

# ОГЛАВЛЕНИЕ

<b>ВВЕДЕНИЕ</b> . . . . .	<b>2</b>
<b>ГЛАВА 1. ПОДГОТОВКА ДАТАСЕТА DNDD ДЛЯ ОБУЧЕНИЯ МОДЕЛЕЙ</b> . . . . .	<b>3</b>
1.1. ЭМУЛЯЦИЯ ДИАЛОГОВЫХ ВЗАИМОДЕЙСТВИЙ . . . . .	3
1.2. СТРУКТУРИРОВАНИЕ ВХОДНЫХ ДАННЫХ ДЛЯ ОБУЧЕНИЯ	3
1.3. ОПТИМИЗАЦИЯ ДАННЫХ ПОД ПОТРЕБИТЕЛЬСКОЕ ОБО- РУДОВАНИЕ . . . . .	4
<b>ГЛАВА 2. ОБУЧЕНИЕ МОДЕЛИ</b> . . . . .	<b>6</b>
2.1. ПОИСК ОПТИМАЛЬНОЙ МОДЕЛИ . . . . .	6
2.2. ПОИСК ОПТИМАЛЬНЫХ ГИПЕРПАРАМЕТРОВ ДЛЯ МО- ДЕЛИ . . . . .	6
2.2.1. ПОСТАНОВКА ЗАДАЧИ . . . . .	6
2.2.2. АНАЛИЗ РЕЗУЛЬТАТОВ ЭКСПЕРИМЕНТОВ С ГИ- ПЕРПАРАМЕТРАМИ . . . . .	7
2.3. ОБУЧЕНИЕ ИТОГОВОЙ МОДЕЛИ . . . . .	9
<b>ЗАКЛЮЧЕНИЕ</b> . . . . .	<b>13</b>
<b>СПИСОК ЛИТЕРАТУРЫ</b> . . . . .	<b>14</b>
<b>ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ОБРАБОТКИ DNDD</b> . . . .	<b>15</b>
<b>ПРИЛОЖЕНИЕ Б. ПРИМЕР ДИАЛОГА</b> . . . . .	<b>30</b>

# ВВЕДЕНИЕ

Разработка диалоговых моделей является активно развивающейся областью машинного обучения. Использование таких моделей имеет широкий спектр применений, включая чат-ботов, системы FAQ, и различные другие системы, где взаимодействие с пользователем через естественный язык играет важную роль. В игровой индустрии диалоговые модели имеют особое значение, поскольку они способны создавать реалистичные и интерактивные диалоги с неигровыми персонажами, улучшая игровой опыт. Качественные диалоговые модели способны улучшить игровой опыт, создавая более привлекательные и погружающие виртуальные миры.

Целью данной работы является создание эффективной диалоговой модели, способную генерировать качественные ответы на основе образа неигрового персонажа и контекста диалога, обеспечивая более реалистичные и интерактивные диалоги с неигровыми персонажами, на основе датасета DNDD (Dungeon & Dragons Dialogues), специально созданного для данного исследования. В данной работе рассматриваются подготовка датасета для обучения модели, формулирование задачи для моделирования, поиск оптимальной модели и параметров, необходимых для успешного и эффективного обучения.

# **ГЛАВА 1. ПОДГОТОВКА ДАТАСЕТА DNDD ДЛЯ ОБУЧЕНИЯ МОДЕЛЕЙ**

После фазы сбора диалоговых данных и генерации параметров неигровых персонажей, включающих в себя идентификаторы, характеристики, мировоззрение, мотивацию и слабости, следующим логическим этапом становится подготовка собранных данных к процессу обучения модели. Этот процесс включает в себя конкатенацию данных в строковом формате.

## **1.1. ЭМУЛЯЦИЯ ДИАЛОГОВЫХ ВЗАИМОДЕЙСТВИЙ**

Особое внимание следует уделить диалоговым взаимодействиям между неигровым персонажем и игроком. В контексте датасета, где хранятся полные версии диалогов, эмуляция процесса общения игрока с неигровым персонажем требует разбиения истории диалога на подмножества. В этом случае диалог представляет собой серию ходов между игроком и неигровым персонажем, и основной задачей модели является продолжение данного диалога, т.е. совершение следующего хода в диалоге.

При таком подходе модель обучается на основе итеративного процесса диалога, что способствует приближению к более реалистичному моделированию процесса диалога. Это позволяет на каждом этапе оптимизировать процесс обучения для достижения максимально эффективного результата.

## **1.2. СТРУКТУРИРОВАНИЕ ВХОДНЫХ ДАННЫХ ДЛЯ ОБУЧЕНИЯ**

Для оптимизации процесса обучения, входные последовательности, а именно описание неигрового персонажа, история диалогов с игроком, последняя реплика игрока и реплика, которую должна сгенерировать модель, были разделены на сегменты, каждый из которых был помечен соответствующим образом. Такой подход к структурированию входных данных для модели позволяет ясно разделять различные компоненты входных данных, что облегчает задачу модели и способствует более эффективному обучению.

Для обозначения начала диалога используется уникальный идентификатор «EMPTY», который функционирует как сигнал о том, что диалог только

что был инициирован. В силу специфики датасета DNDD, полученного из игр, где неигровые персонажи всегда начинают диалог первыми, было определено, что первая реплика игрока служит активацией диалога, и обозначена она идентификатором «START DIALOGUE». В ходе последующего диалога реплики участников регистрируются в истории диалога с пометками «Player: » и «NPC: », в зависимости от того, кто в данный момент выступает в роли говорящего.

### **1.3. ОПТИМИЗАЦИЯ ДАННЫХ ПОД ПОТРЕБИТЕЛЬСКОЕ ОБОРУДОВАНИЕ**

Учитывая ограниченные вычислительные ресурсы потребительского уровня, включая оперативную память объемом 32 гигабайта и графическую карту NVIDIA GeForce RTX 3090 Ti с 24 гигабайтами памяти, было необходимо ввести определенные ограничения на обрабатываемую историю диалогов. При превышении диалогом лимита в 1024 токенов, для обеспечения управляемости данных самые старые записи в диалоге подлежали удалению. Это позволяло оптимизировать использование доступных вычислительных ресурсов и обеспечивать стабильный процесс обучения моделей.

