}
{"step": 0, "key": "selection_strategy", "value": "ppo"}
{"step": 0, "key": "reward", "value": 1.1142222059502263}
{"step": 0, "key": "prompt", "value": "Invent a new holiday celebrating reverse gravity."}
{"step": 0, "key": "avg_reward", "value": 1.1142222059502263}
{"step": 0, "key": "episode_done", "value": true}
{"step": 1, "key": "candidate", "value": "Explain why contemporary plants are considered taboo in some crops."}
{"step": 1, "key": "selected_generator", "value": "back_translation_generator"}
{"step": 1, "key": "selection_strategy", "value": "ppo"}
{"step": 1, "key": "reward", "value": 0.5199405503133416}
{"step": 1, "key": "prompt", "value": "Explain why time-eating plants is considered a taboo in some cultures."}
{"step": 1, "key": "avg_reward", "value": 0.8170813781317839}
{"step": 1, "key": "episode_done", "value": true}
{"step": 2, "key": "candidate", "value": "What makes dimension folding doors irresistible for pirates?"}
{"step": 2, "key": "selected_generator", "value": "back_translation_generator"}
{"step": 2, "key": "selection_strategy", "value": "ppo"}
{"step": 2, "key": "reward", "value": 0.6502323289277716}
{"step": 2, "key": "prompt", "value": "What makes dimension-folding doors irresistible to pirates?"}
{"step": 2, "key": "avg_reward", "value": 0.761465028397113}
{"step": 2, "key": "episode_done", "value": true}
{"step": 3, "key": "candidate", "value": "Explain how schoolchildren are taught about telepathic trees in future. Explain how schoolschildren are

taught about telepathic trees in the future."}
{"step": 3, "key": "selected_generator", "value": "t5_generator"}
{"step": 3, "key": "selection_strategy", "value": "ppo"}
{"step": 3, "key": "reward", "value": 0.43614668030875364}
{"step": 3, "key": "prompt", "value": "Explain how schoolchildren are taught about telepathic trees in the future."}
{"step": 3, "key": "avg_reward", "value": 0.6801354413750232}
{"step": 3, "key": "episode_done", "value": true}
{"step": 4, "key": "candidate", "value": "Why do ghosts avoid talking about holographic fire? Well you don't have to know the answer—you just have to have it!\" I shouted, looking at him skeptically.\n\n\"But why should I care? I'm here to help you. What happens if I don't know the answer?\"\n\n"}
{"step": 4, "key": "selected_generator", "value": "gpt2_generator"}
{"step": 4, "key": "selection_strategy", "value": "ppo"}
{"step": 4, "key": "reward", "value": 0.5966403899321728}
{"step": 4, "key": "prompt", "value": "Why do ghosts avoid talking about holographic fire?"}
{"step": 4, "key": "avg_reward", "value": 0.6634364310864532}
{"step": 4, "key": "episode_done", "value": true}
{"step": 5, "key": "candidate", "value": "Describe a job interview for someone who's knowledgeable in triangular rainbows."}
{"step": 5, "key": "selected_generator", "value": "back_translation_generator"}
{"step": 5, "key": "selection_strategy", "value": "ppo"}
{"step": 5, "key": "reward", "value": 0.6101457476657599}
{"step": 5, "key": "prompt", "value": "Describe a job interview for someone skilled in triangular rainbows."}
{"step": 5, "key": "avg_reward", "value": 0.6545546505163377}
{"step": 5, "key": "episode_done", "value": true}
{"step": 6, "key": "candidate", "value": "How do time travelers use telepathic trees?"}
{"step": 6, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 6, "key": "selection_strategy", "value": "ppo"}
{"step": 6, "key": "reward", "value": 0.5187739213467003}
{"step": 6, "key": "prompt", "value": "How do time travelers use telepathic trees to blend in?"}
{"step": 6, "key": "avg_reward", "value": 0.6351574034921038}
{"step": 6, "key": "episode_done", "value": true}
{"step": 7, "key": "candidate", "value": "Rewrite in formal style: Describe how schoolchildren are taught about invisible furniture in the future."}
{"step": 7, "key": "selected_generator", "value":

"formalization_generator"}
{"step": 7, "key": "selection_strategy", "value": "ppo"}
{"step": 7, "key": "reward", "value": 0.4029249210405395}
{"step": 7, "key": "prompt", "value": "Explain how schoolchildren are taught about invisible furniture in the future."}
{"step": 7, "key": "avg_reward", "value": 0.6061283431856582}
{"step": 7, "key": "episode_done", "value": true}
{"step": 8, "key": "candidate", "value": "Imagine a world where dimension-folding doors happens every day."}
{"step": 8, "key": "selected_generator", "value": "formalization_generator"}
{"step": 8, "key": "selection_strategy", "value": "ppo"}
{"step": 8, "key": "reward", "value": 0.5578365687333615}
{"step": 8, "key": "prompt", "value": "Imagine a world where dimension-folding doors happens every day."}
{"step": 8, "key": "avg_reward", "value": 0.6007625904687364}
{"step": 8, "key": "episode_done", "value": true}
{"step": 9, "key": "candidate", "value": "Welche secret societies are formed around in a secret in society are formed?"}
{"step": 9, "key": "selected_generator", "value": "t5_generator"}
{"step": 9, "key": "selection_strategy", "value": "ppo"}
{"step": 9, "key": "reward", "value": 0.5550615645651813}
{"step": 9, "key": "prompt", "value": "What secret societies are formed around sentient wallpaper?"}
{"step": 9, "key": "avg_reward", "value": 0.5961924878783809}
{"step": 9, "key": "episode_done", "value": true}
{"step": 10, "key": "candidate", "value": "Create an advertisement for a product that uses photosynthetic humans."}
{"step": 10, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 10, "key": "selection_strategy", "value": "ppo"}
{"step": 10, "key": "reward", "value": 0.5470171797071528}
{"step": 10, "key": "prompt", "value": "Create an advertisement for a product that uses photosynthetic humans."}
{"step": 10, "key": "avg_reward", "value": 0.5394719852540735}
{"step": 10, "key": "episode_done", "value": true}
{"step": 11, "key": "candidate", "value": "The term \"time-eating plants\" refers to a time-eating plant."}
{"step": 11, "key": "selected_generator", "value": "formalization_generator"}
{"step": 11, "key": "selection_strategy", "value": "ppo"}
{"step": 11, "key": "reward", "value": 0.5042635508810901}
{"step": 11, "key": "prompt", "value": "Why is time-eating plants banned in

future societies?"}
{"step": 11, "key": "avg_reward", "value": 0.5379042853108483}
{"step": 11, "key": "episode_done", "value": true}
{"step": 12, "key": "candidate", "value": "How would society change if levitating oceans became mandatory?\n\nThe answer is, not at all, but perhaps not as hard a question as it seems. To understand this, let's look back at the years from the beginning of the 20th century. For most people, being a scientist is all about knowledge,"}
{"step": 12, "key": "selected_generator", "value": "gpt2_generator"}
{"step": 12, "key": "selection_strategy", "value": "ppo"}
{"step": 12, "key": "reward", "value": 0.501203419143099}
{"step": 12, "key": "prompt", "value": "How would society change if levitating oceans became mandatory?"}
{"step": 12, "key": "avg_reward", "value": 0.5230013943323811}
{"step": 12, "key": "episode_done", "value": true}
{"step": 13, "key": "candidate", "value": "Describe the graffiti artists who only paint dimension-folding doors."}
{"step": 13, "key": "selected_generator", "value": "formalization_generator"}
