# Содержание

# Аннотация

Наша работа является одним из этапов проекта «NL2ML», целью которого является генерация кода по текстовому описанию задачи машинного обучения. Это актуальная задача, призванная упростить внедрение стандартных моделей машинного обучения в обиход, а также ускорить разработку более сложных моделей. В проекте используются семантические классы фрагментов кода как промежуточный этап между текстовым описанием задачи и кодом. Мы установили, что обучение с подкреплением, а именно метод PPO [19], с использованием недифференцируемых метрик в качестве наград существенно улучшает качество модели-генератора последовательности семантических классов по сравнению с предобучением на основе кросс-энтропийной функции потерь. Также была реализована модель, предсказывающая качество сгенерированной последовательности семантических классов.

# Ключевые слова

Обучение с подкреплением, PPO, предметно-ориентированный язык (DSL), генерация кода, обработка естественного языка

# 1 Введение

## 1.1 Описание предметной области и актуальность задачи

В настоящее время машинное обучение применяется для решения всё большего круга задач в технических, естественных и гуманитарных науках, а также в бизнесе. Людям, не являющимся специалистами в области машинного обучения, проще всего сформулировать такие задачи естественным языком и поручить специалистам написание кода для анализа данных, подбора моделей и гиперпараметров и самого обучения. При этом нейронные сети демонстрируют способности как к пониманию естественного языка, так и к генерации кода на различных языках программирования. Поэтому приобретает актуальность использование искусственного интеллекта для генерации кода, решающего задачу машинного обучения, по текстовому описанию этой задачи. Это позволит решать типовые задачи машинного обучения быстрее и качественнее, что упростит интеграцию машинного обучения во все сферы деятельности человека. Кроме того, это ускорит разработку более совершенных методов, так как рутинное написание кода можно будет поручить созданной модели.

У существующих методов решения этой задачи есть ряд недостатков. Большие языковые модели, решающие широкий круг задач, связанных с обработкой естественного языка, в том числе генерацию кода, имеют большое число параметров и требуют значительных объёмов данных и вычислительных ресурсов для обучения. Разработка метода генерации кода по текстовому описанию задачи, специализированного именно для задач машинного обучения, поможет решить эти проблемы. Именно такая цель у проекта «Natural Language to Machine Learning» или «NL2ML», в рамках которого выполнялась наша работа. Наш проект уникален тем, что мы учитываем специфику этой задачи, используя семантические классы фрагментов кода машинного обучения как промежуточный этап между текстовым описанием задачи и кодом.

Самый популярный метод обучения нейронных сетей – градиентный спуск – может быть использован только для оптимизации дифференцируемых функций потерь. Но во многих задачах метрики, которые хорошо отражают качество предсказаний модели, не обладают этим свойством, и модели, оптимизируемые только с использованием дифференцируемых функций, не достигают оптимальных результатов. В случаях, когда используемая в задаче модель генерирует последовательность элементов из конечного словаря, эту проблему можно решить с помощью методов из другой области искусственного интеллекта – обучения с подкреплением. Эти методы позволяют обучить такую модель на основе любых численных

метрик качества генерируемых ею последовательностей. В нашей работе мы успешно применяем один из самых популярных на сегодняшний день методов обучения с подкреплением для оптимизации недифференцируемых метрик качества последовательностей семантических классов кода, генерируемых нейронной сетью по текстовому описанию задачи.

## 1.2 Структура проекта

В рамках проекта «NL2ML» задача генерации кода по текстовому описанию задачи машинного обучения делится на два этапа:

- Первый этап состоит в генерации последовательности семантических классов по текстовому описанию задачи. Семантический класс – это одна из 75 категорий, описывающих назначение определённого фрагмента кода, который будет впоследствии сгенерирован. Например, класс `load_from_csv` соответствует загрузке данных, таких как обучающая или тестовая выборка, из файла в формате CSV, а класс `predict_on_test` – запуску некоторой обученной ранее модели машинного обучения для генерации предсказаний на тестовой выборке. Эти семантические классы были введены в статье [5]. Полная классификация семантических классов изображена на рисунке 1.1. Этот этап выполняет нейронная сеть, которую мы будем называть **генератором** последовательностей семантических классов.

- Второй этап состоит в генерации кода на языке Python по этой последовательности семантических классов (возможно, используя также данные об изначальной задаче, такие как используемая метрика или структура используемого набора данных). Этот этап планируется выполнять с помощью нейронной сети, называемой **транслятором**.

Наша работа состоит из двух частей, в которых решаются две разные задачи:

- В главе 3 мы исследуем применение обучения с подкреплением для оптимизации генератора последовательностей семантических классов. Конкретно мы применяем метод PPO (Proximal Policy Optimization) [19].

- В главе 4 мы исследуем способы предсказания целевой метрики соревнования по текстовому описанию этого сорвенования и по сгенерированной последовательности семантических классов.

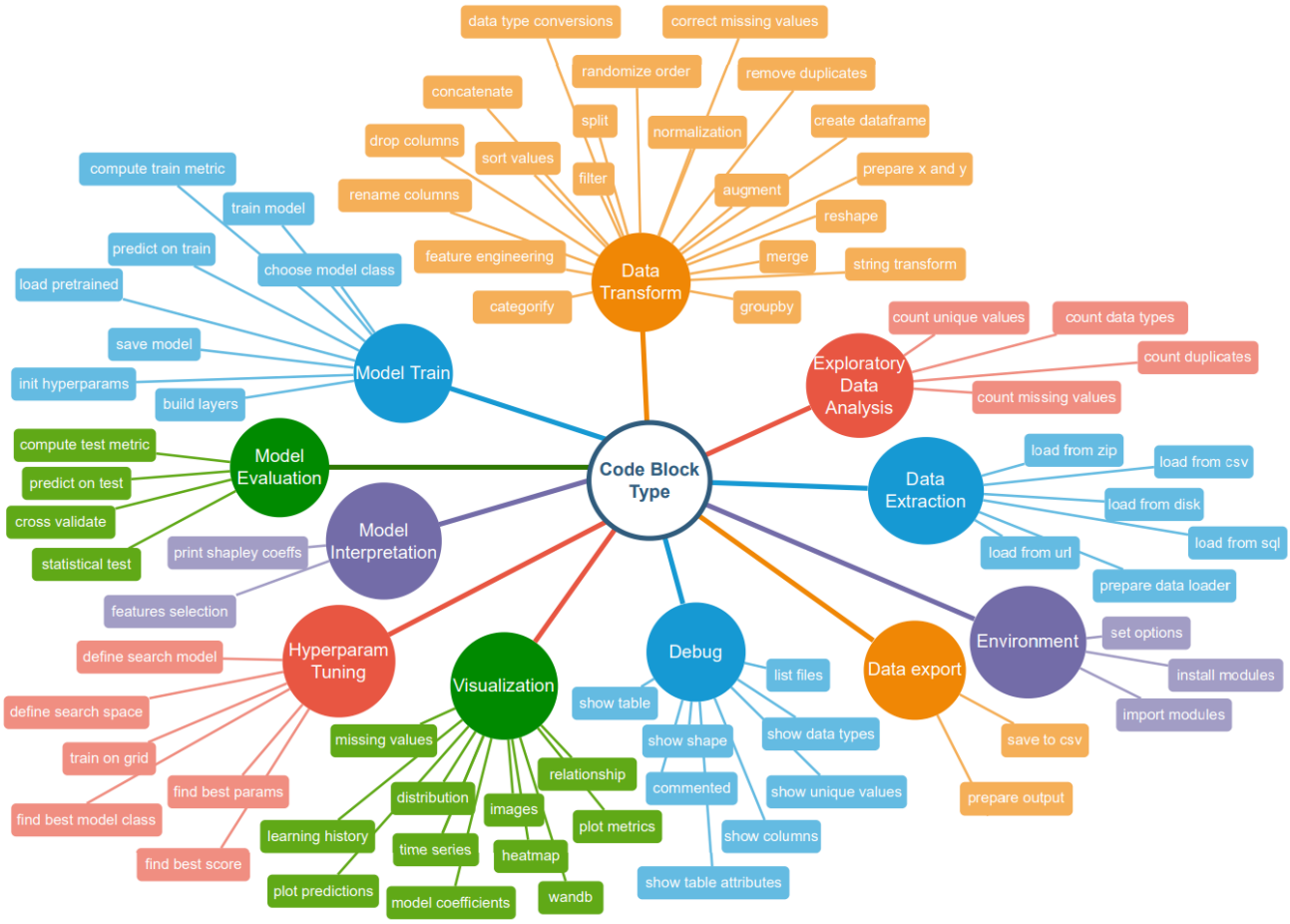


Рис. 1.1: Классификация семантических классов кода. Семантические классы (прямоугольники) сгруппированы в более высокоуровневые категории (круги).

## 1.3 Основные результаты

В данной работе мы установили, что по сравнению с предобучением кросс-энтропии метод PPO позволяет существенно улучшить качество генератора последовательностей семантических классов с точки зрения обеих рассматриваемых нами метрик – BLEU [15] и WER (word error rate). Мы также выяснили, что метрика BLEU лучше подходит в качестве награды для модели, чем метрика WER. Кроме того, мы реализовали модель для предсказания качества сгенерированных последовательностей семантических классов.

Наша работа является важным этапом в проекте «NL2ML», так как полученный нами оптимизированный генератор позволит улучшить качество генерации кода на следующем этапе проекта, а модель для предсказания качества последовательностей семантических классов позволит ускорить этот процесс. Кроме того, полученные нами результаты будут полезны для других проектов, использующих семантические классы кода для задач машинного обучения.