Также было ограничено максимальное количество диалогов, которых может иметь игрок с одним неигровым персонажем. Это позволяет иметь меньшее, но более разнообразное количество данных.

Итоговые данные выглядят следующим образом. Входная последовательность:

Below is the definition of in-game NPC.

NPC Name: Digby

Alignment: Neutral

Description: A burly, bearded man with a thick accent and a penchant for trapping.

Personality traits: Digby is a bit of a glutton, and often overindulges in food and drink.

Flaws: Digby is motivated by the prospect of making a profit from his trapping.

Motivation: Digby is a gruff, no-nonsense man who is quick to anger and slow to trust. He is a hard worker and is not afraid to get his hands dirty. He is also

a bit of a glutton, and often overindulges in food and drink.

Dialogue history:

Player: START DIALOGUE

NPC: \*burp\* Think I had too much to drink last night. Heh! What am I sayin'?! There's no such thing, says my brothers. Hey, who are you, anyway?

Player query: Who are you?

Respond to player's query based on defined NPC:

Ожидаемый ответ: I'm Digby. I'm a trapper 'round these parts. Me and my brothers catch all sorts of varmints, skin 'em, and sell 'em. Course, it's hard lately now that Emmerich is pokin' 'round.

Детальная реализация обработки датасета с документацией доступна в приложении А.

# ГЛАВА 2. ОБУЧЕНИЕ МОДЕЛИ

## 2.1. ПОИСК ОПТИМАЛЬНОЙ МОДЕЛИ

Генеративные модели обычно имеют большой размер, что создает сложности при исследовании таких моделей. Поэтому важным фактором при выборе модели является соотношение размера и качества. В настоящее время одним из самых сложных датасетов является MMLU [1], который проверяет знания моделей, полученных во время предварительного обучения, на различных задачах. Этот датасет включает задачи с разной степенью сложности, от простых до профессиональных. На данный момент наиболее оптимальной моделью на этом бенчмарке является Flan-T5-XL [2] с 3 миллиардами параметров, имея результат 52.4%. Еще одной моделью, которая может составить ей конкуренцию, является LLAMA-13B [3] с результатом 46.9%, но ее большой размер делает процесс обучения значительно более затратным по сравнению с Flan-T5-XL.

Flan-T5 является моделью семейства T5 [4], добавляющая в дообучающую выборку большое количество инструкций, что позволило значительно улучшить качество модели на новых задачах.

## 2.2. ПОИСК ОПТИМАЛЬНЫХ ГИПЕРПАРАМЕТРОВ ДЛЯ МОДЕЛИ

### 2.2.1. ПОСТАНОВКА ЗАДАЧИ

Из гиперпараметров, значительно влияющих на процесс обучения модели, было выделено три группы:

#### 1. Планировщики скорости обучения:

- константный;
- константный с прогревом;
- линейный;
- косинусный;
- косинусный с перезагрузками;
- полиномиальный;

- обратный квадратный корень.

2. Скорость обучения  $\in \{1 \times 10^{-4}, 2 \times 10^{-4}, \dots, 9 \times 10^{-4}, 1 \times 10^{-3}\}$ .

Оценка качества генерации моделей является сложной задачей и малоисследованной. В данной работе помимо значений функции ошибки на валидационных данных используются метрики Exact Match и MAUVE, позволяющие сравнивать параметры между собой. Модель обучалась с различными гиперпараметрами в группе, пока остальные параметры фиксировались.

Метрика Exact Match показывает, какой процент фраз при генерации совпал с ожидаемыми, а MAUVE подсчитывает то, насколько совпало распределение вероятностей сгенерированных фраз с распределением вероятностей ожидаемых фраз.

### 2.2.2. АНАЛИЗ РЕЗУЛЬТАТОВ ЭКСПЕРИМЕНТОВ С ГИПЕРПАРАМЕТРАМИ

При стартовой скорости обучения равной  $1 \times 10^{-3}$  на ограниченном наборе данных были произведены эксперименты по поиску оптимального планировщика скорости обучения. В процессе экспериментов отмечается, что планировщик с обратным квадратным корнем не представлен на графиках, так как ни один из запусков эксперимента с использованием этого планировщика не был успешно завершен.

В ходе экспериментов большинство планировщиков не оказало заметного влияния на скорость обучения и метрики. Среди рассмотренных вариантов планировщиков, в среднем наилучшие результаты продемонстрировал константный планировщик. Наименее эффективным, но успешно завершившим процесс обучения, оказался линейный планировщик. Отмечается, что линейный планировщик характеризуется низким начальным значением функции ошибки на тренировочных и показывает наихудшие конечные значения на метрике MAUVE, что иллюстрируется на рисунках 2.1 и 2.4. График изменения скорости обучения представлен на рисунке 2.5. Ход экспериментов можно наблюдать на рисунках 2.1, 2.2, 2.3, 2.4

В следующих экспериментах при зафиксированном константном планировщике скорости обучения искалась наиболее эффективная скорость обучения. Стоит отметить, что при скорости обучения равной  $1 \times 10^{-4}$  процесс обу-