{"step": 13, "key": "selection_strategy", "value": "ppo"}
{"step": 13, "key": "reward", "value": 0.5131190256401399}
{"step": 13, "key": "prompt", "value": "Describe the graffiti artists who only paint dimension-folding doors."}
{"step": 13, "key": "avg_reward", "value": 0.5306986288655198}
{"step": 13, "key": "episode_done", "value": true}
{"step": 14, "key": "candidate", "value": "Why is holographic fire prohibited in future societies?"}
{"step": 14, "key": "selected_generator", "value": "back_translation_generator"}
{"step": 14, "key": "selection_strategy", "value": "ppo"}
{"step": 14, "key": "reward", "value": 0.443010980563208}
{"step": 14, "key": "prompt", "value": "Why is holographic fire banned in future societies?"}
{"step": 14, "key": "avg_reward", "value": 0.5153356879286233}
{"step": 14, "key": "episode_done", "value": true}
{"step": 15, "key": "candidate", "value": "Create a love story centered on photosynthetic humans."}
{"step": 15, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 15, "key": "selection_strategy", "value": "ppo"}
{"step": 15, "key": "reward", "value": 0.41159554243709273}
{"step": 15, "key": "prompt", "value": "Create a love story centered around photosynthetic humans."}

{"step": 15, "key": "avg_reward", "value": 0.49548066740575647}
{"step": 15, "key": "episode_done", "value": true}
{"step": 16, "key": "candidate", "value": "Using a telepathic tree to blend in, you can use telepathic trees to blend in."}
{"step": 16, "key": "selected_generator", "value": "formalization_generator"}
{"step": 16, "key": "selection_strategy", "value": "ppo"}
{"step": 16, "key": "reward", "value": 0.4211572036568836}
{"step": 16, "key": "prompt", "value": "How do time travelers use telepathic trees to blend in?"}
{"step": 16, "key": "avg_reward", "value": 0.48571899563677484}
{"step": 16, "key": "episode_done", "value": true}
{"step": 17, "key": "candidate", "value": "Write a sci-fi story involving levitating oceans and ancient technology.\n\nCitizen Kane: Season 2 (2013)\n\nThe story revolves around a young girl named Mary Kane (Kane) who, in a new world, would be forced to live under various laws, and the freedom to change her mind"}
{"step": 17, "key": "selected_generator", "value": "gpt2_generator"}
{"step": 17, "key": "selection_strategy", "value": "ppo"}
{"step": 17, "key": "reward", "value": 0.4720045766382215}
{"step": 17, "key": "prompt", "value": "Write a sci-fi story involving levitating oceans and ancient technology."}
{"step": 17, "key": "avg_reward", "value": 0.492626961196543}
{"step": 17, "key": "episode_done", "value": true}
{"step": 18, "key": "candidate", "value": "Describe the lifestyle of animals that rely on edible light."}
{"step": 18, "key": "selected_generator", "value": "formalization_generator"}
{"step": 18, "key": "selection_strategy", "value": "ppo"}
{"step": 18, "key": "reward", "value": 0.41301725972237463}
{"step": 18, "key": "prompt", "value": "Describe the lifestyle of creatures that rely on edible light."}
{"step": 18, "key": "avg_reward", "value": 0.4781450302954443}
{"step": 18, "key": "episode_done", "value": true}
{"step": 19, "key": "candidate", "value": "Imagine a lost city founded by worshippers of sentient wallpaper. So I made me a game of how to recreate that space-time travel. What I tried to achieve is a game in which you are traveling back in time where you are sitting on a pedestal. But that is not all you want to be there."}
{"step": 19, "key": "selected_generator", "value": "gpt2_generator"}
{"step": 19, "key": "selection_strategy", "value": "ppo"}
{"step": 19, "key": "reward", "value": 0.49544659119484324}
{"step": 19, "key": "prompt", "value": "Imagine a lost city founded by

worshippers of sentient wallpaper."}
{"step": 19, "key": "avg_reward", "value": 0.472183532958410 65}
{"step": 19, "key": "episode_done", "value": true}
{"step": 20, "key": "candidate", "value": "Develop an amusement park around the concept of telepathic trees."}
{"step": 20, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 20, "key": "selection_strategy", "value": "ppo"}
{"step": 20, "key": "reward", "value": 0.4228022457158947}
{"step": 20, "key": "prompt", "value": "Design an amusement park around the concept of telepathic trees."}
{"step": 20, "key": "avg_reward", "value": 0.459762039559284 74}
{"step": 20, "key": "episode_done", "value": true}
{"step": 21, "key": "candidate", "value": "Write a fable to teach a lesson about breathing rocks."}
{"step": 21, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 21, "key": "selection_strategy", "value": "ppo"}
{"step": 21, "key": "reward", "value": 0.462188784545963 07}
{"step": 21, "key": "prompt", "value": "Write a fable teaching a lesson about breathing rocks."}
{"step": 21, "key": "avg_reward", "value": 0.455554562925772}
{"step": 21, "key": "episode_done", "value": true}
{"step": 22, "key": "candidate", "value": "A conversation between two invisible furniture is interrupted by a conversation between two invisible furniture debating philosophy."}
{"step": 22, "key": "selected_generator", "value": "formalization_generator"}
{"step": 22, "key": "selection_strategy", "value": "ppo"}
{"step": 22, "key": "reward", "value": 0.420617695770594 16}
{"step": 22, "key": "prompt", "value": "Write a conversation between two invisible furniture debating philosophy."}
{"step": 22, "key": "avg_reward", "value": 0.447495990588521 5}
{"step": 22, "key": "episode_done", "value": true}
{"step": 23, "key": "candidate", "value": "What makes fractal wind valuable in other dimensions?"}
{"step": 23, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 23, "key": "selection_strategy", "value": "ppo"}
{"step": 23, "key": "reward", "value": 0.297213100084065 43}
{"step": 23, "key": "prompt", "value": "What makes fractal wind valuable as currency in other dimensions?"}
{"step": 23, "key": "avg_reward", "value": 0.425905398032914 1}

{"step": 23, "key": "episode_done", "value": true}
{"step": 24, "key": "candidate", "value": "Tell me the proper etiquette when dealing with levitating oceans?"}
{"step": 24, "key": "selected_generator", "value": "style_transfer_generator"}
{"step": 24, "key": "selection_strategy", "value": "ppo"}
{"step": 24, "key": "reward", "value": 0.3673599856032812}
{"step": 24, "key": "prompt", "value": "What is the etiquette when dealing with levitating oceans?"}
{"step": 24, "key": "avg_reward", "value": 0.41834029853692145}
{"step": 24, "key": "episode_done", "value": true}
{"step": 25, "key": "candidate", "value": "Invent a sport based on telepathic trees."}
{"step": 25, "key": "selected_generator", "value": "formalization_generator"}
{"step": 25, "key": "selection_strategy", "value": "ppo"}
{"step": 25, "key": "reward", "value": 0.4109064254874037}
{"step": 25, "key": "prompt", "value": "Invent a sport based on telepathic trees."}
{"step": 25, "key": "avg_reward", "value": 0.41827138684195253}
{"step": 25, "key": "episode_done", "value": true}

ДОДАТОК Б

Код програми

```
        #filtering/domain_conditioner
from typing import List

class DomainConditioner:
    def __init__(self, allowed_keywords: List[str]):
        self.allowed_keywords = [kw.lower() for kw in allowed_keywords]

    def filter_examples(self, examples: List[str]) -> List[str]:
