Санкт-Петербургский государственный университет

Кафедра системного программирования
Математическое обеспечение и администрирование информационных систем

Елисеева Александра Михайловна

Автодополнение сообщений к коммитам в среде разработки IntelliJ IDEA

Выпускная квалификационная работа бакалавра

Научный руководитель:
доцент кафедры СП, к.т.н. Брыксин Т. А.

Консультант:
программист-исследователь
в АНО ДПО «Научно-исследовательский и образовательный центр ДжетБрейнс»
Соколов Я. С.

Рецензент:
заведующий центром анализа данных и машинного обучения
в НИУ ВШЭ–Санкт-Петербург
Шпильман А. А.

Санкт-Петербург
2021

SAINT-PETERSBURG STATE UNIVERSITY

System Programming
Software and Administration of Information Systems

Aleksandra Eliseeva

Completion of commit messages in IntelliJ IDEA

Bachelor's Thesis

Scientific supervisor:
Associate professor, Ph.D. Timofey Bryksin

Advisor:
Researcher
at Research and Education Center JetBrains INO APE
Yaroslav Sokolov

Reviewer:
Director of Center for Data Analysis and Machine Learning
at HSE University–Saint Petersburg
Aleksei Shpilman

Saint-Petersburg
2021

Оглавление

Введение

В настоящее время очень распространены системы автодополнения. Они предлагают пользователям варианты продолжения введенного ими начала некоторого текста и тем самым делают ввод текста более удобным и быстрым. Повсеместно встречается автодополнение текстов на естественном языке (например, в поисковых системах [46], в клавиатурах мобильных устройств [25] или в почтовом сервисе Gmail [28]), также многие современные интегрированные среды разработки (Integrated Development Environment, IDE) поддерживают автодополнение кода: этот инструмент часто используется и полезен для разработчиков [42, 57].

Часто разработчики пишут не только программный код, но и небольшие тексты на естественном языке — в основном это некоторые пояснения, направленные на то, чтобы облегчить сопровождение программ. Так, в системах контроля версий (например, git[1]), использующихся при разработке многих программных проектов, есть возможность при внесении изменений (коммите) оставить сообщение, поясняющее, что изменилось или зачем.

Недавно в плагине Grazie[2] для среды разработки IntelliJ IDEA появилась функциональность автодополнения сообщений к коммитам. Плагин Grazie направлен в первую очередь на работу с естественным языком: в нем учитываются особенности стиля письма каждого пользователя, но не используется информация о внесенных в коммитах изменениях. Сообщения к коммитам обычно связаны с внесенными изменениями, поэтому такая информация может быть важна для их автодополнения, а её отсутствие может приводить к более плохому качеству.

Задача автодополнения сообщений к коммитам на основе специфичной для коммитов информации не встречается в литературе, но при этом активно исследуется задача автоматической генерации таких сообщений [32, 27, 43, 18]. Эти задачи очень близки и отличаются только

[1]git: `https://git-scm.com/` (дата обращения: 10.05.2021)

[2]Grazie: `https://plugins.jetbrains.com/plugin/16136-grazie-professional` (дата обращения: 10.05.2021)

тем, что при автодополнении есть введенный пользователем префикс сообщения, а автоматическая генерация предполагает построение целого сообщения. Для решения этих задач обычно используются идентичные методы, поэтому многие подходы для автоматической генерации сообщений к коммитам на основе изменений хорошо подойдут для использования в системе автодополнения.

В данной работе предлагается улучшить существующую в IntelliJ IDEA систему автодополнения сообщений к коммитам с помощью адаптации и улучшения методов для автоматической генерации. В дальнейшем предложенный метод можно будет встроить в плагин Grazie для IntelliJ IDEA.

Постановка задачи

Целью данной работы является улучшение существующей системы автодополнения сообщений к коммитам в среде разработки IntelliJ IDEA. Для достижения этой цели были сформулированы следующие задачи.

- Провести обзор систем автодополнения, методов для задачи генерации сообщений к коммитам и открытых наборов данных, подходящих для обучения методов автодополнения сообщений к коммитам на основе изменений.
- Воспроизвести лучший на данный момент подход к генерации сообщений к коммитам, адаптировать его для задачи автодополнения.
- Предложить улучшения к методу автодополнения сообщений к коммитам на основе изменений.
- Провести апробацию предложенного метода в сравнении с существующей в среде разработки IntelliJ IDEA системой и с методом, основанным на лучшем на данный момент подходе для автоматической генерации.