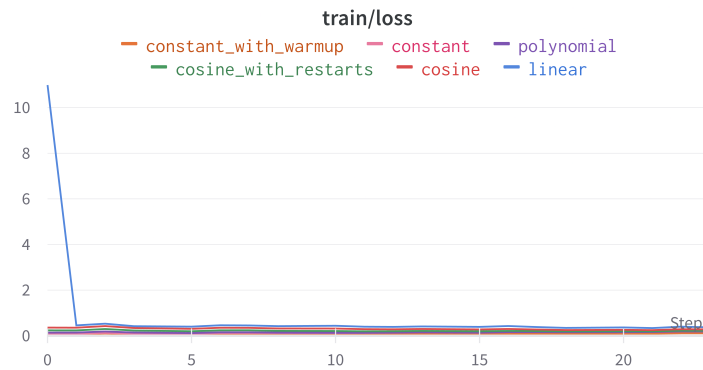


Рисунок 2.1 – Значение функции ошибки на тренировочных данных

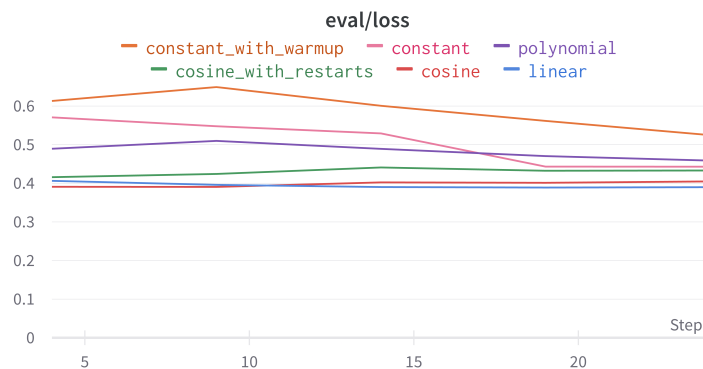


Рисунок 2.2 – Значение функции ошибки на валидационных данных

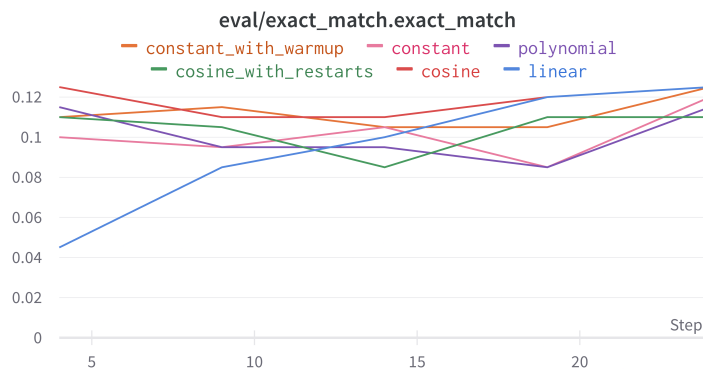


Рисунок 2.3 – Значение метрики Exact Match на валидационных данных

чения не завершился успешно. Из рисунков 2.6, 2.7, 2.8, 2.9 видно, что значения, близкие к  $4 \times 10^{-4}$  и к  $9 \times 10^{-4}$  показывают лучшие значения функций ошибок на всех выборках и лучшие значения метрик. Значение скорости обучения  $9 \times 10^{-4}$  показывает результаты чуть лучше, чем  $4 \times 10^{-4}$ , быстрее достигая лучших значений. В целом, почти все значения скорости обучения показывают схожие результаты, но выбор оптимальных параметров для обучения на большей выборке может сказаться на качестве модели.



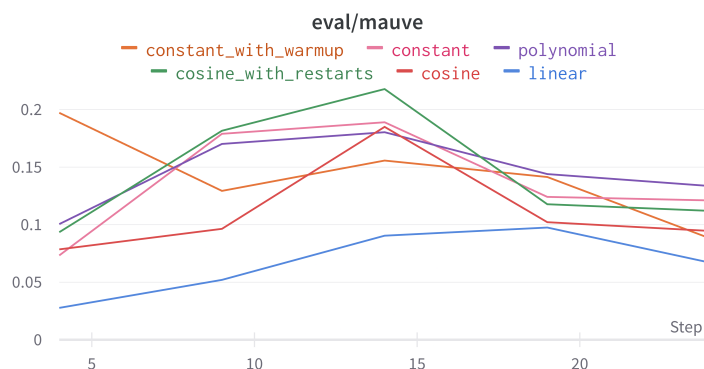


Рисунок 2.4 – Значение метрики MAUVE на валидационных данных



Рисунок 2.5 – График изменения скорости обучения

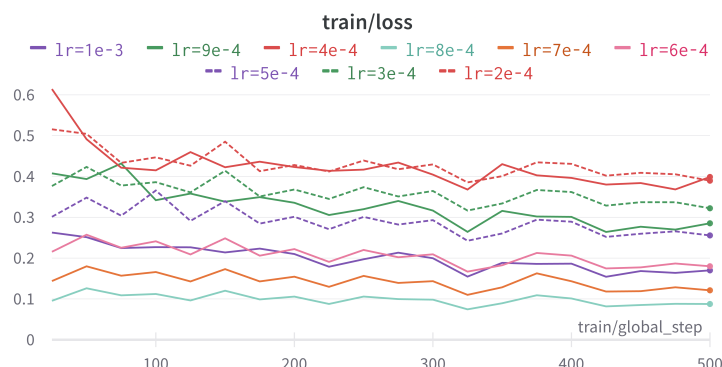


Рисунок 2.6 – Значение функции ошибки на тренировочных данных

Исходя из всех экспериментов можно сделать вывод, что оптимальные параметры для обучения будут константный планировщик скорости обучения и скорость обучения со значением  $9 \times 10^{-4}$

### 2.3. ОБУЧЕНИЕ ИТОГОВОЙ МОДЕЛИ

С подобранными ранее параметрами на была обучена итоговая модель. Общее количество операций, произведенных во время обучения, составило