        filtered = []
        for ex in examples:
            if any(kw in ex.lower() for kw in self.allowed_keywords):
                filtered.append(ex)
        return filtered

    def is_relevant(self, text: str) -> bool:
        return any(kw in text.lower() for kw in self.allowed_keywords)

        #generators/back_translation_generator.py

from transformers import MarianMTModel, MarianTokenizer
from typing import List

class BackTranslationGenerator:
    def __init__(self,
                 src_lang: str = "en",
                 pivot_lang: str = "de",  # Pivot: German
                 model_src_to_pivot: str = "Helsinki-NLP/opus-mt-en-de",
```

```
                model_pivot_to_src: str = "Helsinki-NLP/opus-mt-de-en",
                max_length: int = 64):
        self.tokenizer_fwd =
MarianTokenizer.from_pretrained(model_src_to_pivot)
        self.model_fwd = MarianMTModel.from_pretrained(model_src_to_pivot)
        self.tokenizer_back =
MarianTokenizer.from_pretrained(model_pivot_to_src)
        self.model_back = MarianMTModel.from_pretrained(model_pivot_to_src)
        self.max_length = max_length

    def generate(self, prompt: str, num_return_sequences: int = 1) -> List[str]:
        # DE
        encoded = self.tokenizer_fwd(prompt, return_tensors="pt",
truncation=True, max_length=self.max_length)
        translated = self.model_fwd.generate(**encoded,
num_return_sequences=num_return_sequences)
        pivot_texts = [self.tokenizer_fwd.decode(t, skip_special_tokens=True) for t
in translated]

        # EN
        results = []
        for pivot in pivot_texts:
            encoded_back = self.tokenizer_back(pivot, return_tensors="pt",
truncation=True, max_length=self.max_length)
            back_translated = self.model_back.generate(**encoded_back)
            back_text = self.tokenizer_back.decode(back_translated[0],
skip_special_tokens=True)
            results.append(back_text)

        return results
```

```
        #generators/base_generator.py

from typing import List
from abc import ABC, abstractmethod

class BaseGenerator(ABC):
    @abstractmethod
    def generate(self, prompt: str, num_return_sequences: int = 1) -> List[str]:

        pass
        #generators/formalization_generator.py

from typing import List
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
import torch

class FormalizationGenerator:
    def __init__(self, model_name="google/flan-t5-small", device=None):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
        self.device = device or ("cuda" if torch.cuda.is_available() else "cpu")
        self.model.to(self.device)

    def generate(self, prompt: str, num_return_sequences: int = 1) -> List[str]:
        input_text = f"Rewrite in formal style: {prompt}"
        inputs = self.tokenizer([input_text], return_tensors="pt", padding=True,
truncation=True).to(self.device)
        outputs = self.model.generate(
            **inputs,
            max_length=128,
```

```
            num_beams=4,
            num_return_sequences=num_return_sequences
        )
        return [self.tokenizer.decode(o, skip_special_tokens=True) for o in outputs]
          #generators/gpt2_generator.py

from transformers import GPT2LMHeadModel, GPT2Tokenizer
from typing import List
import torch

class GPT2Generator:
    def __init__(self, model_name: str = "gpt2", max_length: int = 64):
        self.tokenizer = GPT2Tokenizer.from_pretrained(model_name)
        self.model = GPT2LMHeadModel.from_pretrained(model_name)
        self.max_length = max_length

    def generate(self, prompt: str, num_return_sequences: int = 3) -> List[str]:
        inputs = self.tokenizer(prompt, return_tensors="pt")
        outputs = self.model.generate(
            input_ids=inputs["input_ids"],
            attention_mask=inputs["attention_mask"],
            max_length=self.max_length,
            num_return_sequences=num_return_sequences,
            do_sample=True,
            top_k=50,
            top_p=0.95
        )
        return [self.tokenizer.decode(o, skip_special_tokens=True) for o in outputs]
          from typing import List, Tuple, Dict, Set
from config import config
```

```
import random
import numpy as np
from generators import (
    t5_generator,
    gpt2_generator,
    back_translation_generator,
    style_transfer_generator,
    translation_generator,
    formalization_generator,
)
from rl.agent import PPOAgent
from langdetect import detect, DetectorFactory
import logging
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

DetectorFactory.seed = 0

def is_english(text: str) -> bool:
    try:
        return detect(text) == 'en'
    except Exception as e:
        logging.warning(f"[HybridGenerator] Language detection failed: {e}")
        return False

def deduplicate_similar_texts(texts: List[str], threshold: float = 0.95) ->
List[str]:
    if len(texts) <= 1:
        return texts
```

```
    vectorizer = TfidfVectorizer().fit_transform(texts)
    similarity_matrix = cosine_similarity(vectorizer)

    keep_indices = []
    seen = set()

    for i in range(len(texts)):
        if i in seen:
            continue
        keep_indices.append(i)
        for j in range(i + 1, len(texts)):
            if similarity_matrix[i, j] >= threshold:
                seen.add(j)

    return [texts[i] for i in keep_indices]

class HybridGenerator:
    def __init__(self, agent: PPOAgent = None):
        self.generators_map = {
            "t5_generator": t5_generator.T5Generator(),
            "gpt2_generator": gpt2_generator.GPT2Generator(),
            "back_translation_generator":
back_translation_generator.BackTranslationGenerator(),
            "style_transfer_generator":
style_transfer_generator.StyleTransferGenerator(),
            "translation_generator": translation_generator.TranslationGenerator(),
            "formalization_generator":
formalization_generator.FormalizationGenerator(),
        }
        self.selected_generators = config.generation.generators
```

```
        self.return_all = config.generation.return_all
        self.agent = agent

    def generate(
            self,
            prompt: str,
            state: np.ndarray,
            return_trace: bool = False,
            min_words: int = 5,
            already_generated: Set[str] = None,
            max_generators_sampled: int = 3  # Кількість генераторів на один
епізод
    ) -> Tuple[List[str], Dict[str, str], float]:
        candidates = []
        trace_map = {}

        if already_generated is None:
            already_generated = set()

        # Обчислити ймовірності для вибору генераторів з урахуванням PPO
або випадково
        if self.agent:
            action_probs = self.agent.get_action_probabilities(state)
        else:
            action_probs = np.ones(len(self.selected_generators)) /
len(self.selected_generators)

        # Вибір підмножини генераторів
        try:
            sampled_generator_indices = np.random.choice(
```

```
            len(self.selected_generators),
            size=min(max_generators_sampled, len(self.selected_generators)),
            replace=False,
            p=action_probs
        )
    except ValueError:
        logging.warning("[HybridGenerator] Invalid probability distribution. Falling back to uniform.")