1. Обзор предметной области

1.1. Системы автодополнения

В общем виде задача автодополнения заключается в том, чтобы предсказать следующую часть текста на основе его начала и, возможно, какой-либо ещё дополнительной информации (контекста).

И контекст, и следующая часть текста обычно представлены в виде последовательностей некоторых токенов. В случае текстов на естественном языке это могут быть символы, слова или части слов, в случае программного кода также возможны различные варианты на основе абстрактных синтаксических деревьев или другой структурной информации.

Рассмотрим несколько распространенных подходов к автодополнению в целом, а также существующую систему автодополнения сообщений к коммитам системы контроля версий в среде разработки IntelliJ IDEA.

1.1.1. Методы

Ранжирование списка кандидатов

В части работ по автодополнению используется следующий подход: строить на основе контекста список кандидатов для следующей части текста и некоторым образом ранжировать его, предлагая пользователю несколько наиболее подходящих вариантов.

Обычно кандидатами считаются все возможные варианты продолжения текста из некоторого множества, например, во многих работах по автодополнению запросов (query autocompletion) кандидаты выбираются из истории запросов для поисковой системы [7], а в работах по автодополнению кода — из множества корректных идентификаторов, полученного с помощью инструментов статического анализа [24].

Один из самых простых методов ранжирования заключается в том, чтобы сортировать кандидатов по частоте появления в истории использования системы автодополнения [7], он называется MostPopularComple-

tion. Также распространен подход к ранжированию, идейно близкий к методу k ближайших соседей: контекст и кандидаты некоторым образом представляются в виде векторов, затем кандидаты сортируются по значениям некоторой функции расстояния между ними и контекстом.

В ранних работах для построения векторных представлений из данных извлекались вручную выбранные признаки, основанные на частоте появления в истории [7], времени [12], информации о пользователе [39], или, в случае автодополнения кода, на специфичной для кода информации [9]. В одной из новых работ [24] используются современные методы построения векторных представлений из области естественного языка, не требующие ручного выбора признаков.

Системы автодополнения, основанные на данном подходе, могут быть достаточно эффективными, но предлагаемые ими варианты для следующей части текста ограничены списком кандидатов. Такие системы обычно используются для автодополнения небольших последовательностей, например, окончания запроса в поисковой системе или следующего уникального идентификатора (имени переменной, метода и др.) в программном коде.

Генерация с помощью языкового моделирования

Рассмотрим еще один часто встречающийся подход к автодополнению. Можно построить некоторую функцию, приближающую распределение следующей части текста на основе контекста (см. Формулу 1), и использовать её для генерации.

$$M(c_0, \ldots, c_T, \Theta) \approx p(m_0, \ldots, m_N \mid c_0, \ldots, c_T), \tag{1}$$

где $p(m_0, \ldots, m_N \mid c_0, \ldots, c_T)$ — распределение следующей части текста на основе контекста, $m_0, \ldots, m_N$ — последовательность токенов в следующей части текста, $c_0, \ldots, c_T$ — последовательность токенов в контексте, M — функция, приближающая распределение, Θ — её настраиваемые параметры.

Сделав распространенное предположение, что вероятность токена

зависит только от конкретного количества предыдущих токенов [31], придем к равенству из Формулы 2.

$$p(m_0, \ldots, m_N \mid c_0, \ldots, c_T) = \prod_{i=1}^{n} p(m_i \mid c_0, \ldots, c_T, m_0, \ldots, m_{i-1}) \qquad (2)$$

Настройка параметров такой функции проводится путем минимизации кросс-энтропийной функции потерь (cross-entropy loss) (см. формулу 3).

$$\min_{\Theta} \; -\frac{1}{N} \sum_{i=1}^{N} \log \hat{p}_i, \qquad (3)$$

где $\hat{p}_i$ — предсказанная функцией M вероятность появления на i-той позиции истинного токена m_i, Θ — настраиваемые параметры M.

В области обработки естественного языка (Natural Language Processing, NLP) задача приближения распределения последовательностей токенов встречается очень часто [33] и называется языковым моделированием. Для генеративных задач, к которым можно отнести и данный подход к автодополнению, обычно используются авторегрессионные языковые модели [30, 35, 34] — языковые модели, в которых при предсказании вероятности каждого токена учитываются только предыдущие токены, как в Формуле 2.