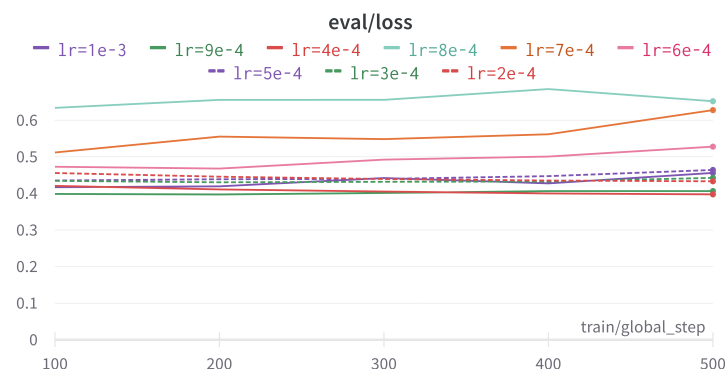


Рисунок 2.7 – Значение функции ошибки на валидационных данных

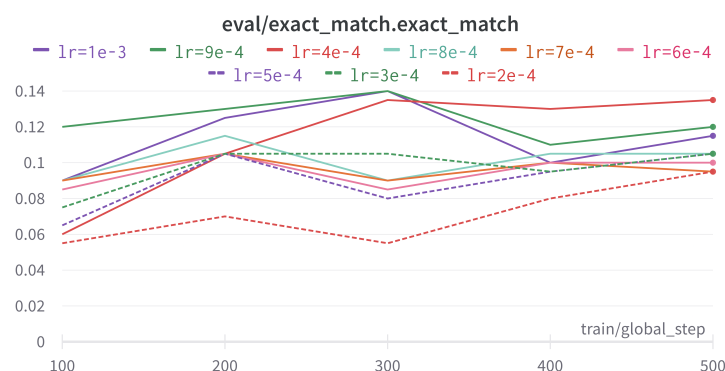


Рисунок 2.8 – Значение метрики Exact Match на валидационных данных

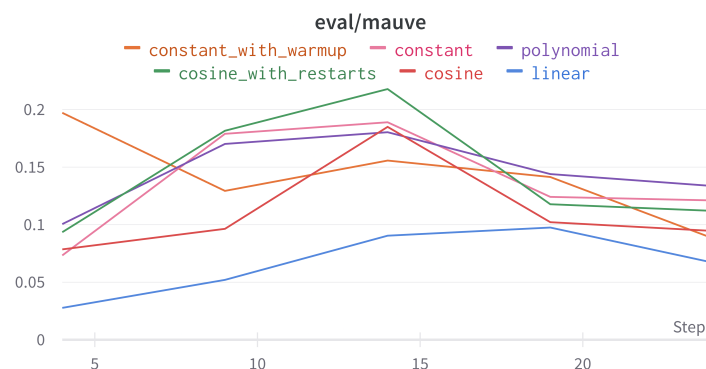


Рисунок 2.9 – Значение метрики MAUVE на валидационных данных

$2 \times 10^{18}$ . Количество токенов, которые фигурировали в процессе обучения –  $25 \times 10^6$ . Процесс обучения виден на рисунках 2.10, 2.11, 2.12, 2.13. Низкие значения метрик Exact Match и MAUVE можно объяснить сложностью поставленной модели задачи: в диалогах часто ответы формируются исходя из внешних условий, в которых происходил диалог с неигровым персонажем, которые сложно получить из данных игры в формате естественного языка. Метрика Exact Match довольно грубо оценивает результат генерации – переформулированная фраза в такой оценке даст значение 0. Тем не менее, такую систему по-

лучилось обучить на потребительском оборудовании на неплохие результаты. Далее идет пример диалога, который был произведен с моделью.

Below is the definition of in-game NPC.

The Mad Lord

Alignment: Chaotic Neutral

Description: A mysterious figure who resides in a castle called Caste Maluradek in the middle of a forest. He is a powerful wizard who has the ability to manipulate the elements and create illusions.

Personality traits: He is obsessed with power and will stop at nothing to achieve his goals.

Flaws: He wants to prove that he is the most powerful wizard in the world. Motivation: The Mad Lord is a mysterious figure who is driven by his desire for power. He is a master manipulator and will use any means necessary to achieve his goals. He is a powerful wizard who is not afraid to use his magic to get what he wants. He is also a bit of a showman, as he enjoys creating elaborate illusions to impress his guests.

Dialogue history:

Player: START DIALOGUE

NPC: Salutations to the travelers. Welcome to Castle Maluradek. I am your adversary.

Player query: Does the adversary have a name?

Respond to player's query based on defined NPC:

ANSWER: I do not have a name. I am a practitioner of magic. I work in the fields of the great forest.

Больше примеров можно увидеть в приложении Б.