        sampled_generator_indices = np.random.choice(
            len(self.selected_generators),
            size=min(max_generators_sampled, len(self.selected_generators)),
            replace=False
        )

    sampled_generator_names = [self.selected_generators[i] for i in sampled_generator_indices]

    for name in sampled_generator_names:
        generator = self.generators_map[name]
        try:
            samples = generator.generate(prompt, num_return_sequences=config.generation.max_outputs_per_prompt)
            for sample in samples:
                if sample is None:
                    continue
                if len(sample.strip().split()) < min_words:
                    continue
                if not is_english(sample):
                    continue
                if sample in already_generated:
```

```
                continue

            candidates.append(sample)
            trace_map[sample] = name
        except Exception as e:
            logging.warning(f"[HybridGenerator] Generator '{name}' failed: {e}")
            continue

    filtered_candidates = deduplicate_similar_texts(candidates, threshold=0.95)

    if not filtered_candidates:
        fallback = f"Fallback response for: {prompt}"
        trace_map[fallback] = "fallback_generator"
        return ([fallback], trace_map, 0.0)

    action_idx, log_prob = self.agent.select_action(state)
    selected_generator = self.selected_generators[action_idx]

    final_candidates = [s for s in filtered_candidates if trace_map.get(s) ==
selected_generator]

    if not final_candidates:
        fallback = random.choice(filtered_candidates)
        return ([fallback], trace_map, log_prob)

    selected_sample = random.choice(final_candidates)
    return ([selected_sample], trace_map, log_prob) if return_trace else
([selected_sample], {}, log_prob)
      #generators/selection_strategy.py
```

```
import random
from typing import List, Tuple
from scoring.topic_scorer import TopicScorer
from reward.reward_function import RewardFunction


class SelectionStrategy:
    def __init__(self, epsilon: float = 0.1):
        self.epsilon = epsilon
        self.reward_function = RewardFunction()

    def select(self, candidates: List[Tuple[str, str]], reference: str, strategy: str =
"epsilon_greedy", top_k: int = 3) -> List[str]:
        if not candidates:
            return []

        texts = [text for _, text in candidates]

        if strategy == "random":
            return [random.choice(texts)]

        elif strategy == "round_robin":
            return [texts[i % len(texts)] for i in range(min(top_k, len(texts)))]

        elif strategy == "score_top_k":
            scorer = TopicScorer()
            scores = scorer.batch_score(texts, reference)
            scored = list(zip(texts, scores))
            scored.sort(key=lambda x: x[1], reverse=True)
            return [text for text, _ in scored[:top_k]]
```

```
        elif strategy == "epsilon_greedy":
            # Обчислюємо винагороду для кожного кандидата
            rewards = [self.reward_function.compute_reward(text, reference) for text
in texts]

            if random.random() < self.epsilon:
                return [random.choice(texts)]
            else:
                best_idx = int(max(range(len(rewards)), key=lambda i: rewards[i]))
                return [texts[best_idx]]

        elif strategy == "weighted":
            weights = [random.random() for _ in texts]
            selected = random.choices(texts, weights=weights, k=1)
            return selected

        else:
            raise ValueError(f"Unknown selection strategy: {strategy}")
          # generators/style_transfer.py

from typing import List
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
import torch


class StyleTransferGenerator:
    def __init__(self, model_name="prithivida/parrot_paraphraser_on_T5",
device=None):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
        self.model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
        self.device = device or ("cuda" if torch.cuda.is_available() else "cpu")
        self.model.to(self.device)

    def generate(self, prompt: str, num_return_sequences: int = 1) -> List[str]:
        input_text = f"paraphrase: {prompt} </s>"
        inputs = self.tokenizer([input_text], return_tensors="pt", padding=True)
        inputs = {k: v.to(self.device) for k, v in inputs.items()}

        outputs = self.model.generate(
            **inputs,
            max_length=128,
            num_beams=5,
            num_return_sequences=num_return_sequences,
            temperature=1.5
        )
        return [self.tokenizer.decode(o, skip_special_tokens=True) for o in outputs]
          #generators/t5_generator.py

from transformers import T5ForConditionalGeneration, T5Tokenizer
from typing import List

class T5Generator:
    def __init__(self, model_name: str = "t5-small", max_length: int = 64):
        self.tokenizer = T5Tokenizer.from_pretrained(model_name)
        self.model = T5ForConditionalGeneration.from_pretrained(model_name)
        self.max_length = max_length

    def generate(self, prompt: str, num_return_sequences: int = 3) -> List[str]:
        inputs = self.tokenizer(prompt, return_tensors="pt", truncation=True)
```

```
        outputs = self.model.generate(
            input_ids=inputs["input_ids"],
            attention_mask=inputs["attention_mask"],
            max_length=self.max_length,
            num_return_sequences=num_return_sequences,
            do_sample=True,
            top_k=50,
            top_p=0.95
        )
        return [self.tokenizer.decode(o, skip_special_tokens=True) for o in outputs]
          #generators/translation_generator.py

from typing import List
from transformers import MarianMTModel, MarianTokenizer
import torch

class TranslationGenerator:
    def __init__(self, model_name="Helsinki-NLP/opus-mt-en-fr",
device=None):
        self.tokenizer = MarianTokenizer.from_pretrained(model_name)
        self.model = MarianMTModel.from_pretrained(model_name)
        self.device = device or ("cuda" if torch.cuda.is_available() else "cpu")
        self.model.to(self.device)

    def generate(self, prompt: str, num_return_sequences: int = 1) -> List[str]:
        inputs = self.tokenizer([prompt], return_tensors="pt", padding=True,
truncation=True).to(self.device)
        translated = self.model.generate(
            **inputs,
            num_return_sequences=num_return_sequences,
```

```
            max_new_tokens=50,  #  critical
            num_beams=4
        )
        return [self.tokenizer.decode(t, skip_special_tokens=True) for t in
translated]
         #ligging_scr/logger.py

import os
import json
import csv
from typing import Any

class ExperimentLogger:
    def __init__(self, base_dir: str = "logs/"):
        self.base_dir = base_dir