Данный подход обычно требует бóльших вычислительных затрат, чем ранжирование, но основанные на нем системы автодополнения способны генерировать произвольные последовательности токенов из имеющегося словаря, что особенно ценно для автодополнения более длинных последовательностей, к которым иногда можно отнести и сообщения к коммитам.

Нейронные сети для языкового моделирования

В настоящее время чаще всего в качестве языковых моделей применяются нейронные сети, поэтому рассмотрим две наиболее популярные архитектуры.

Рекуррентные нейронные сети (recurrent neural network,

RNN) — разновидность нейронных сетей, направленная на обработку данных, представленных в виде последовательности, и хорошо подходящая для задачи языкового моделирования.

Эта архитектура проходит по входным данным последовательно и на каждом шаге выполняет вычисления над текущим токеном и собственным скрытым состоянияем (hidden state). В классическом варианте RNN [50] вычисления на каждом шаге представляют собой линейное преобразование входных данных, к результату которого применяется нелинейная функция активации (см. Формулу 4).

$$h_t = \sigma(h_{t-1} \cdot W_h + x_t \cdot W_x) \tag{4}$$

Здесь h_t — скрытое состояние сети на шаге t, x_t — t-тый элемент входной последовательности, W_h и W_t — настраиваемые параметры, σ — некоторая нелинейная функция, часто используется tanh или логистическая функция.

Результат этих вычислений становится новым значением скрытого состояния и используется для обработки следующих элементов последовательности, а также может использоваться как выходное значение для данного шага, над котором можно производить дальнейшие вычисления (см. Рис. 1).

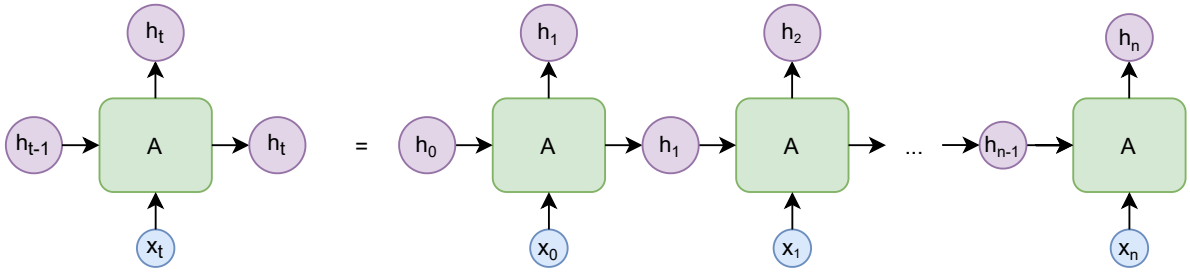


Рис. 1: Схема применения рекуррентной нейронной сети для обработки последовательности данных. Здесь A — функция, вычисляемая над входными данными, x_t — t-тый элемент входной последовательности, h_t — скрытое состояние на шаге t.

Для оптимизации параметров нейронных сетей почти всегда используются методы на основе градиентного спуска. Вид вычислений (см. Формулу 4) в классических RNN приводит к проблемам затуха-

ющих (слишком маленьких) или взрывающихся (слишком больших) градиентов, из-за этого они не очень хорошо обрабатывают долгосрочные зависимости [29]. Существуют модификации, не обладающие этим недостатком: управляемые рекуррентные блоки (Gated Recurrent Unit, GRU) [36] и долгая краткосрочная память (Long Short-Term Memory, LSTM) [29].

Рекуррентные нейронные сети и их модификации используются как при автодополнении текстов на естественном языке [28, 45, 49], так и при автодополнении кода [14, 48].

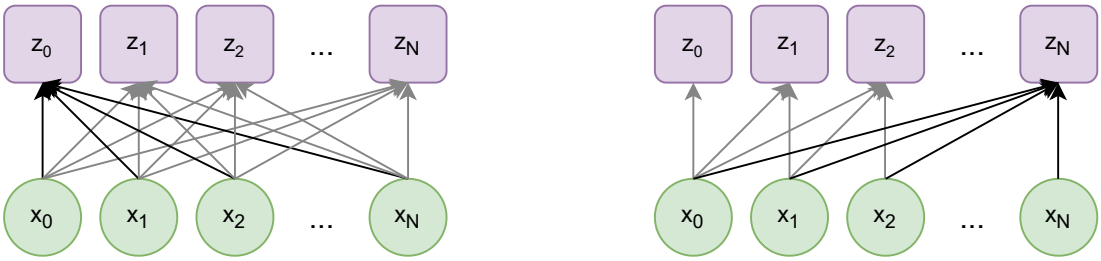
В большинстве работ рассматриваются одиночные нейронные сети, но встречается [28] и архитектура кодировщик-декодировщик (encoder-decoder) [37], состоящая из двух нейронных сетей: первая из них (кодировщик) используется для получения векторного представления входной последовательности, вторая (декодировщик) — для генерации выходной последовательности токенов на основе этого векторного представления.