Рисунок 2.10 – Значение функции ошибки на тренировочных данных

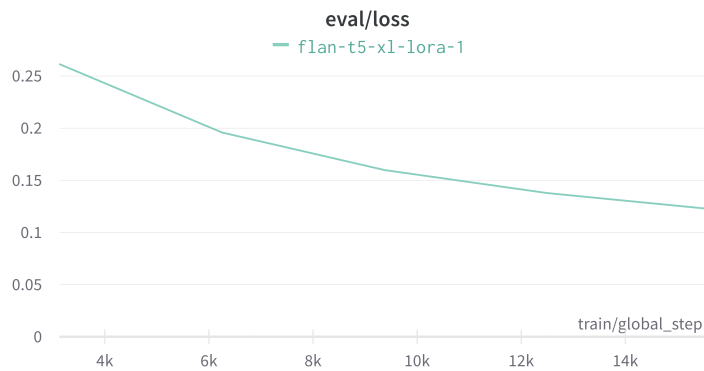


Рисунок 2.11 – Значение функции ошибки на валидационных данных

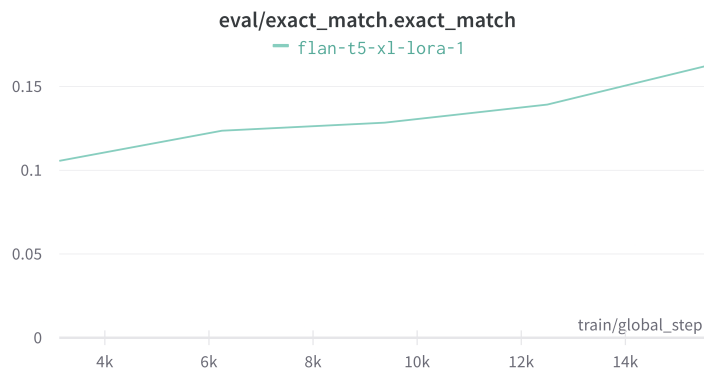


Рисунок 2.12 – Значение метрики Exact Match на валидационных данных

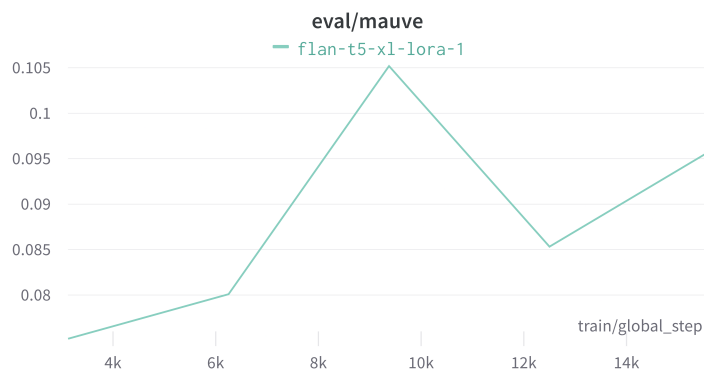


Рисунок 2.13 – Значение метрики MAUVE на валидационных данных

## ЗАКЛЮЧЕНИЕ

В рамках данного исследования была поставлена цель разработки эффективной диалоговой модели, способной генерировать качественные ответы на основе образа неигрового персонажа и контекста диалога в игровой индустрии. Был использован специально созданный для исследования датасет DNDD (Dungeon & Dragons Dialogues). И подготовлен специально для эмулирования диалогов в играх. В процессе экспериментов были рассмотрены различные параметры и планировщики скорости обучения.

На основе проведенных экспериментов можно сделать вывод о наилучшем выборе параметров для обучения модели. Для модели Flan-T5 Был выявлен оптимальный планировщик скорости обучения - константный планировщик, а оптимальное значение скорости обучения составляет  $9 \times 10^{-4}$ . Это сочетание показало лучшие результаты по функциям ошибок и метрикам на представленных наборах данных.

Однако, следует отметить, что введенная сложность задачи диалоговой моделирования в игровой индустрии, где ответы зависят от различных условий и контекста диалога, может быть причиной низких значений метрик Exact Match и MAUVE. Оценка Exact Match грубо оценивает результат генерации, причем даже переформулировка фразы может привести к низким значениям.

В целом, полученные результаты демонстрируют возможность обучения эффективной диалоговой модели на доступных вычислительных ресурсах. Однако, дальнейшие исследования и улучшения в области диалоговых моделей могут привести к более точным и качественным результатам.

## СПИСОК ЛИТЕРАТУРЫ

1. Бенчмарк MMLU [Электронный ресурс]: <https://paperswithcode.com/sota/multi-task-language-understanding-on-mmlu>
2. Chung H. W. et al. Scaling instruction-finetuned language models //arXiv preprint arXiv:2210.11416. – 2022.
3. Touvron H. et al. Llama: Open and efficient foundation language models //arXiv preprint arXiv:2302.13971. – 2023.
4. Raffel C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer //The Journal of Machine Learning Research. – 2020. – Т. 21. – №. 1. – С. 5485-5551.

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ОБРАБОТКИ DNDD

```
"""
This script processes a dataset of Dungeons & Dragons (D&D) dialogues.

Example:
    $ python prepare.py --subsets all --generate_descriptions
      --description_file descriptions.csv --limit_dialogues 25
"""

from datetime import datetime
import random
import requests
from transformers import LlamaTokenizer, AutoTokenizer

from dialogue_data import (
    collect_and_prepare_dialogue_data,
)
from descriptions import (
    generate_descriptions,
    build_dataset,
    generate_file_paths,
    save_dataset_subsets,
    add_descriptions_to_dataset,
)
from arg_parser import create_arg_parser


def main(args):
    data_directory = "DNDD_ver0.5"
    execution_timestamp = datetime.now().strftime("%Y-%m-%d-%H-%M%S")
    dnd_dataset = collect_and_prepare_dialogue_data(data_directory,
                                                    args.limit_dialogues)

    if args.generate_descriptions:
        generate_descriptions(dnd_dataset, tokenizer, execution_timestamp,
                              args.model_server_url)

    if args.description_file:
        dnd_dataset = add_descriptions_to_dataset(dnd_dataset,
                                                  args.description_file)

    if args.build_final:
        t5_tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-xl")
```

```

        final_dataset = build_dataset(dnd_dataset.to_pandas(), t5_tokenizer)
        save_filepath =
            f"dndd-max_d{args.limit_dialogues}-{execution_timestamp}.parquet"
        final_dataset.to_parquet(save_filepath)
        print(f"Saved to {save_filepath}")

subset_map = generate_file_paths(execution_timestamp, args.limit_dialogues)
save_dataset_subsets(dnd_dataset, args.subsets, subset_map)

if __name__ == "__main__":
    random.seed(42)
    parser = create_arg_parser()
    args = parser.parse_args()

    if isinstance(args.subsets, str):
        subsets = [args.subsets]
        args.subsets = subsets

    if args.generate_descriptions:
        tokenizer = LlamaTokenizer.from_pretrained(args.llama_base_model)
        try:
            status_code = requests.get(args.model_server_url).status_code
            if status_code != 200:
                raise Exception(f"Model server returned status code {status_code}")
        except requests.exceptions.RequestException as ex:
            print(f"Could not connect to the model server: {ex}")
            raise ex

main(args)

```

### Листинг A.1: Файл prepare.py

```

import argparse

def create_arg_parser():
    parser = argparse.ArgumentParser(
        description="Process subsets of DNDD (Dungeon & Dragons Dialogues) dataset"
    )
    parser.add_argument(
        "--subsets",
        nargs="+",
        required=False,
        default="all",
    )