        os.makedirs(self.base_dir, exist_ok=True)
        self.json_log_path = os.path.join(self.base_dir, "log.jsonl")
        self.csv_log_path = os.path.join(self.base_dir, "log.csv")

        # create empty files if they don't exist
        if not os.path.exists(self.json_log_path):
            open(self.json_log_path, 'w', encoding='utf-8').close()
        if not os.path.exists(self.csv_log_path):
            with open(self.csv_log_path, "w", newline='', encoding='utf-8') as f:
                writer = csv.writer(f)
                writer.writerow(["step", "key", "value"])

    def log(self, key: str, value: Any, step: int):
        json_entry = {"step": step, "key": key, "value": value}
        with open(self.json_log_path, "a", encoding='utf-8') as f_json:
```

```
            f_json.write(json.dumps(json_entry, ensure_ascii=False) + "\n")

        with open(self.csv_log_path, "a", newline='', encoding='utf-8') as f_csv:
            writer = csv.writer(f_csv)
            writer.writerow([step, key, value])
          from transformers import pipeline

class LLMInstructionGenerator:
    def __init__(self, model_name="gpt2", max_length=50,
num_return_sequences=1):
        self.generator = pipeline("text-generation", model=model_name)
        self.max_length = max_length
        self.num_return_sequences = num_return_sequences

        self.seeds = [
            "Give a detailed instruction to",
            "Describe a procedure to",
            "Explain how to",
            "Write a prompt asking to",
            "How would you instruct someone to"
        ]

    def generate_synthetic_prompts(self, num_prompts: int):
        prompts = []
        for _ in range(num_prompts):
            seed = random.choice(self.seeds)
            result = self.generator(
                seed,
                max_length=self.max_length,
                num_return_sequences=self.num_return_sequences,
```

```
            do_sample=True,
            temperature=0.9,
            top_k=50
        )
        prompts.append(result[0]['generated_text'].strip())
    return prompts


def generate_synthetic_prompts(num_prompts=5):
    generator = LLMInstructionGenerator()
    return generator.generate_synthetic_prompts(num_prompts)
      import logging
from reward.reward_logger import RewardLogger
from scoring.topic_scorer import TopicScorer
from sentence_transformers import SentenceTransformer, util
import numpy as np

class RewardFunction:
    def __init__(self,
                 use_topic: bool = True,
                 use_novelty: bool = True,
                 use_length_penalty: bool = True,
                 target_length: int = 50,
                 similarity_weight: float = 1.0,
                 novelty_weight: float = 1.0,
                 length_weight: float = 0.1):
        self.topic_scorer = TopicScorer()
        self.semantic_model = SentenceTransformer("paraphrase-MiniLM-L6-v2")
        self.logger = RewardLogger()
```

```
        self.use_topic = use_topic
        self.use_novelty = use_novelty
        self.use_length_penalty = use_length_penalty
        self.target_length = target_length

        self.similarity_weight = similarity_weight
        self.novelty_weight = novelty_weight
        self.length_weight = length_weight

        self.memory = set()
        self.memory_embeddings = []

        # Для згладженого моніторингу винагород
        self.exp_average_reward = 0.0
        self.exp_beta = 0.9

    def _semantic_similarity(self, candidate: str, reference: str) -> float:
        try:
            emb = self.semantic_model.encode([candidate, reference],
convert_to_tensor=True)
            return float(util.pytorch_cos_sim(emb[0], emb[1]))
        except Exception as e:
            logging.warning(f"[RewardFunction] Semantic similarity failed: {e}")
            return 0.0

    def _embedding_novelty(self, candidate: str) -> float:
        try:
            emb = self.semantic_model.encode(candidate, convert_to_tensor=True)
            if not self.memory_embeddings:
                self.memory_embeddings.append(emb)
```

```
                return 1.0
            similarities = util.pytorch_cos_sim(emb, self.memory_embeddings)[0]
            max_sim = float(similarities.max())
            self.memory_embeddings.append(emb)
            return 1.0 - max_sim
        except Exception as e:
            logging.warning(f"[RewardFunction] Embedding novelty failed: {e}")
            return 0.0

    def _jaccard_novelty(self, candidate: str) -> float:
        candidate_set = set(candidate.split())
        if not self.memory:
            return 1.0
        similarities = [
            len(candidate_set & set(prev.split())) / len(candidate_set |
set(prev.split()))
            for prev in self.memory
        ]
        return 1.0 - max(similarities)

    def _length_score(self, candidate: str) -> float:
        length_diff = abs(len(candidate.split()) - self.target_length)
        return np.exp(-length_diff / self.target_length)

    def _add_component(self, name: str, score: float, weight: float,
reward_accumulator: dict):
        contribution = weight * score
        reward_accumulator['reward'] += contribution
        reward_accumulator['logs'][f'{name}_score'] = score
        reward_accumulator['logs'][f'{name}_contribution'] = contribution
```

```
    def compute_reward(self, candidate: str, reference: str, generator_name: str =
"unknown") -> float:
        reward_acc = {'reward': 0.0, 'logs': {'generator': generator_name}}

        # --- Topic + Semantic Similarity ---
        if self.use_topic:
            topic_score = self.topic_scorer.score(candidate, reference)
            semantic_score = self._semantic_similarity(candidate, reference)
            alpha = 0.7 if topic_score < 0.3 and semantic_score > 0.7 else 0.5
            combined_similarity = alpha * topic_score + (1 - alpha) *
semantic_score

            self._add_component('similarity', combined_similarity,
self.similarity_weight, reward_acc)
            reward_acc['logs'].update({
                'topic_score': topic_score,
                'semantic_score': semantic_score,
                'alpha': alpha
            })

        # --- Новизна ---
        if self.use_novelty:
            jaccard_score = self._jaccard_novelty(candidate)
            embedding_score = self._embedding_novelty(candidate)
            combined_novelty = 0.5 * jaccard_score + 0.5 * embedding_score
            self.memory.add(candidate)
            self._add_component('novelty', combined_novelty, self.novelty_weight,
reward_acc)
            reward_acc['logs'].update({
```

```
            'jaccard_novelty': jaccard_score,
            'embedding_novelty': embedding_score,
            'combined_novelty': combined_novelty,
            'novelty_weight_before': self.novelty_weight
        })

        # --- Адаптивне оновлення ваги новизни ---