## 1.4 Разделение задач

- Применение обучения с подкреплением для оптимизации генератора последовательностей семантических классов (глава 3) – Николай Людвиг
- Реализация модели для предсказания целевой метрики соревнования (глава 4) – Арслан Разин

# 2 Обзор литературы

## 2.1 Обучение с подкреплением для seq2seq моделей

Seq2seq модель (sequence-to-sequence, «последовательность-в-последовательность») принимает на вход и выдаёт в качестве результата последовательность элементов из некоторого конечного словаря, называемых токенами. Оптимизацию такой модели можно представить в виде задачи обучения с подкреплением. В такой постановке задачи сама seq2seq модель является агентом, последовательность уже сгенерированных токенов – состоянием, а генерация того или иного токена – действием. Seq2seq модели обычно выдают на каждой итерации вероятностное распределение на словаре токенов, то есть каждому токену сопоставляют вероятность того, что он идёт следующим в последовательности. Тогда эти вероятности и есть политика агента, которая оптимизируется в задаче обучения с подкреплением.

Обучение с подкреплением обычно применяется для оптимизации seq2seq моделей в тех случаях, когда метрика задачи недифференцируема, то есть обычным градиентным спуском её оптимизировать невозможно. Так, в статье [16] для оптимизации генерации подписей к изображениям максимизируется недифференцируемая метрика CIDEr [24]. Для этого используется вариант алгоритма REINFORCE [27], в котором в качестве baseline используется последовательность, сгенерированная «жадно» – при генерации каждого следующего токена вместо сэмплирования из выданного моделью распределения детерминированно берётся токен с наибольшей вероятностью. Алгоритм был назван Self-Critical Sequence Training (SCST). Этот алгоритм показал свою эффективность и для других задач обработки естественного языка, например, задачи преобразования текста с целью сделать его легче для понимания [11] и задачи перевода текста в граф с информацией, заключенной в этом тексте, и обратно [4]. Ближе к теме нашей работы статья [3], где Self-Critical Sequence Training используется для исправления ошибок в коде, написанном на языке Java.

Для оптимизации seq2seq моделей широко применяются и другие методы обучения с

подкреплением. В статье [1] метод Actor-Critic, изначально введённый в статье [10], адаптируется для seq2seq задач: основная модель («актор») генерирует последовательности токенов, а другая модель («критик») даёт оценку состояниям «актора». Авторы успешно применяют этот подход для задачи исправления опечаток в тексте и машинного перевода. Этот подход используется и в приобретающем популярность в последнее время алгоритме Proximal Policy Optimization (PPO) [19], который мы исследуем в нашей работе. В статьях [29, 22] PPO применяется для суммаризации текста, причём награды генерируются с помощью модели, обученной на человеческих предпочтениях (reinforcement learning from human feedback). Этот же метод применяется для файн-тюнинга моделей InstructGPT [14], родственных моделям, лежащим в основе известного чат-бота ChatGPT.

Наша работа отличается от перечисленных выше тем, что мы используем обучение с подкреплением для генерации последовательности семантических классов, отражающих назначение фрагментов кода, с целью разработать генератор кода по текстовому описанию задачи. В следующей секции мы рассмотрим существующие подходы к задаче генерации кода по текстовому описанию с использованием обучения с подкреплением.

## 2.2 Обучение с подкреплением для генерации кода по текстовому описанию

Генерация кода по текстовому описанию является seq2seq задачей, так как код можно представить в виде последовательности токенов. Следовательно, для решения этой задачи также применимы методы обучения с подкреплением.

В статье [28] используется policy gradient [23] в сочетании с более традиционным cross entropy loss, чтобы обучить модель переводить запросы, заданные естественным языком, на язык SQL. Но устройство модели, используемой в этой статье, опирается на специфическую структуру запроса SQL и не может быть непосредственно применено для генерации кода на других языках программирования. Авторы статьи [2] используют алгоритм REINFORCE для оптимизации генерации кода на языке программирования для начинающих Karel. Однако синтаксис языка Karel существенно проще языка Python, на котором планируется генерировать код в нашем проекте.

В статьях, где исследуется генерация кода на современных языках программирования, таких как Python, часто берут за основу предобученные модели для генерации кода (например, CodeT5 [26]) и используют обучение с подкреплением для их оптимизации. При этом часто акцент делается на том, чтобы обучить эти модели генерировать компилируемый

код, так как сами они с этой задачей справляются плохо. В статье [25] сгенерированному коду присваивается положительная награда в случае, если он компилируется, и отрицательная иначе. Полученная в результате модель генерирует компилируемый код гораздо чаще, чем предобученные языковые модели, но корректность работы кода никак не проверяется. В некоторых более новых статьях для оптимизации корректности кода используются юнит-тесты. Так, в статье [12] используется подход Actor-Critic, где награда, на которой обучается модель-критик, зависит не только от успешной компиляции кода, но и от прохождения этим кодом тестов. Авторы статьи [21] совершенствуют этот подход, в дополнение к результатам тестов измеряя также семантическую и синтаксическую схожесть с целевым кодом. В обеих статьях [12, 21] для обучения и валидации используется бенчмарк APPS [6], состоящий из текстового условия, тестов (в виде наборов входных и соответствующих выходных данных) и эталонного решения на языке Python задач по программированию с таких сайтов, как Codeforces и Kattis. Хотя в нашем проекте код также будет генерироваться на языке Python, задачи машинного обучения не подходят под такой формат, так как тестов как таковых у них нет, а для оценки качества используются другие метрики.

Ещё одним ключевым отличием нашей работы от перечисленных выше является то, что модель, которую мы оптимизируем, генерирует не непосредственно код, а последовательность семантических классов, описывающих назначение разных фрагментов кода, которые затем будут переводиться в код. Этот подход можно сравнить со статьёй [7], где авторы обучают модель переводить текстовое описание задачи в абстрактное синтаксическое дерево (AST), которое затем конвертируется в код. На этапе обучения с подкреплением одной из компонент функции потерь является награда из метода Self-Critical Sequence Training, упомянутого в предыдущей секции. В нашей работе промежуточный этап между текстовым описанием задачи и кодом является более высокоуровневым, чем абстрактное синтаксическое дерево, так как один семантический класс соответствует одной или нескольким строкам кода.

Все перечисленные в этой секции статьи исследуют задачу генерации кода или в целом, или для решения задач в более конкретной области, не связанной с машинным обучением. Наша работа нацелена именно на генерацию кода, решающего задачи машинного обучения – семантические классы, которые мы используем, были разработаны специально для этих задач. Насколько мы знаем, обучение с подкреплением пока не применялось для генерации кода по текстовому описанию задачи машинного обучения.

# 3 Обучение с подкреплением для оптимизации генерации последовательностей семантических классов кода

В этой части работы нашей целью было улучшение качества генератора, принимающего на вход текстовое описание задачи и выдающего последовательность семантических классов кода, подходящую для решения этой задачи. Генератор описан подробнее в разделе 3.3, а используемые нами метрики качества – в разделе 3.2. Начнём с описания используемых нами данных.

## 3.1 Используемые данные

Основным источником данных для нашей работы был датасет Code4ML [5]. В этом датасете собрана информация о соревнованиях с сайта по исследованию данных Kaggle [8]. В качестве входных данных генератору для каждого соревнования было использовано его название (title), короткое описание (subtitle) и метрика качества, которую требовалось улучшать в этом соревновании (один из 20 классов метрик, выделенных авторами датасета Code4ML). В качестве эталонных (целевых) выходных данных были использованы последовательности семантических классов, полученные из Jupyter-ноутбуков, выложенных в открытый доступ на сайте Kaggle. Чтобы перевести Jupyter-ноутбуки в последовательности семантических классов, авторы датасета извлекли из каждого ноутбука последовательность ячеек, содержащих фрагменты кода, вручную классифицировали часть этих фрагментов по семантическим классам, а оставшиеся фрагменты классифицировали с помощью модели SVM, обученной на этой ручной разметке.

Набор данных, который мы использовали в этой работе, состоял из 4423 целевых последовательностей семантических классов, соответствующих ноутбукам с наилучшими значениями метрики из 242 соревнований. Эти последовательности были случайным образом разделены на обучающую и тестовую выборки в отношении примерно 70:30. Мы также отложили 20% обучающей выборки в качестве валидационной выборки таким образом, чтобы наборы соревнований, встречающихся в валидационной выборке и в оставшейся части обучающей выборки, не пересекались.

В тех случаях, когда в наших целевых последовательностях один и тот же семантический класс встречался несколько раз подряд, повторные вхождения из последовательности удалялись (как в обучающей, так и в тестовой выборке). Это было сделано для того, чтобы не учить модель генерировать несколько одинаковых семантических классов подряд, так как

такие последовательности вряд ли впоследствии будут осмысленны для генерации кода.

## 3.2 Метрики качества

Задачу генерации последовательности семантических классов по текстовому описанию задачи можно рассмотреть как задачу машинного перевода с английского языка на наш предметно-ориентированный язык (domain-specific language, DSL) – язык семантических классов кода, в котором эти классы рассматриваются как слова, а их последовательности – как предложения. Тогда качество последовательностей семантических классов, сгенерированных нашей моделью, определяется тем, насколько они схожи с целевыми последовательностями, полученными разметкой набирающего высокие значения метрик кода с сайта Kaggle. Мы использовали две метрики, широко используемые для оценки качества машинного перевода: BLEU (bilingual evaluation understudy) и WER (word error rate). При описании метрик мы будем называть **гипотезами** (hypotheses) сгенерированные моделью последовательности семантических классов («слов»), качество которых оценивается, а **эталонами** (references) целевые последовательности. При этом одной гипотезе, вообще говоря, может соответствовать несколько эталонов.

### 3.2.1 BLEU

Метрика BLEU была введена в статье [15]. Она измеряет схожесть между набором гипотез $\hat{S} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m)$ и набором эталонов $S = \left((y_1^{(1)}, \ldots, y_1^{(N_1)}), \ldots, (y_m^{(1)}, \ldots, y_m^{(N_m)})\right)$ (каждой из $m$ гипотез $\hat{y}_i$ соответствует $N_i \geqslant 1$ эталонов $(y_i^{(1)}, \ldots, y_i^{(N_i)})$). Чтобы записать формулу для метрики BLEU, определим сначала формулы для нескольких её составных компонент.