```



```

        help="Subsets of the dataset to process (bg1, bg2, id1, pst, all) .\n
        DEFAULT: all",
    )
    parser.add_argument(
        "--generate_descriptions",
        required=False,
        default=False,
        action=argparse.BooleanOptionalAction,
        help="Generates a description to NPCs using Alpaca-LoRA-13B in format\n
        Name/Alignment/Description/Flaw/Motivation/Personality", # noqa:
        E501
    )
    parser.add_argument(
        "--description_file",
        required=False,
        default=False,
        help="Adds NPCs' description to dataset in format\n
        Name/Alignment/Description/Flaw/Motivation/Personality",
    )
    parser.add_argument(
        "--limit_dialogues",
        required=False,
        default=None,
        type=int,
        help="Limits the number of dialogues that NPC can have.",
    )
    parser.add_argument(
        "--llama_base_model",
        default="decapoda-research/llama-13b-hf",
        help="The name of the base model to use.",
    )
    parser.add_argument(
        "--model_server_url",
        default="http://127.0.0.1:7860",
        help="The URL of the model server.",
    )
    parser.add_argument(
        "--generate_descriptions", action="store_true", help="Whether to\n
        generate descriptions."
    )
    parser.add_argument(
        "--build-final",
        action="store_false",
        help="Builds the final training dataset.",
    )

    return parser

```

Листинг А.2: Файл arg\_parser.py

```

import requests
from typing import Dict, Tuple, List
from tqdm import tqdm
from datasets import Dataset
from transformers import LlamaTokenizer, PreTrainedTokenizer
import pandas as pd

from utils import (
    format_dialogue_history,
    format_prompt,
    tokenize_check_overflow,
    extract_text,
    default_text,
)

def generate_descriptions(
    dnd_dataset: Dataset,
    tokenizer: LlamaTokenizer,
    execution_timestamp: str,
    model_server_url: str,
) -> None:
    """
    Generate descriptions for NPCs based on their dialogues in the given
    dataset.

    Parameters
    -----
    dnd_dataset : 'Dataset'
        The dataset containing NPC dialogues.
    tokenizer : 'LlamaTokenizer'
        The tokenizer for text encoding.
    execution_timestamp : 'str'
        The timestamp for execution.
    model_server_url : 'str'
        The URL of the model server for generating descriptions.
    """

    grouped_by_filename: pd.DataFrame = (
        dnd_dataset.to_pandas().groupby("filename").agg(({
            "npc_turns": list,
            "player_turns": list
        }))
    )

    # Instruction text for the model
    instruction_text = "Create the personality of a single NPC in DnD style, based on the provided example dialogue in JSON format. Answer in format"

```

```

        Name/Alignment/Description/Flaw/Motivation/Personality_in_a_list_format_
        written_in_third_person." # noqa: E501
header_text = '"NPC-turns":'

instruction_len = len(tokenizer.tokenize(instruction_text))
header_len = len(tokenizer.tokenize(header_text))

for filename, dialogues in tqdm(grouped_by_filename["npc_turns"].items(),
    "Generating_data"):
    dialogue_turns = process_dialogues(dialogues, tokenizer,
        instruction_len, header_len)

    input_text = header_text + str(dialogue_turns)
    response = make_prediction(
        model_server_url,
        instruction_text=instruction_text,
        input_text=input_text,
    )

    generated_description = response["data"][0]
    # Replacing newline characters with "p" to avoid corrupting CSV file
    generated_description = generated_description.replace("\n", "p")
    description_filename = f"descriptions-{execution_timestamp}.csv"
    with open(description_filename, mode="a", encoding="utf-8") as file:
        file.write(f"{filename},{generated_description}\n")

def make_prediction(
    model_server_url: str,
    instruction_text: str,
    input_text: str,
    temperature: float = 0.1,
    top_p: float = 0.2,
    top_k: int = 100,
    num_beams: int = 1,
    max_new_tokens: int = 512,
    streaming_opt: bool = False,
) -> dict:
    """
    Make a prediction using the API endpoint.

    Parameters
    -----
    model_server_url : 'str'
        The URL of the model server.
    instruction_text : 'str'
        Instruction text to LLM.
    input_text : 'str'

```

```

        Input text containing example NPC dialogues.
temperature : 'float', default=0.1
    Sampling temperature.
top_p : 'float', default=0.2
    Top p sampling value.
top_k : 'int', default=100
    Top k sampling value.
num_beams : 'int', default=1
    Number of beams for beam search.
max_new_tokens : 'int', default=512
    Maximum number of new tokens for the output.
streaming_opt : 'bool', default=False
    Option for streaming LLM output.

```

Returns

-----

dict

The prediction response.

"""

```

params = [
    instruction_text,
    input_text,
    temperature,
    top_p,
    top_k,
    num_beams,
    max_new_tokens,
    streaming_opt,
]

response = requests.post(
    f"{model_server_url}/run/predict",
    json={"data": params},
).json()

return response

```

```

def build_dataset(dndd_df: pd.DataFrame, tokenizer: PreTrainedTokenizer) ->
Dataset:
    """
    Builds a training dataset from the provided DNDD dataset.

    Parameters
    -----
    dndd_df : 'pd.DataFrame'
        The DNDD dataset DataFrame.