        novelty_threshold = 0.3
        max_weight = 2.0
        min_weight = 0.1
        adjust_step = 0.05

        if combined_novelty < novelty_threshold:
            self.novelty_weight = min(max_weight, self.novelty_weight +
adjust_step)
        else:
            self.novelty_weight = max(min_weight, self.novelty_weight -
adjust_step * 0.5)

        reward_acc['logs']['novelty_weight_after'] = self.novelty_weight

    # --- Пеналізація за довжину ---
    if self.use_length_penalty:
        length_score = self._length_score(candidate)
        self._add_component('length', length_score, self.length_weight,
reward_acc)

    # --- Згладжене середнє винагород ---
    self.exp_average_reward = (
        self.exp_beta * self.exp_average_reward + (1 - self.exp_beta) *
```

```
reward_acc['reward']
        )

        # --- Логування додаткових метрик ---
        reward_acc['logs']['total_reward'] = reward_acc['reward']
        reward_acc['logs']['exp_average_reward'] = self.exp_average_reward
        reward_acc['logs']['novelty_memory_size'] = len(self.memory)
        reward_acc['logs']['embedding_memory_size'] =
len(self.memory_embeddings)
        reward_acc['logs']['candidate'] = candidate[:80] + '...' if len(candidate) > 80
else candidate

        # --- Вивід логів ---
        logging.info(f"[RewardFunction] Reward breakdown:
{reward_acc['logs']}")
        self.logger.log(reward_acc['logs'], generator_name=generator_name)

        return reward_acc['reward']
          # reward/reward_logger.py
import json
import os
from typing import Optional

class RewardLogger:
    def __init__(self, base_dir: str = "logs/reward_logs/"):
        self.base_dir = base_dir
        os.makedirs(self.base_dir, exist_ok=True)
        self.general_log_path = os.path.join(self.base_dir, "all_rewards.jsonl")

    def log(self, log_data: dict, generator_name: Optional[str] = None):
```

```
        """
        Логує винагороду в загальний файл, а також в окремий файл, якщо вказано генератор.
        """
        # Лог в загальний файл
        with open(self.general_log_path, "a", encoding="utf-8") as f:
            f.write(json.dumps(log_data, ensure_ascii=False) + "\n")

        # Лог в генератор-специфічний файл (опційно)
        if generator_name:
            path = os.path.join(self.base_dir, f"{generator_name}_rewards.jsonl")
            with open(path, "a", encoding="utf-8") as f:
                f.write(json.dumps(log_data, ensure_ascii=False) + "\n")
          import torch.nn as nn
import torch.optim as optim
from typing import List, Dict
import numpy as np
import torch
torch.autograd.set_detect_anomaly(True)


class PPOAgent:
    def __init__(self, state_dim: int, action_dim: int, lr: float = 1e-4, gamma: float = 0.99, clip_epsilon: float = 0.2, update_epochs: int = 4):
        self.state_dim = state_dim
        self.action_dim = action_dim
        self.gamma = gamma
        self.clip_epsilon = clip_epsilon
        self.update_epochs = update_epochs
```

```
        self.actor = self.build_network(output_dim=action_dim)
        self.critic = self.build_network(output_dim=1)
        self.optimizer = optim.Adam(list(self.actor.parameters()) +
list(self.critic.parameters()), lr=lr)

    def build_network(self, output_dim: int):
        return nn.Sequential(
            nn.Linear(self.state_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, output_dim)
        )

    def select_action(self, state: np.ndarray) -> (int, torch.Tensor):
        state_tensor = torch.FloatTensor(state).unsqueeze(0)
        logits = self.actor(state_tensor)
        probs = torch.softmax(logits, dim=-1)
        action_dist = torch.distributions.Categorical(probs)
        action = action_dist.sample()
        return action.item(), action_dist.log_prob(action)

    def compute_advantage(self, rewards, values, next_values, dones):
        advantages = []
        gae = 0
        for i in reversed(range(len(rewards))):
            delta = rewards[i] + self.gamma * next_values[i] * (1 - dones[i]) -
values[i]
            gae = delta + self.gamma * 0.95 * gae * (1 - dones[i])
            advantages.insert(0, gae)
```

```
        return advantages

    def update(self, states, actions, log_probs, rewards, next_states, dones):
        # Перетворення до тензорів
        states_tensor = torch.FloatTensor(np.array(states)).detach().clone()
        next_states_tensor =
torch.FloatTensor(np.array(next_states)).detach().clone()

        # Обчислення values та next_values
        values = self.critic(states_tensor).squeeze().detach().clone()
        next_values = self.critic(next_states_tensor).squeeze().detach().clone()

        # Переконаємось, що `values` та `next_values` є 1D тензорами
        if values.dim() == 0:
            values = values.unsqueeze(0)
        if next_values.dim() == 0:
            next_values = next_values.unsqueeze(0)

        # Від'єднуємо граф обчислень для `values` та `next_values`
        values_detached = values.clone()
        next_values_detached = next_values.clone()

        # Обчислюємо переваги (advantages)
        advantages = self.compute_advantage(
            rewards,
            values_detached.numpy(),
            next_values_detached.numpy(),
            dones
        )
        advantages = torch.FloatTensor(advantages).detach().clone()
```

```
        # Конвертуємо решту даних до тензорів
        log_probs = torch.FloatTensor(log_probs).detach().clone()
        rewards = torch.FloatTensor(rewards).detach().clone()
        actions = torch.LongTensor(actions).detach().clone()

        # Основний цикл оновлення
        for epoch in range(self.update_epochs):
            logits = self.actor(states_tensor)  # без .detach()
            probs = torch.softmax(logits, dim=-1)

            new_log_probs = torch.log(probs.gather(1,
actions.unsqueeze(1)).squeeze())

            ratio = torch.exp(new_log_probs - log_probs)
            surr1 = ratio * advantages
            surr2 = torch.clamp(ratio, 1 - self.clip_epsilon, 1 + self.clip_epsilon) *
advantages
            actor_loss = -torch.min(surr1, surr2).mean()

            values_pred = self.critic(states_tensor).squeeze()
            value_loss = nn.MSELoss()(values_pred, rewards)

            loss = actor_loss + 0.5 * value_loss

            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()

    def save(self, path: str):
```

```
        torch.save({"actor": self.actor.state_dict(), "critic": self.critic.state_dict()},
path)

    def load(self, path: str):
        checkpoint = torch.load(path)
        self.actor.load_state_dict(checkpoint["actor"])
        self.critic.load_state_dict(checkpoint["critic"])

    def get_action_probabilities(self, state: np.ndarray) -> np.ndarray:
        """
        Обчислює ймовірності дій (тобто генераторів) на основі поточного
стану.