Для натурального числа $n$ модифицированная точность $n$-грамм (modified n-gram precision) вычисляется по следующей формуле:

$$p_n(\hat{S}, S) = \frac{\sum\limits_{i=1}^{m} \sum\limits_{s \in G_n} \min\left(C(s, \hat{y}_m), \max\limits_{j=1}^{N_m} C(s, y_i^{(j)})\right)}{\sum\limits_{i=1}^{m} \sum\limits_{s \in G_n} C(s, \hat{y}_m)} \tag{1}$$

Здесь:

- $G_n$ – множество всех возможных различных $n$-грамм (последовательностей из $n$ слов) в используемом словаре,

- $C(s, y)$ – количество вхождений $n$-граммы $s$ как подстроки в строку $y$.

Эффективная длина эталонного корпуса (effective reference corpus length) вычисляется по формуле

$$r(\hat{S}, S) = \sum_{i=1}^{m} |y_{k_i}| \tag{2}$$

Здесь $k_i$ – номер эталона для гипотезы $i$, который наиболее близок к ней по длине (так как метрика BLEU работает на уровне слов, то длина предложения определяется как количество слов в нём). Формально,

$$k_i = \underset{j=1}{\overset{N_i}{\operatorname{argmin}}} \left| |\hat{y}_i| - |y_i^{(j)}| \right| \tag{3}$$

Штраф за краткость (brevity penalty) вычисляется по формуле

$$\mathrm{BP}(\hat{S}, S) = \begin{cases} 1 & \text{при } c(\hat{S}) > r(\hat{S}, S) \\ \exp\left(1 - \frac{r(\hat{S},S)}{c(\hat{S})}\right) & \text{при } c(\hat{S}) \leqslant r(\hat{S}, S) \end{cases} \tag{4}$$

Здесь:

- $c(\hat{S}) = \sum_{i=1}^{m} |\hat{y}_i|$ – сумма длин гипотез,
- $r(\hat{S}, S)$ вычисляется по формуле 2.

Наконец, сама метрика BLEU вычисляется как

$$\mathrm{BLEU}(\hat{S}, S) = \mathrm{BP}(\hat{S}, S) \cdot \sum_{n=1}^{\infty} w_n \ln p_n(\hat{S}, S) \tag{5}$$

Здесь:

- $\mathrm{BP}(\hat{S}, S)$ вычисляется по формуле 4,
- $w_n$ – вес $n$-граммы, определяющий, насколько точность $n$-грамм для некоторого $n$ будет влиять на метрику. Обязаны выполняться условия $0 \leqslant w_n \leqslant 1$, $\sum_{n=1}^{\infty} w_n = 1$. В нашей работе мы используем $w_1 = w_2 = 0.5,\ w_n = 0\ \forall\, n \geqslant 3$.
- $p_n(\hat{S}, S)$ вычисляется по формуле 1.

Значение метрики BLEU находится в диапазоне от 0 до 1 включительно. Чем больше её значение, тем более похож набор гипотез на набор эталонов.

### 3.2.2 WER

Метрика WER измеряет схожесть между гипотезой $\hat{y}$ и одним эталоном $y$. Она вычисляется по формуле

$$\mathrm{WER}(\hat{y}, y) = \frac{S + D + I}{N} \tag{6}$$

Здесь:

- $S$ – минимальное число замен слов, которое нужно произвести, чтобы превратить эталон в гипотезу,
- $D$ – минимальное число удалений слов, которое нужно произвести, чтобы превратить эталон в гипотезу,
- $I$ – минимальное число вставок слов, которое нужно произвести, чтобы превратить эталон в гипотезу,
- $N$ – число слов в эталоне.

Чем меньше значение WER, тем более похожа гипотеза на эталон. Минимально возможное значение метрики 0 соответствует идеальному совпадению гипотезы с эталоном, а сверху значения метрики не ограничены.

Мы адаптировали метрику WER для работы с несколькими эталонами $y^{(1)}, y^{(2)}, \ldots, y^{(N)}$ для одной гипотезы $\hat{y}$ следующим образом:

$$\mathrm{WER}(\hat{y}, y^{(1)}, \ldots, y^{(N)}) = \min_{j=1}^{N} \mathrm{WER}(\hat{y}, y^{(i)}) \tag{7}$$

Для подсчёта метрики по набору гипотез $\hat{S}$ и эталонов $S$ (по аналогии с BLEU) мы использовали взвешенное среднее:

$$\mathrm{WER}(\hat{S}, S) = \sum_{i=1}^{m} \frac{N_i}{\sum_{k=1}^{m} N_k} \cdot \mathrm{WER}(\hat{y}_i, y_i^{(1)}, \ldots, y_i^{(N_i)}) \tag{8}$$

В этой формуле используются обозначения, введённые в начале подраздела 3.2.1.

## 3.3 Оптимизируемая модель

Ранее в рамках проекта «NL2ML» была создана и обучена нейронная сеть-генератор последовательностей семантических классов. В этой части работы нашей задачей было улучшение качества работы этой модели. Её архитектура изображена на рисунке 3.1.

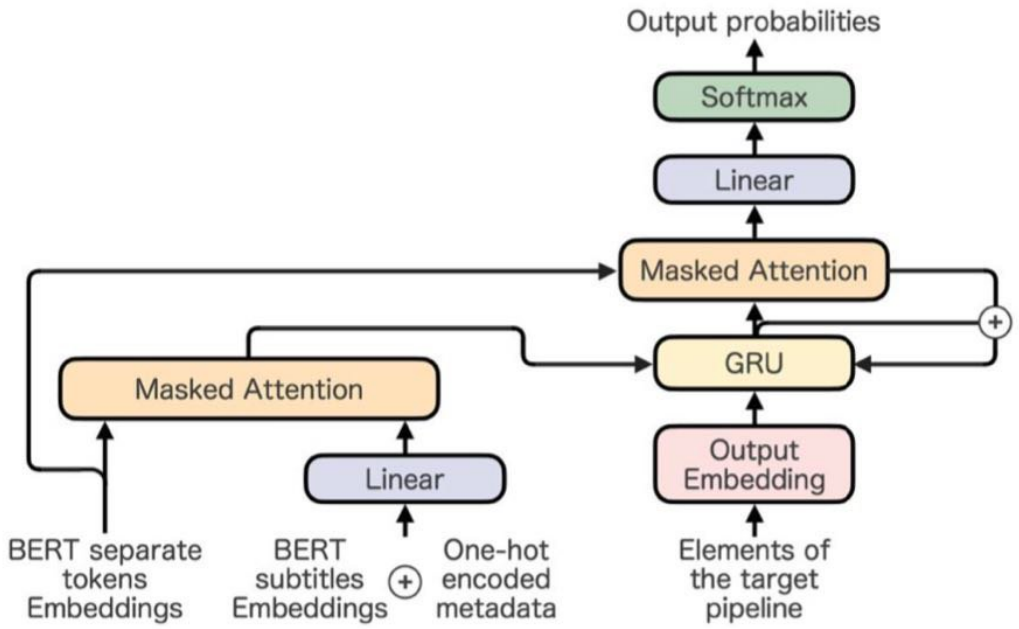


Рис. 3.1: Архитектура модели-генератора последовательностей семантических классов

Модель принимает на вход:

1) последовательность векторов-эмбеддингов длины 768, полученных из слов в названиях и описаниях Kaggle-соревнований (BERT separate tokens embeddings на рисунке),

2) один вектор длины 768, представляющий собой эмбеддинг всего текстового описания соревнования сразу (BERT subtitles embeddings на рисунке),

3) информацию о виде метрики, используемой в соревновании, в виде одного one-hot закодированного вектора длины 20, сконкатенированного с вектором из предыдущего пункта (One-hot encoded metadata на рисунке),

4) последовательность токенов (чисел от 0 до 77, соответствующих 75 семантическим классам и 3 служебным токенам `[SOS]`, `[EOS]` и `[PAD]`), в которой требуется предсказать следующий токен (Elements of the target pipeline на рисунке).

Эмбеддинги для пунктов 1) и 2) были получены с помощью предобученной модели DistilBERT [17] из репозитория HuggingFace. На вход модели для каждого соревнования подавалась строка вида `"[CLS] title [SEP] subtitle [SEP]"`, где `title` – название соревнования, `subtitle` – краткое описание. После этого из последнего скрытого состояния модели (last hidden state) для каждого соревнования для пункта 2) был взят эмбеддинг, соответствующий токену `[CLS]`, а для пункта 1) – эмбеддинги, соответствующие остальным словам в строке.

На выходе модель выдаёт вектор из 78 чисел (Output probabilities на рисунке). Они обозначают вероятности того, что соответствующий токен будет следующим в последовательности.

Эта модель была обучена на данных, описанных в разделе 3.1, с помощью градиентного спуска с использованием кросс-энтропийной функции потерь (cross entropy loss). Однако метрики BLEU и WER, описанные в разделе 3.2, недифференцируемы и поэтому не могут быть оптимизированы непосредственно градиентным спуском. Для оптимизации этих метрик мы применяем методы обучения с подкреплением, которые позволяют обучить модель максимизировать любую численную награду.

## 3.4 Исследуемый метод

Мы применяем метод PPO (Proximal Policy Optimization), введённый в статье [19]. В этом разделе мы изложим теоретическое описание этого метода и того, как мы его применяем для оптимизации модели из предыдущего раздела.

Начнём с определения нескольких понятий из области обучения с подкреплением, основываясь на определениях из статьи [18], на которую также опираются авторы статьи [19].

### 3.4.1 Обучение с подкреплением, V- и Q-функции

В задаче обучения с подкреплением агент взаимодействует со средой, моделируемой как марковский процесс принятия решений [13]. Этот процесс определяется четвёркой $(S, A, P, R)$, где:

- $S$ – множество состояний, в которых может находиться агент,
- $A$ – множество действий, которые может выбирать агент,
- $P(s, a, s') \in [0, 1]$ – вероятность, что сделав из состояния $s \in S$ действие $a \in A$, агент попадёт после этого в состояние $s' \in S$,
- $R(s, a, s') \in \mathbb{R}$ – награда, получаемая агентом за попадание из состояния $s \in S$ в состояние $s' \in S$ благодаря действию $a \in A$.