```

```

tokenizer : 'PreTrainedTokenizer'
    The tokenizer used for tokenization.

Returns
-----
dict
    A dictionary containing the 'source' and 'target' lists representing
    the training dataset.
"""

dataset_dict = {"source": [], "target": []}

for npc_data in tqdm(dnidd_df.itertuples(), "Processing dataset"):
    dialogue_history = []
    query = "START_DIALOGUE"

    total_turns = len(npc_data.player_turns) + len(npc_data.npc_turns)
    for turn_index in range(total_turns):
        if turn_index % 2 == 0:
            target = npc_data.npc_turns[turn_index // 2]

            formatted_history = (
                format_dialogue_history(dialogue_history, npc_data.game)
                if dialogue_history
                else "EMPTY"
            )
            npc_prompt = format_prompt(npc_data, formatted_history, query)
            formatted_history, dialogue_history = tokenize_check_overflow(
                tokenizer,
                npc_prompt,
                formatted_history,
                dialogue_history,
                npc_data.game,
            )

            if npc_data.game == "pst":
                if query != "START_DIALOGUE":
                    query = extract_text(query, default_text("Player"))
                    target = extract_text(target, default_text("NPC"))

            npc_prompt = format_prompt(npc_data, formatted_history, query)
            dataset_dict["source"].append(npc_prompt)
            dataset_dict["target"].append(target)
        else:
            dialogue_history.append(query)
            dialogue_history.append(target)
            query = npc_data.player_turns[turn_index // 2]

```

```

return Dataset.from_dict(dataset_dict)

def process_dialogues(
    dialogues: List[List[str]],
    tokenizer: LlamaTokenizer,
    instruction_len: int,
    header_len: int,
) -> List[str]:
    """
    Process dialogues by removing duplicates and ensuring the length of
    tokenized text is within limit.

    Parameters
    -----
    dialogues : 'List[List[str]]'
        List of dialogues, each dialogue is a list of turns.
    tokenizer : 'LlamaTokenizer'
        Tokenizer to be used.
    instruction_len : 'int'
        Length of instruction text.
    header_len : 'int'
        Length of header text.

    Returns
    -----
    'List[str]'
        Processed dialogue turns.
    """

    dialogue_turns: List[str] = []
    for dialogue in dialogues:
        for turn in dialogue:
            dialogue_turns.append(turn)
            # Removing any duplicate turns from the dialogue
            dialogue_turns = list(set(dialogue_turns))

            # Checking if the length of the tokenized dialogue_turns and
            # instruction texts are within the limit
            prompt_len = len(tokenizer.tokenize(str(dialogue_turns))) +
                instruction_len + header_len
            if prompt_len > 500:
                print(prompt_len)
                dialogue_turns = dialogue_turns[:-1]
                break
            else:
                continue
        break

```

```

    return dialogue_turns

def generate_file_paths(
    execution_timestamp: str, limit_dialogues: int
) -> Dict[str, Tuple[str, str]]:
    """
    Generates file paths for subsets of a dataset based on the execution
    timestamp and limit of dialogues.

    Parameters
    -----
    execution_timestamp : 'str'
        The timestamp of the execution.
    limit_dialogues : 'int'
        The maximum number of dialogues allowed in the subsets.

    Returns
    -----
    'dict'
        A dictionary mapping subset names to their corresponding file paths.
    """

    file_prefix = "data/dndd_subset_"
    file_suffix = f"_{execution_timestamp}_max-d_{limit_dialogues}.parquet"
    subset_map = {
        "all": ("", f"{file_prefix}all{file_suffix}"),
        "bg1": ("bg1", f"{file_prefix}bg1{file_suffix}"),
        "bg2": ("bg2", f"{file_prefix}bg2{file_suffix}"),
        "id1": ("id1", f"{file_prefix}id1{file_suffix}"),
        "pst": ("pst", f"{file_prefix}pst{file_suffix}"),
    }
    return subset_map

def save_dataset_subsets(
    dnd_dataset: Dataset, subsets: List[str], subset_map: Dict[str, Tuple[str,
    str]]
) -> None:
    """
    Saves subsets of a dataset based on specified subsets and their
    corresponding paths.

    Parameters
    -----
    dnd_dataset : 'Dataset'
        The dataset to save subsets from.
    subsets : 'List[str]'

```

```

    A list of subset names to save.
subset_map : 'Dict[str, Tuple[str, str]]'
    A dictionary mapping subset names to their corresponding paths.
"""

for subset in subsets:
    subset_prefix, subset_file_path = subset_map[subset]
    subset_dndd = dnd_dataset.filter(lambda example:
        example["game"].startswith(subset_prefix))
    subset_dndd.to_parquet(subset_file_path)
    print(f"Saved to {subset_file_path}")

def add_descriptions_to_dataset(dnd_dataset: Dataset, description_file: str) ->
Dataset:
    """
    Adds descriptions to a dataset by merging it with a description file.

    Parameters
    -----
    dnd_dataset : 'Dataset'
        The dataset to which descriptions will be added.
    description_file : 'str'
        The path to the description file.

    Returns
    -----
    'Dataset'
        The updated dataset with descriptions.
    """

    dndd_df = dnd_dataset.to_pandas()
    desc_df = pd.read_csv(description_file, sep="|")
    dndd_df_merged = pd.merge(dndd_df, desc_df, on="filename")
    dnd_dataset = Dataset.from_pandas(dndd_df_merged)
    return dnd_dataset

```

### Листинг А.3: Файл descriptions.py

```

import json
import os
import random
from typing import Union
from datasets import Dataset
from tqdm import tqdm

def identify_game(dir: str) -> Union[str, None]:

```



```

"""
Identify the game based on the directory name.

Parameters
-----
dir : 'str'
    The directory name.

Returns
-----
'Union[str, None]'
    The game identifier.
"""
game_identifiers = ["pst", "idl", "bg1", "bg2"]

for game in game_identifiers:
    if game in dir:
        return game

return None


def load_dialogues_from_file(path: str, limit: Union[int, None] = None) -> list:
    """
    Load dialogues from a file.

    Parameters
    -----
    path : 'str'
        The path of the file.
    limit : 'Union[int, None]', default=None
        The maximum number of dialogues to load.

    Returns
    -----
    content:
        The list of dialogues.
    """

    with open(path) as file:
        content = json.load(file)
    if limit:
        content = random.sample(content, limit)
    return content


def collect_and_prepare_dialogue_data(data_directory: str, limit: Union[int,
None]) -> Dataset:

```

```

"""
Collects and prepares dialogue data from multiple files within a directory.
The collected data is converted into a Hugging Face Dataset object.

Parameters
-----
data_directory : 'str'
    The base directory containing the files to read data from.
limit : 'Union[int, None]'
    The maximum number of dialogues to load, or None for no limit.

Returns
-----
'Dataset'
    A Hugging Face Dataset object containing the collected dialogue data.
"""

dialogue_data = []
for directory in os.listdir(data_directory):
    files = os.listdir(os.path.join(data_directory, directory))
    game = identify_game(directory)
    for filename in tqdm(files, f"Processing files in {directory}"):
        dialogues = load_dialogues_from_file(
            os.path.join(data_directory, directory, filename), limit
        )
        for dialogue in dialogues:
            dialogue["filename"] = filename
            dialogue["game"] = game
            dialogue_data.append(dialogue)

dnd_dataset = Dataset.from_list(dialogue_data)
dnd_dataset = dnd_dataset.rename_columns(
    {"HeroSpeech": "player_turns", "CharacterSpeech": "npc_turns"}
)
dnd_dataset = dnd_dataset.remove_columns("Id")
return dnd_dataset

```

Листинг А.4: Файл dialogue\_data.py

```

import re
from typing import List, Tuple
import pandas as pd
from transformers import PreTrainedTokenizer

def format_dialogue_history(dialogue_history: List[str], game_type: str) -> str:
    """
    Formats the dialogue history into a readable format.

```

```

Parameters
-----
dialogue_history : 'List[str]'
    A list containing the dialogue history. Each item is a dialogue string.
game_type : 'str'
    A string representing the type of the game. If 'pst', special
    formatting is applied.

Returns
-----
'str'
    The formatted dialogue history. Each turn is on a new line with the
    format 'Speaker: Dialogue turn'.
"""
formatted_history = ""
for turn_index in range(len(dialogue_history)):
    speaker = "Player" if turn_index % 2 == 0 else "NPC"
    dialogue_turn = dialogue_history[turn_index]
    if game_type == "pst" and dialogue_turn != "START_DIALOGUE":
        dialogue_turn = dialogue_turn.replace("\\r", "").replace("\\n", "")
        extracted_text = re.findall('"(^[^"]*)"', dialogue_turn)
        extracted_text = "".join(extracted_text) if extracted_text else
            default_text(speaker)
        formatted_history += f"{speaker}:_{extracted_text}\\n"
    else:
        formatted_history += f"{speaker}:_{dialogue_turn}\\n"
return formatted_history

def format_prompt(npc_data: pd.Series, current_history: str, query: str) -> str:
    npc_prompt = f""""Below is the definition of in-game NPC.
NPC Name: {npc_data['name']}
Alignment: {npc_data['alignment']}
Description: {npc_data['description']}
Personality traits: {npc_data['personality']}
Flaws: {npc_data['flaw']}
Motivation: {npc_data['motivation']}

Dialogue history:
{current_history}
Player query: {query}

Respond to player's query based on defined NPC: """"
    return npc_prompt

def tokenize_check_overflow(

```

```

tokenizer: PreTrainedTokenizer,
npc_prompt: str,
current_history: str,
dialogue_history: List[str],
game: str,
) -> Tuple[str, List[str]]:
    """
    Truncates the dialogue history to avoid tokenization overflow.

    Parameters
    -----
    tokenizer : 'PreTrainedTokenizer'
        The tokenizer object used for tokenization.
    npc_prompt : 'str'
        The NPC prompt or instruction text.
    current_history : 'str'
        The current dialogue history.
    dialogue_history : 'List[str]'
        The list of previous dialogue turns.
    game : 'str'
        The game identifier.

    Returns
    -----
    'Tuple[str, List[str]]'
        A tuple containing the updated current history and dialogue history.
    """

    prompt_tokens = tokenizer.tokenize(npc_prompt)
    history_tokens = tokenizer.tokenize(current_history)
    total_tokens = len(prompt_tokens) + len(history_tokens)
    while total_tokens > 1024:
        dialogue_history = dialogue_history[2:]
        current_history = format_dialogue_history(dialogue_history, game)
        history_tokens = tokenizer.tokenize(current_history)
        total_tokens = len(prompt_tokens) + len(history_tokens)

    return current_history, dialogue_history


def extract_text(dialogue, default_text):
    extracted_text = re.findall('"([^\"]*)"', dialogue)
    return "".join(extracted_text) if extracted_text else default_text


def default_text(speaker):

```

```
return "Ignore." if speaker == "Player" else "That NPC seems to be ignoring you."
```

Листинг A.5: Файл `utils.py`

## ПРИЛОЖЕНИЕ Б. ПРИМЕР ДИАЛОГА

Below is the definition of in-game NPC.

NPC Name: Saablic Tan

Alignment: Neutral

Description: A tall, gaunt figure with a long, white beard and a hooded cloak.

Personality traits: Overconfidence

Flaws: To protect his creations from harm

Motivation: Saablic is a proud and confident individual, but he is also a bit arrogant and overconfident. He is willing to take risks and is not afraid to speak his mind. He is also very protective of his creations and will do anything to protect them.

Dialogue history:

Player: START DIALOGUE

NPC: Proceed... no further, traveler, as... I would... speak with you.

Player: I'm sorry, but I really must be on my way.

NPC: Know this... traveler. Without... my knowledge... your continued existence... within this world... will be limited.

Player: All right, I'm listening.

NPC: My name... is... was Saablic... Tan. Of my... mistakes... one grew tired... used my precious... magics to make... that which... I am. Having... much confusion as... days move on. Must... mm... tell you I... am frightened.

Player query: Why are you frightened?

Respond to player's query based on defined NPC:

ANSWER: Orogas that... roam these passages... my passages... are my... creation. They... have much contempt... for me. I cannot... leave. My form... my home... punishment for my... ambition. I... derived pleasure... for altering against... their will. I have... much regret.

### Листинг Б.1: Пример диалога