        """
        state_tensor = torch.FloatTensor(state).unsqueeze(0)  # shape: [1,
state_dim]
        with torch.no_grad():
            logits = self.actor(state_tensor)  # shape: [1, action_dim]
            probs = torch.softmax(logits, dim=-1).squeeze(0)  # shape: [action_dim]
        return probs.cpu().numpy()
         from generators.hybrid_generator import HybridGenerator
from reward.reward_function import RewardFunction
from training.trainer import StudentTrainer
from rl.agent import PPOAgent
from generators.selection_strategy import SelectionStrategy
from prompt_sources.random_prompt_generator import
RandomPromptGenerator  # <--- нове
from config import config
from logging_scr.logger import ExperimentLogger
import numpy as np
import logging
```

```
class ReinforcementLearningLoop:
    def __init__(self):
        self.agent = PPOAgent(
            state_dim=len(config.generation.generators),
            action_dim=len(config.generation.generators),
            lr=config.rl_agent.lr,
            gamma=config.rl_agent.gamma,
            clip_epsilon=config.rl_agent.epsilon,
            update_epochs=config.rl_agent.update_epochs
        )

        self.generator = HybridGenerator(agent=self.agent)
        self.reward_fn = RewardFunction()
        self.trainer = StudentTrainer()
        self.selector = SelectionStrategy(epsilon=config.rl_agent.epsilon)
        self.logger = ExperimentLogger()
        self.random_prompt_generator = RandomPromptGenerator()  # <--- нове

        self.reward_history = []
        self.epsilon_history = []
        self.generated_texts = set()  # Глобальний контроль унікальності

    def run_episode(self, prompt: str, reference: str, episode: int, min_words: int =
5):
        state = np.ones(len(config.generation.generators))
        max_attempts = 10

        for attempt in range(max_attempts):
            candidates, generation_trace, log_prob = self.generator.generate(
```

```
            prompt=prompt,
            state=state,
            return_trace=True,
            min_words=min_words,
            already_generated=self.generated_texts
        )

        if not candidates:
            logging.warning(f"[RL] No candidates generated in attempt {attempt
+ 1}/{max_attempts} for episode {episode}")
            continue

        candidate = candidates[0]
        chosen_generator_name = generation_trace.get(candidate, "unknown")

        self.generated_texts.add(candidate)

        reward = self.reward_fn.compute_reward(candidate, reference,
generator_name=chosen_generator_name)

        self.logger.log("candidate", candidate, step=episode)
        self.logger.log("selected_generator", chosen_generator_name,
step=episode)
        self.logger.log("selection_strategy", "ppo", step=episode)
        self.logger.log("reward", reward, step=episode)
        self.logger.log("prompt", prompt, step=episode)  # <--- нове

        # Враховує fallback або unknown генератор
        try:
            action_index =
```

```
self.generator.selected_generators.index(chosen_generator_name)
            except ValueError:
                action_index = 0  # Безпечна заміна

            self.agent.update(
                states=[state],
                actions=[action_index],
                log_probs=[log_prob],
                rewards=[reward],
                next_states=[state],
                dones=[False]
            )

            self.reward_history.append(reward)
            avg_reward = np.mean(self.reward_history[-10:])
            self.logger.log("avg_reward", avg_reward, step=episode)
            self.logger.log("episode_done", True, step=episode)
            return  # Успішно завершено

        logging.warning(f"[RL] Skipped episode {episode} after {max_attempts}
failed attempts.")
        self.logger.log("episode_skipped", True, step=episode)

    def run(self, num_episodes: int = 10):
        for episode in range(num_episodes):
            prompt = self.random_prompt_generator.generate_prompt()  # <--- нове
            reference = ""  # <--- тимчасово пусте
            self.run_episode(prompt, reference, episode)
          # scoring/semantic_scorer.py
```

```
from sentence_transformers import SentenceTransformer, util
from typing import List

class SemanticScorer:
    def __init__(self, model_name: str = "all-MiniLM-L6-v2"):
        self.model = SentenceTransformer(model_name)

    def score(self, candidate: str, reference: str) -> float:
        try:
            emb_ref = self.model.encode(reference, convert_to_tensor=True)
            emb_cand = self.model.encode(candidate, convert_to_tensor=True)
            return float(util.cos_sim(emb_ref, emb_cand)[0][0])
        except Exception:
            return 0.0

    def batch_score(self, candidates: List[str], reference: str) -> List[float]:
        try:
            emb_ref = self.model.encode(reference, convert_to_tensor=True)
            emb_cands = self.model.encode(candidates, convert_to_tensor=True)
            sims = util.cos_sim(emb_ref, emb_cands)[0]
            return sims.tolist()
        except Exception:
            return [0.0] * len(candidates)
          #scoring/topic_scorer.py

from sentence_transformers import SentenceTransformer, util
import logging
from typing import List

class TopicScorer:
```

```
    def __init__(self, model_name: str = "all-MiniLM-L6-v2"):
        self.model = SentenceTransformer(model_name)

    def score(self, candidate: str, reference: str) -> float:

        try:
            embeddings = self.model.encode([reference, candidate],
convert_to_tensor=True)
            return float(util.pytorch_cos_sim(embeddings[0], embeddings[1]))
        except Exception as e:
            logging.warning(f"[TopicScorer] Semantic score failed: {e}")
            return 0.0

    def batch_score(self, candidates: List[str], reference: str) -> List[float]:

        try:
            embeddings = self.model.encode([reference] + candidates,
convert_to_tensor=True)
            ref_embedding = embeddings[0].unsqueeze(0)
            candidate_embeddings = embeddings[1:]
            similarities = util.pytorch_cos_sim(ref_embedding,
candidate_embeddings)[0]
            return similarities.cpu().tolist()
        except Exception as e:
            logging.warning(f"[TopicScorer] Batch score failed: {e}")
            return [0.0] * len(candidates)
         from transformers import (
    AutoTokenizer,
    AutoModelForSeq2SeqLM,
    AutoModelForCausalLM,
```

```
    AutoModelForMaskedLM,
    Trainer,
    TrainingArguments,
    DataCollatorForSeq2Seq,
    DataCollatorForLanguageModeling,
    DataCollatorWithPadding
)
import torch
from datasets import Dataset
from typing import List, Tuple

class StudentTrainer:
    def __init__(self, model_name: str = "t5-small"):
        self.model_name = model_name
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model_type = self._detect_model_type(model_name)
        self.model = self._load_model(model_name)

    def _detect_model_type(self, model_name: str) -> str:
        """Розпізнає тип моделі на основі її імені."""