Агент начинает из некоторого состояния $s_0 \in S$, которое тоже может выбираться случайно, и на каждом шаге ходит из состояния в состояние, выбирая действия. Цель задачи

обучения с подкреплением – подобрать оптимальную политику агента $\pi$, то есть способ выбора некоторого действия из $A$ для всех возможных состояний в $S$, для максимизации наград. Вообще говоря, политика $\pi$ может быть как детерминированной (действие агента всегда однозначно определяется состоянием), так и стохастической (в выборе агентом действий есть элемент случайности). В этом разделе мы будем рассматривать случай, когда политика агента стохастическая, т. е. такая, где из каждого состояния $s \in S$ действие $a \in A$ выбирается случайным образом в соответствии с случайным распределением на $A$, задаваемым политикой $\pi$ и зависящим только от состояния $s$: $a \sim \pi(a \mid s)$.

Пусть агент в задаче обучения с подкреплением начал из некого изначального состояния $s_0$ и в каждый момент времени $t \in \{0, 1, 2, \ldots\}$ случайным образом выбирал согласно политике $\pi$ действие $a_t \sim \pi(a_t \mid s_t)$, получая за это награду $r_t$ и попадая в результате этого в новое состояние $s_{t+1}$. Таким образом образуется траектория $(s_0, a_0, s_1, a_1, s_2, \ldots)$. Определим (дисконтированную) V-функцию (state value function) от состояния $s_t$ как

$$V(s_t) = \mathbb{E}_{(a_t, s_{t+1}, a_{t+1}, \ldots)} \left[ \sum_{l=0}^{\infty} \gamma^l r_{t+l} \right] \tag{9}$$

Здесь $\gamma \in [0, 1]$ – гиперпараметр, используемый для дисконтирования будущих наград (он позволяет дать краткосрочным последствиям действий агента больший вес, чем долгосрочным). Похожим образом определяется (дисконтированная) Q-функция (state-action value function) от состояния $s_t$ и действия $a_t$:

$$Q(s_t, a_t) = \mathbb{E}_{(s_{t+1}, a_{t+1}, \ldots)} \left[ \sum_{l=0}^{\infty} \gamma^l r_{t+l} \right] \tag{10}$$

Другими словами, V-функция равна ожидаемой суммарной награде, которую можно получить из состояния $s_t$ (с учётом дисконтирования), а Q-функция равна ожидаемой награде, которую можно получить из состояния $s_t$, сделав действие $a_t$. Функция выгоды (advantage function) определяется через эти две функции:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{11}$$

Также нам будет полезно определить для момента времени $t$ TD-остаток (TD residual), равный

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{12}$$

### 3.4.2 Policy gradient и актор-критик

Policy gradient методы, к которым относится PPO, стремятся непосредственно оптимизировать политику агента $\pi$ так, чтобы максимизировать ожидаемую суммарную награду, полученную за всю траекторию:

$$\mathbb{E}_{(s_0,a_0,s_1,\ldots)}\left[\sum_{t=0}^{\infty} r_t\right] \to \max_{\pi} \tag{13}$$

Если политика нашего агента зависит от некоторого вектора параметров $\theta$ (например, если она генерируется нейронной сетью с весами $\theta$), то нам хотелось бы делать градиентный спуск функции 13 по $\theta$. Однако эта функция, вообще говоря, недифференцируема. Поэтому в policy gradient методах вместо настоящего градиента этой функции используется его аппроксимация:

$$g = \mathbb{E}_{(s_0,a_0,s_1,\ldots)}\left[\sum_{t=0}^{\infty} A(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] \tag{14}$$

Здесь $\pi_\theta(a_t \mid s_t)$ – вероятность сделать действие $a_t$ в состоянии $s_t$, используя политику агента $\pi$ (зависящую от $\theta$). Градиент по логарифмам вероятностей действий посчитать уже можно, но в формуле 14 всё равно остаются матожидания, которые на практике заменяются на оценки. Авторы статьи [18] вводят обобщённую оценку выгоды (generalized advantage estimator, GAE), определяемую как

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{15}$$

Здесь $\gamma \in [0, 1]$ – тот же гиперпараметр, что и в формулах 9, 10 и 12, а $\lambda \in [0, 1]$ – другой гиперпараметр, который регулирует компромисс между смещением и разбросом (bias-variance tradeoff). Подставив эту оценку вместо настоящей функции выгоды в функцию 14 и заменив внешнее матожидание на среднее по $N$ независимо сэмплированным траекториям, получаем следующую оценку градиента:

$$\hat{g} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty} \hat{A}_{n,t}\nabla_\theta \log \pi_\theta(a_{n,t} \mid s_{n,t}) \tag{16}$$

Здесь индекс $n$ в обозначениях $a_{n,t}$, $s_{n,t}$ и $\hat{A}_{n,t}$ обозначает действия, состояния и оценки выгоды для траектории $n$ соответственно. При работе с фреймворками глубинного обучения, которые автоматически считают градиент методом обратного распространения ошибки (backpropagation), удобно задать функцию, градиентом которой по $\theta$ является оценка 16, и

максимизировать её:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\hat{A}_{n,t}\log\pi_\theta(a_{n,t}\mid s_{n,t})\to\max_\theta \tag{17}$$

Чтобы выполнять градиентный подъём по этой функции, осталось взять откуда-то значения $V(s_t)$, которые возникают после раскрытия $\hat{A}_{n,t}$. Чтобы получить оценку этих значений, часто используется подход «актор-критик» – одновременно с оптимизацией политики агента $\pi$ модель обучается оценивать V-функцию от текущего состояния. Эти оценки используются в вычислении TD-остатков по формуле 12, которые затем подставляются в оценки 15, 16 и 17. Такой подход используется и в методе PPO.

### 3.4.3 Proximal Policy Optimization

Предшественником PPO является метод TRPO (Trust Region Policy Optimization) [20]. В этом методе используется следующая модификация функции 17:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\hat{A}_{n,t}\frac{\pi_\theta(a_{n,t}\mid s_{n,t})}{\pi_{\theta_{\text{old}}}(a_{n,t}\mid s_{n,t})}\to\max_\theta \tag{18}$$

Эта функция максимизируется при каждом обновлении весов с старых их значений $\theta_{\text{old}}$ на новые $\theta$. В методе PPO (Proximal Policy Optimization) [19] используется улучшенная версия этой функции, предотвращающая слишком сильные изменения политики агента.

Рассмотрим одну конечную траекторию агента $(s_0, a_0, s_1, \ldots, a_{T-1}, s_T)$. Будем использовать для этой траектории обрезанную версию обобщённой оценки выгоды из формулы 15:

$$\hat{A}_t=\sum_{l=0}^{T-t-1}(\gamma\lambda)^l\delta_{t+l} \tag{19}$$

Обозначим отношение между новой и старой вероятностями для момента времени $t$

$$r_t(\theta)=\frac{\pi_\theta(a_t\mid s_t)}{\pi_{\theta_{\text{old}}}(a_t\mid s_t)} \tag{20}$$

Тогда основная компонента функции, максимизируемой по $\theta$ в методе PPO, для момента времени $t$ записывается как

$$L_t^{CLIP}(\theta)=\min(r_t(\theta)\cdot\hat{A}_t,\ \text{clip}(r_t(\theta),1-\varepsilon,1+\varepsilon)\cdot\hat{A}_t) \tag{21}$$

Здесь:

- функция $\text{clip}(x,l,r)=\max(l,\min(x,r))$ «зажимает» значение $x$ между $l$ и $r$,

- гиперпараметр $\varepsilon \in (0, 1)$ регулирует то, насколько сильным может быть одно изменение политики агента.

Помимо $L^{CLIP}$, функция, которая максимизируется в методе PPO, содержит ещё две компоненты. Первая из них – функция ошибки для «критика», то есть для модели, оценивающей V-функцию от текущего состояния. Это простая квадратичная ошибка:

$$L_t^{VF} = (V_\theta(s_t) - V_t^{\text{target}})^2 \tag{22}$$

Здесь:

- $V_\theta(s_t)$ – оценка моделью с весами $\theta$ V-функции в состоянии $s_t$,
- $V_t^{\text{target}} = \hat{A}_t + V_{\theta_{\text{old}}}(s_t)$.

Также с целью мотивировать агента пробовать разные действия (exploration) к максимизируемой функции может добавляться энтропия политики агента в состоянии $s_t$:

$$S[\pi_\theta](s_t) = -\sum_{a \in A} \pi_\theta(a \mid s_t) \ln \pi_\theta(a \mid s_t) \tag{23}$$

Здесь $A$ – множество возможных действий агента. Собирая выражения из формул 21, 22 и 23 вместе и рассматривая $N$ конечных траекторий, в $n$-й из которых $T_n$ шагов, получаем следующую функцию, максимизируемую с помощью градиентного подъёма:

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n - 1} [L_{n,t}^{CLIP}(\theta) - c_1 L_{n,t}^{VF}(\theta) + c_2 S[\pi_\theta](s_{n,t})] \to \max_\theta \tag{24}$$

Здесь неотрицательные коэффициенты $c_1, c_2$ являются гиперпараметрами.

### 3.4.4 Применение PPO в нашей задаче

Как было описано ранее, задачу оптимизации sequence-to-sequence модели можно представить в виде задачи обучения с подкреплением. Формализуем это. Пусть модель принимает на вход некоторые входные данные $x$ из множества всех возможных входных данных $\mathbb{X}$ и на каждой итерации выдаёт токен $y$ из словаря $D$. Тогда в терминологии обучения с подкреплением:

- множество действий $A = D$ (т. е. каждое действие соответствует токену, который может сгенерировать модель),

- множество состояний $S = \{(x, y_0, y_1, \ldots, y_{T-1}) \mid x \in \mathbb{X},\ y_i \in D,\ T \geqslant 0\}$ (т. е. состояние задаётся входными данными для модели и последовательностью токенов, которые она уже сгенерировала),
- изначальное состояние $s_0 = (x)$, $x \in \mathbb{X}$ (т. е. модель уже получила входные данные, но пока ещё ничего не сгенерировала),
- вектор вероятностей $\pi_i \in \mathbb{R}^{|D|}$, из которого сэмплируется токен $y_i$, является политикой агента: $\pi_i = \pi(y_i \mid s_i)$,
- при выборе действия $y_i$ из состояния $s_i = (x, y_0, \ldots, y_{i-1})$ агент всегда переходит в однозначно определённое состояние $s_{i+1} = (x, y_0, \ldots, y_{i-1}, y_i)$,
- агент получает ненулевую награду только за последнее действие эпизода (т. е. только в случае генерации служебного токена `[EOS]` или достижения максимальной длины последовательности, которая в нашей работе была равна 100),
- эта награда является некоторой функцией от сгенерированной последовательности и неким образом определяет её качество, которое мы хотим обучить агента максимизировать.