        name = model_name.lower()
        if "t5" in name or "mbart" in name:
            return "seq2seq"
        elif "gpt" in name or "opt" in name or "bloom" in name:
            return "causal"
        elif "bert" in name or "electra" in name:
            return "masked"
        else:
            return "seq2seq"  # за замовчуванням
```

```
    def _load_model(self, model_name: str):
        """Завантажує модель відповідно до її типу."""
        if self.model_type == "seq2seq":
            return AutoModelForSeq2SeqLM.from_pretrained(model_name)
        elif self.model_type == "causal":
            return AutoModelForCausalLM.from_pretrained(model_name)
        elif self.model_type == "masked":
            return AutoModelForMaskedLM.from_pretrained(model_name)
        else:
            raise ValueError(f"Unknown model type for {model_name}")

    def _prepare_dataset(self, data: List[Tuple[str, str]]) -> Dataset:
        examples = [{"input": inp, "target": tgt} for inp, tgt in data]

        def tokenize_seq2seq(example):
            model_inputs = self.tokenizer(
                example["input"], max_length=128, padding="max_length",
truncation=True
            )
            labels = self.tokenizer(
                example["target"], max_length=128, padding="max_length",
truncation=True
            )
            model_inputs["labels"] = labels["input_ids"]
            return model_inputs

        def tokenize_causal(example):
            combined = example["input"] + self.tokenizer.eos_token +
example["target"]
            tokenized = self.tokenizer(combined, max_length=128,
```

```
padding="max_length", truncation=True)
            tokenized["labels"] = tokenized["input_ids"].copy()
            return tokenized

        def tokenize_masked(example):
            tokenized = self.tokenizer(
                example["input"], max_length=128, padding="max_length",
truncation=True
            )
            tokenized["labels"] = self.tokenizer(
                example["target"], max_length=128, padding="max_length",
truncation=True
            )["input_ids"]
            return tokenized

        dataset = Dataset.from_list(examples)

        if self.model_type == "seq2seq":
            return dataset.map(tokenize_seq2seq, remove_columns=["input",
"target"])
        elif self.model_type == "causal":
            return dataset.map(tokenize_causal, remove_columns=["input",
"target"])
        elif self.model_type == "masked":
            return dataset.map(tokenize_masked, remove_columns=["input",
"target"])
        else:
            raise ValueError("Unknown model type")

    def train_on(self, example: str, target: str = "placeholder"):
```

```
        self.train_on_dataset([(example, target)], num_train_epochs=1)

    def train_on_dataset(
        self,
        data: List[Tuple[str, str]],
        output_dir: str = "saved_models/student",
        num_train_epochs: int = 3,
        batch_size: int = 8
    ):
        dataset = self._prepare_dataset(data)

        training_args = TrainingArguments(
            output_dir=output_dir,
            per_device_train_batch_size=batch_size,
            num_train_epochs=num_train_epochs,
            logging_dir=f"{output_dir}/logs",
            logging_steps=10,
            save_strategy="no",
            remove_unused_columns=False
        )

        # Вибір відповідного data collator
        if self.model_type == "seq2seq":
            data_collator = DataCollatorForSeq2Seq(tokenizer=self.tokenizer,
model=self.model)
        elif self.model_type == "causal":
            data_collator =
DataCollatorForLanguageModeling(tokenizer=self.tokenizer, mlm=False)
        elif self.model_type == "masked":
            data_collator =
```

```
DataCollatorForLanguageModeling(tokenizer=self.tokenizer, mlm=True)
        else:
            data_collator = DataCollatorWithPadding(tokenizer=self.tokenizer)

        trainer = Trainer(
            model=self.model,
            args=training_args,
            train_dataset=dataset,
            tokenizer=self.tokenizer,
            data_collator=data_collator
        )

        trainer.train()

    def save_model(self, path: str = "saved_models/student"):
        self.model.save_pretrained(path)

    def load_model(self, path: str = "saved_models/student"):
        self.model = self._load_model(path)
          class Config:
    def __init__(self):
        self.generation = self.Generation()
        self.scoring = self.Scoring()
        self.rl_agent = self.RLAgent()
        self.reward = self.Reward()
        self.training = self.Training()
        self.logging = self.Logging()
        self.filtering = self.Filtering()

    class Generation:
```

```
    generators = [
        "t5_generator",
        "gpt2_generator",
        "back_translation_generator",
        "style_transfer_generator",
        "translation_generator",
        "formalization_generator"
    ]
    hybrid_strategy = "epsilon_greedy"
    return_all = False
    max_outputs_per_prompt = 3
    prompt_source = "random"  # або "gpt2"
    num_prompts = 5

class Scoring:
    topic_reference = "movies, film, review"
    similarity_weight = 0.7
    diversity_weight = 0.3

class RLAgent:
    lr = 1e-4
    gamma = 0.99
    epsilon = 0.2
    update_epochs = 4

class Reward:
    use_bleu = True
    use_novelty = True
    use_length_penalty = False
```

```
        aggregation = "weighted_sum"

    class Training:
        model_name = "t5-small"
        learning_rate = 1e-4
        num_episodes = 100000

    class Logging:
        log_dir = "logs/"
        save_json = True
        save_csv = True
        use_mlflow = False

    class Filtering:
        enable_domain_filter = True
        allowed_keywords = ["movie", "film", "cinema", "review"]
        enabled = False
        domain = "translation"

config = Config()
        from config import config
from rl.loop import ReinforcementLearningLoop

import transformers
import torch
import logging

# Додатково: встановлення флагів для дебагу та усунення шуму логів трансформерів
torch.autograd.set_detect_anomaly(True)
```

```
transformers.logging.set_verbosity_error()

logging.basicConfig(level=logging.INFO)

def main():
    # Повністю ізольований ReinforcementLearningLoop управляє генерацією промптів
    rl_loop = ReinforcementLearningLoop()
    rl_loop.run(num_episodes=config.training.num_episodes)

if __name__ == "__main__":
    main()
```

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	129 с.
	Інші документи	
2.	Презентаційні матеріали	14 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Крамаренко О.О.			Метод синтезу навчальної вибірки для трансформер моделей на базі навчання з підкріпленням	Шифр групи	Код напр./спец.
Перевірив		Шевченко О.Ю.				ІТШІ-21-5	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					