В нашей работе мы применяем метод PPO, описанный в подразделе 3.4.3, для оптимизации seq2seq модели, описанной в разделе 3.3. Мы обучаем эту же модель оценивать V-функцию путём добавления в модель дополнительного линейного слоя, принимающего на вход скрытое состояние декодера и выдающего одно число – оценку V-функции для состояния, в котором сейчас находится модель (в терминологии обучения с подкреплением). Награда $R$, которую мы присваиваем сгенерированной последовательности $\hat{y}$, в нашей работе являлась функцией от этой последовательности и набора эталонов $(y^{(1)}, \ldots, y^{(N)})$ из обучающей выборки для соревнования, информация о котором была подана модели в качестве входных данных. В разделе 3.5 описаны наши эксперименты с разными функциями награды $R(\hat{y}, y^{(1)}, \ldots, y^{(N)})$.

Следуя примеру статьи [29], мы также добавляем к награде слагаемое, пропорциональное минус дивергенции Кульбака-Лейблера между политиками оптимизируемой модели и изначальной модели. Это делается для того, чтобы при оптимизации политика не слишком сильно отходила от политики предобученной модели. Итоговая награда тогда равна

$$r(x, \hat{y}, y^{(1)}, \ldots, y^{(N)}) = R(\hat{y}, y^{(1)}, \ldots, y^{(N)}) - \beta \ln \frac{\pi(\hat{y} \mid x)}{\rho(\hat{y} \mid x)} \tag{25}$$

Здесь:

- $\pi(\hat{y} \mid x)$ – вероятность того, что оптимизируемая модель сгенерирует последовательность $\hat{y}$ на входных данных $x$,
- $\rho(\hat{y} \mid x)$ вероятность того, что изначальная модель (до оптимизации с помощью PPO) сгенерировала бы последовательность $\hat{y}$ на входных данных $x$.

Коэффициент $\beta$ здесь варьируется на каждой итерации оптимизации. На $(k+1)$-й итерации этот коэффициент $\beta_{k+1}$ вычисляется как

$$\beta_{k+1} = \beta_k \left(1 + \frac{k}{h} \cdot \text{clip}\left(\frac{\text{KL}(\pi, \rho)}{\text{KL}_{\text{target}}} - 1, -0.2, 0.2\right)\right) \tag{26}$$

Здесь:

- $\text{KL}(\pi, \rho)$ – дивергенция Кульбака-Лейблера между политиками $\pi$ (оптимизируемой модели) и $\rho$ (изначальной модели),
- $\beta_0$, $h$, $\text{KL}_{\text{target}}$ – гиперпараметры.

## 3.5 Эксперименты

### 3.5.1 Награда как комбинация метрик

Так как нашей целью была оптимизация метрик BLEU и WER, то мы использовали в качестве награды линейную комбинацию этих двух метрик (взяв WER со знаком «минус», так как его мы хотим минимизировать):

$$R(\hat{y}, y^{(1)}, \ldots, y^{(N)}) = \alpha \cdot \text{BLEU}((\hat{y}), (y^{(1)}, \ldots, y^{(N)})) - (1 - \alpha) \cdot \text{WER}(\hat{y}, y^{(1)}, \ldots, y^{(N)}) \tag{27}$$

Здесь BLEU и WER вычисляются по формулам 5 и 7 соответственно. Значения коэффициента $\alpha$ мы перебирали от 0 до 1 включительно с шагом 0.2. Полученные значения метрик BLEU и WER на валидационной и тестовой выборках представлены в таблицах 3.1 и 3.2. Для каждого значения $\alpha$ мы делали 3 запуска и усредняли результаты в столбце mean. Для сравнения в последней строке указаны значения метрик для предобученной модели до применения PPO.

Таблица 3.1: Полученные значения BLEU в зависимости от $\alpha$

| | Validation BLEU | | | | Test BLEU | | | |
|---|---|---|---|---|---|---|---|---|
| $\alpha$ | 1 | 2 | 3 | mean | 1 | 2 | 3 | mean |
| 0.0 | 0.13 | 0.33 | 0.22 | 0.22 | 0.12 | 0.24 | 0.21 | 0.19 |
| 0.2 | 0.23 | 0.23 | 0.13 | 0.20 | 0.22 | 0.23 | 0.16 | 0.20 |
| 0.4 | 0.15 | 0.18 | 0.18 | 0.17 | 0.16 | 0.18 | 0.18 | 0.17 |
| 0.6 | 0.22 | 0.29 | 0.19 | 0.24 | 0.23 | 0.30 | 0.21 | 0.25 |
| 0.8 | 0.19 | 0.25 | 0.24 | 0.23 | 0.21 | 0.23 | 0.23 | 0.23 |
| 1.0 | 0.34 | 0.26 | 0.38 | **0.32** | 0.30 | 0.26 | 0.32 | **0.29** |
| baseline | | | | 0.21 | | | | 0.17 |

Таблица 3.2: Полученные значения WER в зависимости от $\alpha$

| | Validation WER | | | | Test WER | | | |
|---|---|---|---|---|---|---|---|---|
| $\alpha$ | 1 | 2 | 3 | mean | 1 | 2 | 3 | mean |
| 0.0 | 0.38 | 0.36 | 0.39 | 0.38 | 0.46 | 0.71 | 0.49 | 0.56 |
| 0.2 | 0.34 | 0.31 | 0.39 | **0.35** | 0.44 | 0.42 | 0.48 | **0.45** |
| 0.4 | 0.35 | 0.40 | 0.35 | 0.37 | 0.46 | 0.47 | 0.44 | 0.46 |
| 0.6 | 0.38 | 0.32 | 0.40 | 0.37 | 0.42 | 0.43 | 0.51 | **0.45** |
| 0.8 | 0.39 | 0.37 | 0.40 | 0.38 | 0.43 | 0.52 | 0.43 | 0.46 |
| 1.0 | 0.39 | 0.37 | 0.44 | 0.40 | 0.49 | 0.44 | 0.50 | 0.47 |
| baseline | | | | 1.01 | | | | 1.21 |

### 3.5.2 Детали экспериментов

Обучение методом PPO для каждого запуска длилось 10 эпох, после чего значения метрик переставали улучшаться. Мы использовали размер батча, равный 256, и размер минибатча, равный 128. Это означает, что на каждой итерации мы брали 256 объектов (целевых последовательностей семантических классов) из обучающей выборки и для каждого из этих объектов подавали на вход модели текстовое описание соответствующего соревнования и генерировали последовательность семантических классов. Затем каждой из этих последовательностей присваивалась награда в соответствии с процедурой, описанной в начале этого раздела, и на всех этих 256 последовательностях выполнялся шаг PPO. В свою очередь, шаг PPO состоял из 4 «внутренних» эпох, в каждой из которых производилось 2 итерации подсчёта функции 24 и обновления весов модели, каждая по 128 объектам. Размер батча был выбран максимальным, который нам позволяло оборудование, а размер минибатча был выбран таким, так как на больших размерах обучение проходило слишком медленно, а на меньших размерах становилось слишком нестабильным. Значения остальных гиперпараметров PPO были следующие: $\gamma = 1$, $\lambda = 0.95$, $c_1 = 0.1$, $c_2 = 0$, $\varepsilon = 0.2$, $\beta_0 = 0.2$, $h = 10^4$, $\mathrm{KL}_{\mathrm{target}} = 6$.

В отличие от вычисления метрик-компонент награды агента, при вычислении метрик

BLEU и WER на валидационной и тестовой выборках последовательности генерировались «жадно», т. е. на каждой итерации выбирался токен с максимальной вероятностью. Метрики вычислялись по формулам 5 и 8, в которых в качестве набора гипотез $\hat{S}$ выступала одна последовательность на каждое соревнование из соответствующей выборки (так как она генерировалась детерминированно, то генерировать больше одной не имело смысла), а в качестве набора эталонов $S$ – набор целевых последовательностей семантических классов из соответствующей выборки, сгруппированных по соревнованию.

Как и в статье [19], мы использовали оптимизатор Adam [9]. Темп обучения (learning rate) был равен $10^{-4}$ в течение первых двух эпох и $2 \cdot 10^{-5}$ после этого. Значения остальных гиперпараметров Adam были следующие: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$.

### 3.5.3 Выводы

Метод PPO позволяет улучшить качество последовательностей семантических классов кода, генерируемых моделью по текстовому описанию задачи, с точки зрения обеих рассматриваемых нами метрик – BLEU и WER. Особенно сильный эффект можно наблюдать на метрике WER, которую мы смогли уменьшить почти в 3 раза по сравнению с моделью, обученную традиционным градиентным спуском на основе кросс-энтропийной функции потерь.

В ходе экспериментов с различными линейными комбинациями BLEU и WER в качестве награды мы выяснили, что при большом вкладе BLEU в награду метрика WER улучшается практически так же хорошо, как и при большем вкладе WER, однако обратное неверно – при большом вкладе WER метрика BLEU улучшается существенно хуже. Также, изучив последовательности семантических классов, сгенерированные моделями при разных значениях $\alpha$, мы пришли к выводу, что модели с большим вкладом BLEU генерируют более длинные и разнообразные последовательности семантических классов, которые смогут быть переведены в более качественный код. Возможно, это происходит потому, что метрика WER не содержит дополнительного штрафа за слишком короткие предложения, который присутствует в метрике BLEU (см. формулу 4). На основании этих результатов мы считаем, что для этой задачи метрика BLEU подходит в качестве награды лучше, чем WER.

Главу 3 выполнил Николай Людвиг.

# 4 Оценка качества сгенерированной последовательности для оптимизации генерации

В этой главе нашей задачей была оптимизация генератора, используя не только текстовое описание задачи, но и сгенерированную им последовательность. Сам генератор был подробно описан нами в разделе 3.3. Важность такой оценки описана в разделе 4.1. Данные, использованные в этой части работы, описаны в разделе 4.2.

## 4.1 Мотивация

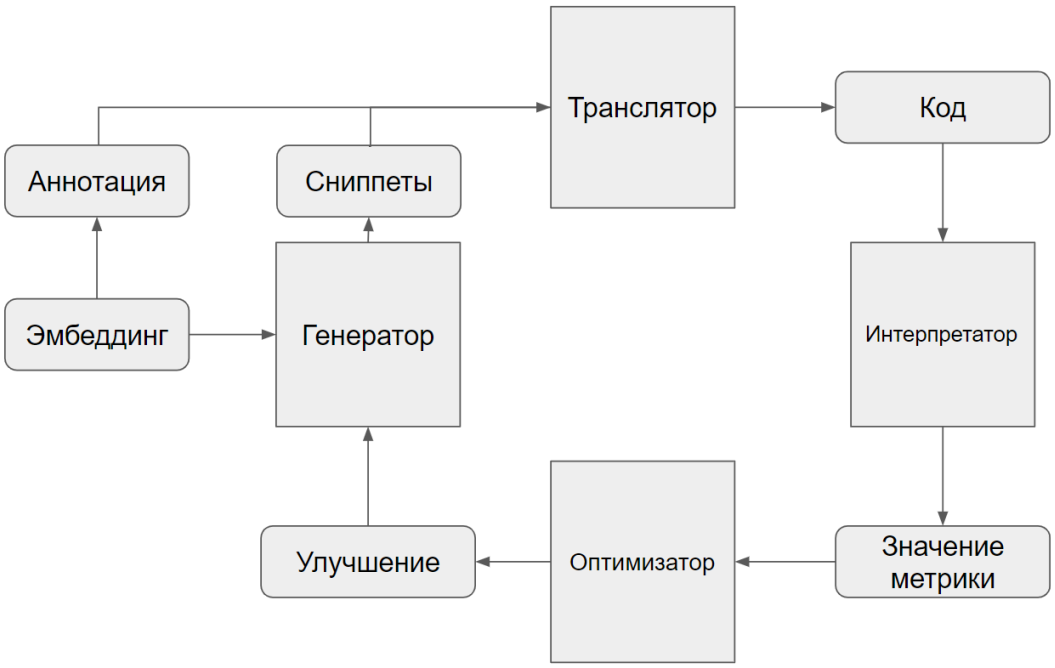


Рис. 4.1: Архитектура проекта «NL2ML»

Как видно на рисунке 4.1 после генерации последовательности семантических классов (на схеме они называются "Сниппеты") следует работа транслятора и интерпретатора. Итогом их работы является некоторое значение метирки, по которому наш оптимизатор должен улучшить генератор. Однако такой метод оптимизации сильно зависит от транслятора и кода, который он сгенерирует, так как для получения итогового результата требуется запустить код. Чтобы ускорить оптимизацию мы оценивать качество сгенерированной последовательности простой моделью. Тогда процесс оптимизации будет таким, как на рисунке 4.2. Так же такой подход позволяет не использовать транслятор для оптимизации генератора последовательностей.

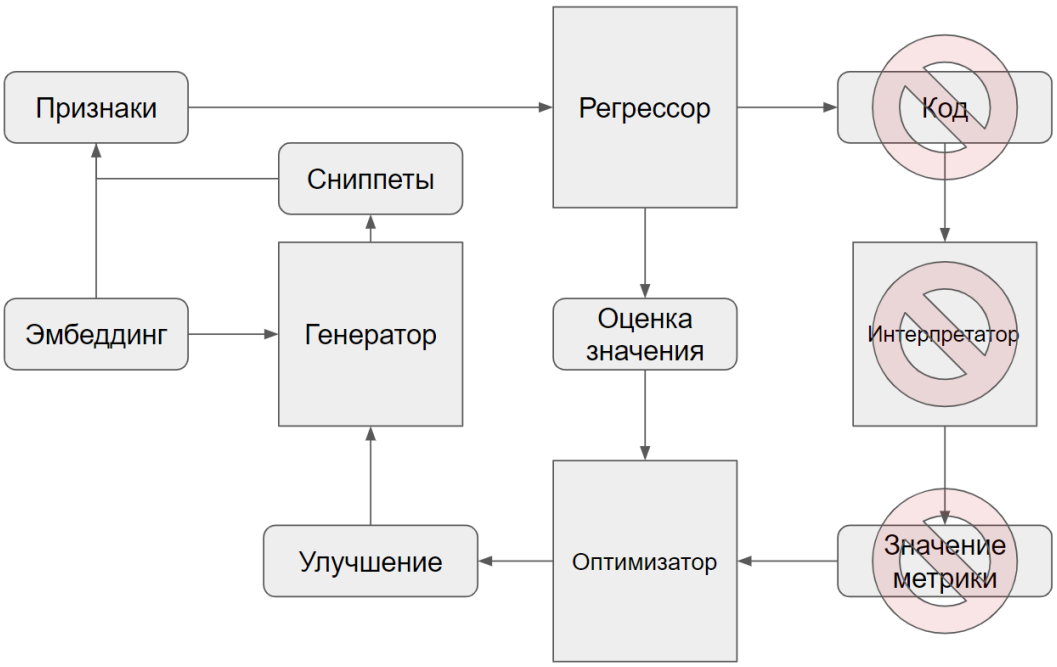


Рис. 4.2: Способ ускорения оптимизации, использованный в данной главе

## 4.2 Используемые данные

Данные для этой части работы были взяты из датасета Code4ML [5]. Но в отличие от оптимизации генератора на основе языковых моделей были взяты не только лучшие ноутбуки из каждого соревнования, а вообще все размеченные (в том числе автоматически размеченные последовательности).

Для каждого ноутбука взятого с Kaggle [8] были использованы следующие данные для обучения: последовательсноть семантических типов (вручную выделенных авторами датасета) и вектор эмбеддинг, описывающий сорвенование. А целевой переменной для данной части работы является значение метрики (одной из 20 типов), которое получил данный ноутбук. Из последовательностей так же были убраны идущие подряд одинаковые семантические классы, чтобы корректно оценивать сгенерированные данные.

Всего для обучения используется 22249 размеченных ноутбуков. Отношение обучающих и тестовых данных 75:25.

## 4.3 Метрика качества

Задача предсказания некоторого числа по данным – задача регрессии. Но так как для каждого соревнования метрика определяется по отдельности, а всего метрик в размеченных данных 20 штук, нужно использовать универсальную метрику качества, показывающую насколько корректно предсказание. Такой метрикой является метрика $R^2$ (коэффициент де-

терминации).

Метрика $R^2$ используется для оценки качества модели регрессии. Её значение находится в диапазоне от 0 до 1, где 1 означает идеальную соответствие модели данным, а 0 - полное отсутствие соответствия.

Данная метрика может быть вычислена по следующей формуле:

$$R^2 = 1 - \frac{\sigma^2}{\sigma_y^2} \tag{28}$$

Здесь:

- $\sigma^2$ - сумма квадратов остатков (разница между фактическими значениями и предсказанными)
- $\sigma_y^2$ - общая сумма квадратов (разница между фактическими значениями и их средним значением)

Плюсы метрики $R^2$:

1) Позволяет оценить качество модели регрессии, сравнивая её со случайной моделью.
2) Легко интерпретируется, так как значение находится в диапазоне от 0 до 1.
3) Может быть использована для сравнения разных моделей регрессии.

Минусы метрики $R^2$:

1) Не учитывает количество наблюдений и количество переменных в модели, что может привести к неверным выводам при сравнении моделей с разным количеством переменных.
2) Не является абсолютной метрикой, так как её значение зависит от выборки данных.

Стоит заметить, что все используемые в дальнейшем модели будут обучаться на одних и тех же данных, а значит минусы данной метрики не существенны.

## 4.4 Преобразование данных

Прежде, чем сделать предсказание результата следует правильно обработать имеющиеся данные, чтобы любые модели регрессии могли обучаться по ним (это нужно, чтобы наилучшую модель для предсказания результата).

### 4.4.1 Преобразование признаков

В данной работе для нормализации признаков был использован StandartScaler. Принцип его работы заключается в нормализации признаков путем приведения их к стандартному нормальному распределению со средним значением равным 0 и стандартным отклонением равным 1. Для этого, StandartScaler вычисляет среднее значение и стандартное отклонение каждого признака на основе обучающей выборки данных. Затем, она применяет эти значения к каждому признаку в обучающей и тестовой выборках данных, чтобы нормализовать их значения. Это позволяет улучшить качество модели машинного обучения, особенно в случаях, когда значения признаков имеют разный масштаб или распределение.

Однако перед нормировкой данных следует преобразовать последовательность семантических классов, так как он имеет переменную длину. Для этого мы закодировали данную последовательность в виде one-hot векторов, после чего сложили все такие вектора, домножая вектор с номером $i$ на число $0.5^i$.

В итоге набор признаков для каждого ноутбука состоит из эмбеддинга (длины 788), описывающего текст соревнования и его метаданные, из преобразованной последовательности семантических классов (теперь уже фиксированной длины 78) и ещё одной переменной - длины последовательности семантических классов.

### 4.4.2 Преобразование целевой переменной

Так целевая переменная - значение неизвестной метрики, то следует так преобразовать данные, чтобы уменьшить разброс, но при этом иметь возможность однозначно восстановить значение. Для этого можно использовать сигмоиду.

Сигмоида - это математическая функция, которая принимает любое значение и преобразует его в диапазон от 0 до 1. Она широко используется в машинном обучении для задач классификации, так как она позволяет получить вероятностную оценку принадлежности объекта к определенному классу.

Для этой задачи мы использовали рациональую сигмоиду: $\sigma(x) = \frac{x}{|x|+1}$, так как она более устойчива к выбросам.

## 4.5 Используемые модели

Для предсказания значения метрики мы использовали пять различных моделей из библиотеки sklearn, а именно: линейные регрессоры - Ridge и ElasticNet, регрессор на основе опорных векторов - LinearSVR, регрессоры на основе ансамблей решающих деревьев -

GradientBoostingRegressor и RandomForestRegressor. Далее идёт краткое описание этих методов, а так же их преимущества и недостатки по сравнию друг с другом.

### 4.5.1 Ridge

Ridge-регрессия из библиотеки sklearn является методом линейной регрессии с добавлением L2-регуляризации. Она минимизирует функцию потерь, которая включает в себя сумму квадратов разностей между предсказанными и фактическими значениями, а также штраф на коэффициенты модели, чтобы уменьшить их величину.

Плюсы Ridge-регрессии:

1) Снижение переобучения модели и улучшение ее обобщающей способности

2) Позволяет использовать все признаки в модели, а не отбрасывать незначимые

3) Легко интерпретируема и проста в использовании

Минусы Ridge-регрессии:

1) Не подходит для отбора признаков, так как не зануляет коэффициенты

2) Не всегда может дать лучший результат, чем другие регрессоры, если в данных много незначимых признаков или они сильно коррелируют между собой

### 4.5.2 ElasticNet

ElasticNet из библиотеки sklearn является методом линейной регрессии с добавлением комбинации L1 и L2-регуляризации. Он минимизирует функцию потерь, которая включает в себя сумму квадратов разностей между предсказанными и фактическими значениями, а также штраф на коэффициенты модели, чтобы уменьшить их величину и занулить некоторые из них.

Плюсы ElasticNet:

1) Позволяет отбирать признаки, зануляя некоторые из коэффициентов

2) Снижение переобучения модели и улучшение ее обобщающей способности

3) Легко интерпретируема и проста в использовании

Минусы ElasticNet:

1) Не всегда может дать лучший результат, чем другие регрессоры, если в данных много незначимых признаков или они сильно коррелируют между собой

2) Может быть более вычислительно сложным, чем Ridge-регрессия или Lasso-регрессия

По сравнению с другими регрессорами, ElasticNet обладает преимуществами обеих регрессий (Ridge и Lasso) и может быть более эффективным в случаях, когда данные имеют большое количество признаков, некоторые из которых являются незначимыми или коррелируют между собой. Однако, если данные имеют меньшее количество признаков, то Ridge-регрессия или Lasso-регрессия могут быть более эффективными.

### 4.5.3 LinearSVR

LinearSVR (Linear Support Vector Regression) - это метод регрессии, основанный на использовании опорных векторов. Он является одним из регрессоров, доступных в библиотеке машинного обучения sklearn.

Плюсы LinearSVR:

1) Хорошо работает с большими наборами данных и признаками высокой размерности

2) Позволяет работать с нелинейными зависимостями через использование ядер

3) Позволяет контролировать уровень регуляризации через параметр C

Минусы LinearSVR:

1) Не обрабатывает выбросы в данных

2) Может быть чувствителен к шуму в данных

3) Не всегда даёт наилучшее качество предсказания в сравнении с другими регрессорами, такими как RandomForestRegressor или GradientBoostingRegressor

В целом, выбор конкретного регрессора зависит от конкретной задачи и её особенностей. Например, если данные содержат выбросы, то может быть лучше использовать более устойчивые к ним регрессоры, такие как HuberRegressor. Если же данные имеют нелинейную зависимость, то может быть лучше использовать другие методы, такие как полиномиальная регрессия или нейронные сети. Важно также учитывать время обучения и предсказания модели, а также её интерпретируемость.

### 4.5.4 GradientBoostingRegressor

GradientBoostingRegressor из библиотеки sklearn является методом градиентного бустинга для решения задачи регрессии. Он строит ансамбль из деревьев решений, где каждое последующее дерево исправляет ошибки предыдущего дерева, пока не достигнет минимального значения функции потерь.

Плюсы GradientBoostingRegressor:

1) Обладает высокой точностью предсказаний

2) Может работать с различными типами признаков (категориальными, числовыми и т.д.)

3) Снижение переобучения модели благодаря встроенной регуляризации

Минусы GradientBoostingRegressor:

1) Требует большого количества времени для обучения и настройки гиперпараметров

2) Может быть склонен к переобучению, если количество деревьев слишком большое или глубина деревьев слишком велика

По сравнению с другими регрессорами, GradientBoostingRegressor обладает высокой точностью предсказаний и может работать с различными типами признаков. Однако, он требует большого количества времени для обучения и настройки гиперпараметров, что может быть проблемой в случае большого объема данных. Кроме того, он может быть склонен к переобучению, если количество деревьев слишком большое или глубина деревьев слишком велика. В таких случаях, Ridge-регрессия или Lasso-регрессия могут быть более эффективными.

### 4.5.5 RandomForestRegressor

RandomForestRegressor из библиотеки sklearn является методом случайного леса для решения задачи регрессии. Он строит ансамбль из деревьев решений, где каждое дерево обучается на случайной подвыборке данных и случайном подмножестве признаков.

Плюсы RandomForestRegressor:

1) Обладает высокой точностью предсказаний

2) Может работать с различными типами признаков (категориальными, числовыми и т.д.)

3) Снижение переобучения модели благодаря случайности в выборе данных и признаков

Таблица 4.1: Значения $R^2$ для Ridge

| $\alpha$ | mean time | mean score |
|---|---|---|
| 0.1 | 0.652 | 0.768 |
| 1 | 0.596 | 0.770 |
| 10 | 0.575 | 0.776 |
| 100 | 0.579 | **0.777** |
| 1000 | 0.554 | 0.752 |

Минусы RandomForestRegressor:

1) Не подходит для моделирования сложных зависимостей между признаками

2) Может быть склонен к переобучению, если количество деревьев слишком большое или глубина деревьев слишком велика

По сравнению с другими регрессорами, RandomForestRegressor обладает высокой точностью предсказаний и может работать с различными типами признаков. Однако, он не подходит для моделирования сложных зависимостей между признаками и может быть склонен к переобучению, если количество деревьев слишком большое или глубина деревьев слишком велика. В таких случаях, GradientBoostingRegressor или другие регрессоры, такие как Ridge-регрессия или Lasso-регрессия, могут быть более эффективными.

## 4.6 Эксперименты

### 4.6.1 Детали экспериментов

Для подбор гиперпараметров моделей использовался GridSearchCV из библиотеки sklearn. Каждая комбинация гиперпараметров использовалась для кросс-валидации с разделением на 5 частей. В качестве метрики использовался коэффициент детерминации, вычисляемый по формуле 28.

### 4.6.2 Результаты экспериментов

Результы экспериментов показаны в таблицах 4.1, 4.2, 4.3, 4.4 и 4.5. В колонке mean time – среднее время обучения одной модели в секундах, в колонке mean score – среднее значение коэффициента детерминации.

### 4.6.3 Выводы

Лучше всего себя показал градиентный бустинг, чуть хуже случайный лес. Однако обе этих модели учатся слишком долго, а вот обычная линейная регрессия с L2 регуляризацией

Таблица 4.2: Значения $R^2$ для ElasticNet

| $\alpha$ | l1 ratio | mean time | mean score |
|---|---|---|---|
| 0.001 | 0.1 | 35.716 | **0.777** |
| | 0.5 | 39.756 | 0.763 |
| | 0.9 | 23.931 | 0.749 |
| 0.01 | 0.1 | 24.365 | 0.741 |
| | 0.5 | 2.366 | 0.628 |
| | 0.9 | 1.301 | 0.544 |
| 0.1 | 0.1 | 1.100 | 0.510 |
| | 0.5 | 0.667 | 0.144 |
| | 0.9 | 0.614 | 0.001 |

Таблица 4.3: Значения $R^2$ для LinearSVR

| loss | C | mean time | mean score |
|---|---|---|---|
| epsilon insensitive | 0.01 | 90.260 | 0.749 |
| | 0.1 | 159.295 | 0.737 |
| | 1 | 182.357 | 0.716 |
| | 10 | 161.023 | 0.415 |
| | 100 | 160.285 | 0.415 |
| squared epsilon insensitive | 0.01 | 9.569 | **0.778** |
| | 0.1 | 91.483 | 0.773 |
| | 1 | 175.827 | 0.763 |
| | 10 | 158.728 | 0.554 |
| | 100 | 145.408 | 0.433 |

Таблица 4.4: Значения $R^2$ для GradientBoostingRegressor

| max depth | n estimators | mean time | mean score |
|---|---|---|---|
| 4 | 50 | 52.028 | 0.773 |
| | 100 | 103.971 | 0.794 |
| | 150 | 158.188 | 0.801 |
| 6 | 50 | 73.431 | 0.799 |
| | 100 | 154.792 | 0.806 |
| | 150 | 236.340 | 0.808 |
| 8 | 50 | 93.786 | 0.806 |
| | 100 | 202.547 | 0.809 |
| | 150 | 295.853 | **0.809** |

Таблица 4.5: Значения $R^2$ для RandomForestRegressor

| max depth | n estimators | mean time | mean score |
|---|---|---|---|
| 4 | 50 | 34.153 | 0.517 |
| | 100 | 67.943 | 0.517 |
| | 150 | 102.135 | 0.518 |
| 6 | 50 | 50.157 | 0.613 |
| | 100 | 100.250 | 0.612 |
| | 150 | 149.472 | 0.613 |
| 8 | 50 | 64.126 | 0.683 |
| | 100 | 128.198 | 0.683 |
| | 150 | 192.189 | 0.684 |
| None | 50 | 157.689 | 0.807 |
| | 100 | 318.127 | 0.808 |
| | 150 | 450.206 | **0.809** |

Таблица 4.6: Лучшие значения $R^2$ для всех моделей

| model | mean time | mean score |
|---|---|---|
| Ridge | 0.579 | 0.777 |
| ElasticNet | 35.716 | 0.777 |
| LinearSVR | 9.569 | 0.778 |
| GradientBoostingRegressor | 295.853 | **0.809** |
| RandomForestRegressor | 450.206 | 0.809 |

показала себя лишь немного хуже, но при этом обучилась в сотни раз быстрее. Сравнение результатов моделей представлено в таблице 4.6.

Так как лучший коэффициент детерминации больше 0.8, то этой моделью можно с высокой точностью предсказать значение целевой метрики для сорвенования после генерации ноутбука по последовательности семантических классов. В дальнейшем с помощью этого предсказания может быть улучшен генератор.

Главу 4 выполнил Арслан Разин.

# 5 Заключение

В этой работе мы успешно применили метод Proximal Policy Optimization (PPO) для оптимизации генератора последовательностей семантических классов кода по текстовому описанию задачи – одной из двух моделей, лежащих в основе проекта «NL2ML». Конкретно мы добились улучшения метрик BLEU и WER (улучшение последней из которых было особенно значительным), используя линейную комбинацию этих двух метрик в качестве награды для агента задачи обучения с подкреплением, в роли которого выступала наша модель-генератор. В ходе экспериментов мы обнаружили, что метрика BLEU лучше подхо-

дит в качестве награды для нашей задачи, чем метрика WER, основываясь на сравнении полученных значений метрик на тестовой выборке, а также на эмпирическом анализе сгенерированных моделями последовательностей семантических классов.

Оптимизированный генератор, полученный нами в результате нашей работы, позволит генерировать более качественный код для задач машинного обучения на следующем этапе проекта «NL2ML». Кроме того, полученные в результате нашей работы код и экспериментальные результаты использования обучения с подкреплением для оптимизации недифференцируемых метрик генератора упростят дальнейшее использование этих методов для улучшения любых других недифференцируемых наград, не рассмотренных нами в этой работе. Конкретно в перспективе после разработки транслятора семантических классов в код мы хотим запускать сгенерированный транслятором код на реальных данных соответствующей задачи машинного обучения и использовать набранные этим кодом значения метрик машинного обучения (таких как доля правильных ответов или среднеквадратическая ошибка) в качестве наград для оптимизации генератора. Таким образом мы сможем оценивать качество генерируемых последовательностей семантических классов непосредственно по качеству, которое получаемый из этих последовательностей код показывает в реальной задаче.

В настоящий момент оптимизации уже возможно использовать модель, реализованную в главе 4, подавая её предсказания в качестве наград модели. Впоследствии эта модель позволит существенно ускорить оптимизацию, уменьшив количество требуемых вызовов транслятора в процессе обучения. Такое ускорение оптимизации генератора кода для задач машинного обучения может привести к более быстрой и эффективной разработке моделей машинного обучения. Это может помочь улучшить точность и качество моделей, что в свою очередь может привести к более точным прогнозам и решениям. Кроме того, ускорение оптимизации генератора кода может помочь снизить затраты на разработку и поддержку моделей машинного обучения, что является важным фактором для компаний, занимающихся анализом данных и машинным обучением. В целом, ускорение оптимизации генератора кода для задач машинного обучения может привести к повышению производительности и эффективности работы алгоритмов машинного обучения, что является важным фактором для различных отраслей, включая финансы, медицину, транспорт и другие.

Как одно из первых исследований семантических классов кода машинного обучения, введённых в статье [5], наша работа будет полезна для других проектов, использующих эти классы, особенно как промежуточный этап для генерации кода. Обобщение наших результатов на другие виды семантических классификаций кода или другие задачи, кроме задач машинного обучения, требует дальнейших исследований.

# Список литературы


[1] Dzmitry Bahdanau и др. “An Actor-Critic Algorithm for Sequence Prediction”. В: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=SJDaqqveg.

[2] Rudy Bunel и др. “Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis”. В: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=H1Xw62kRZ.

[3] Z. Chen и др. “SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair”. В: *IEEE Transactions on Software Engineering* 47.09 (сент. 2021), с. 1943—1959. ISSN: 1939-3520. DOI: 10.1109/TSE.2019.2940179.

[4] Pierre Dognin и др. “ReGen: Reinforcement Learning for Text and Knowledge Base Generation using Pretrained Language Models”. В: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online и Punta Cana, Dominican Republic: Association for Computational Linguistics, нояб. 2021, с. 1084—1099. DOI: 10.18653/v1/2021.emnlp-main.83. URL: https://aclanthology.org/2021.emnlp-main.83.

[5] Anastasia Drozdova и др. “Code4ML: a Large-scale Dataset of annotated Machine Learning Code”. В: *arXiv preprint, arXiv:2210.16018* (2022). DOI: 10.48550/ARXIV.2210.16018. URL: https://arxiv.org/abs/2210.16018.

[6] Dan Hendrycks и др. “Measuring Coding Challenge Competence With APPS”. В: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=sD93GOzH3i5.

[7] Hui Jiang и др. “Exploring Dynamic Selection of Branch Expansion Orders for Code Generation”. В: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, авг. 2021, с. 5076—5085. DOI: 10.18653/v1/2021.acl-long.394. URL: https://aclanthology.org/2021.acl-long.394.

[8] *Kaggle: Your Home for Data Science*. URL: https://kaggle.com (дата обр. 03.05.2023).

[9] Diederik P. Kingma и Jimmy Ba. “Adam: A Method for Stochastic Optimization”. В: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Под ред. Yoshua Bengio и Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[10] Vijay Konda и John Tsitsiklis. “Actor-Critic Algorithms”. В: *Advances in Neural Information Processing Systems*. Под ред. S. Solla, T. Leen и K. Müller. Т. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[11] Philippe Laban и др. “Keep It Simple: Unsupervised Simplification of Multi-Paragraph Text”. В: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, авг. 2021, с. 6365—6378. DOI: 10.18653/v1/2021.acl-long.498. URL: https://aclanthology.org/2021.acl-long.498.

[12] Hung Le и др. “CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning”. В: *Advances in Neural Information Processing Systems*. Под ред. Alice H. Oh и др. 2022. URL: https://openreview.net/forum?id=WaGvb7OzySA.

[13] Martijn van Otterlo и Marco Wiering. “Reinforcement Learning and Markov Decision Processes”. В: *Reinforcement Learning: State-of-the-Art*. Под ред. Marco Wiering и Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, с. 3—42. ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_1. URL: https://doi.org/10.1007/978-3-642-27645-3_1.

[14] Long Ouyang и др. “Training language models to follow instructions with human feedback”. В: *Advances in Neural Information Processing Systems*. Под ред. S. Koyejo и др. Т. 35. Curran Associates, Inc., 2022, с. 27730—27744. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

[15] Kishore Papineni и др. “BLEU: A Method for Automatic Evaluation of Machine Translation”. В: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL ’02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, с. 311—318. DOI: 10.3115/1073083.1073135. URL: https://doi.org/10.3115/1073083.1073135.

[16] Steven J. Rennie и др. “Self-Critical Sequence Training for Image Captioning”. В: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Июль 2017.

[17] Victor Sanh и др. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. В: *arXiv preprint, arXiv:1910.01108* (2019). URL: https://arxiv.org/abs/1910.01108.

[18] John Schulman и др. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. В: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Под ред. Yoshua Bengio и Yann LeCun. 2016. URL: http://arxiv.org/abs/1506.02438.

[19] John Schulman и др. “Proximal Policy Optimization Algorithms”. В: *arXiv preprint, arXiv: 1707.06347* (2017). DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[20] John Schulman и др. “Trust Region Policy Optimization”. В: *Proceedings of the 32nd International Conference on Machine Learning*. Под ред. Francis Bach и David Blei. Т. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, июль 2015, с. 1889—1897. URL: http://proceedings.mlr.press/v37/schulman15.pdf.

[21] Parshin Shojaee и др. “Execution-based Code Generation using Deep Reinforcement Learning”. В: *arXiv preprint, arXiv:2301.13816* (2023). DOI: 10.48550/ARXIV.2301.13816. URL: https://arxiv.org/abs/2301.13816.

[22] Nisan Stiennon и др. “Learning to summarize with human feedback”. В: *Advances in Neural Information Processing Systems*. Под ред. H. Larochelle и др. Т. 33. Curran Associates, Inc., 2020, с. 3008—3021. URL: https://proceedings.neurips.cc/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf.

[23] Richard S Sutton и др. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. В: *Advances in Neural Information Processing Systems*. Под ред. S. Solla, T. Leen и K. Müller. Т. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[24] Ramakrishna Vedantam, C. Lawrence Zitnick и Devi Parikh. “CIDEr: Consensus-based image description evaluation”. В: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, с. 4566—4575. DOI: 10.1109/CVPR.2015.7299087.

[25] Xin Wang и др. “Compilable Neural Code Generation with Compiler Feedback”. В: *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, май 2022, с. 9—19. DOI: 10.18653/v1/2022.findings-acl.2. URL: https://aclanthology.org/2022.findings-acl.2.

[26] Yue Wang и др. “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation”. В: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online и Punta Cana, Dominican Republic: Association

for Computational Linguistics, нояб. 2021, с. 8696—8708. DOI: 10.18653/v1/2021.emnlp-main.685. URL: https://aclanthology.org/2021.emnlp-main.685.

[27] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. В: *Mach. Learn.* 8.3–4 (май 1992), с. 229—256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[28] Victor Zhong, Caiming Xiong и Richard Socher. “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. В: *arXiv preprint, arXiv:1709.00103* (2017). DOI: 10.48550/ARXIV.1709.00103. URL: https://arxiv.org/abs/1709.00103.

[29] Daniel M. Ziegler и др. “Fine-Tuning Language Models from Human Preferences”. В: *arXiv preprint, arXiv:1909.08593* (2019). URL: https://arxiv.org/abs/1909.08593.