У рекуррентных нейронных сетей есть важный недостаток: так как их архитектура предполагает последовательную обработку данных, распараллеливать вычисления по входным последовательностям затруднительно. На данный момент распространена другая архитектура, Transformer, не основанная на рекуррентном подходе.

Архитектура Transformer [2] состоит из кодировщика и декодировщика. При этом и кодировщик, и декодировщик содержат только слои, основанные на механизме внимания [5], полносвязные слои и несколько дополнительных операций. Такой подход позволяет эффективно распараллеливать вычисления, тем самым ускоряя обучение моделей, а также показывает лучшие результаты для многих задач из области обработки естественного языка.

Кодировщик и декодировщик в архитектуре Transformer похожи, но отличаются способом вычисления механизма внимания: в кодировщике при подсчете результата для каждого входного элемента учитываются все остальные токены последовательности, а в декодировщике — только идущие до него (см. Рис 2). Также в декодировщике присутствуют

дополнительные слои, обрабатывающие выход кодировщика.



(a) Кодировщик (b) Декодировщик

Рис. 2: Схема применения кодировщика и декодировщика архитектуры Transformer для обработки последовательности данных. Здесь x_t — t-тый элемент входной последовательности, z_t — результат применения механизма внимания, соответствующий t-тому элементу входной последовательности.

Таким образом, именно декодировщик из архитектуры Transformer является авторегресионной языковой моделью. Часто его применяют отдельно, в таком случае при обработке каждого токена учитываются только предыдущие токены последовательности. Например, основанные на декодировщике архитектуры Transformer и предобученные на больших наборах данных из открытых источников модели GPT, GPT-2 и GPT-3 показывают очень хорошее качество генерации [30, 35, 34].

В работах по автодополнению встречаются как полноценные кодировщик-декодировщик модели Transformer [28, 15], так и модели, основанные на декодировщике из этой архитектуры [28, 31].

1.1.2. Метрики для оценки качества

Для оценки систем автодополнения обычно применяются метрики качества предсказания следующего токена. Среди них наиболее популярны следующие:

- Accuracy@K (Точность)

  Эта метрика подсчитывает, для какой части токенов выходной последовательности среди K предсказанных системой вариантов

токенов с наибольшей вероятностью есть настоящий.

$$Accuracy@K = \frac{1}{N}\sum_{i=1}^{N}[real_i \in \{pred_{i_1}, \ldots, pred_{i_K}\}]$$

Здесь N — количество токенов в выходной последовательности, $real_i$ — настоящий токен для i-той позиции, $pred_{i_j}$ — j-тый предсказанный токен для i-той позиции.

- MRR@K (Mean Reciprocal Rank — среднеобратный ранг)

  Эта метрика позволяет учитывать, на какой позиции среди K предсказанных системой вариантов с наибольшей вероятностью находится настоящий токен: чем выше, тем больше её значение.

$$MRR@K = \frac{1}{N}\sum_{i=1}^{N} RR_i@K$$

$$RR_i@K = \begin{cases} \frac{1}{\min\{k \in [1...K] : pred_{i_k} = real_i\}}, \text{ если } real_i \in \{pred_{i_1}, \ldots, pred_{i_K}\} \\ 0, \text{ иначе} \end{cases}$$

Здесь N — количество токенов в выходной последовательности, $real_i$ — настоящий токен для i-той позиции, $pred_{i_j}$ — j-тый предсказанный токен для i-той позиции.

1.1.3. Система автодополнения сообщений к коммитам в IntelliJ IDEA

Существующая в среде разработки IntelliJ IDEA система автодополнения сообщений к коммитам представлена в плагине Grazie[3].

В Grazie для решения этой задачи используется авторегресионная языковая модель distilGPT-2[4]. distilGPT-2 представляет собой уменьшенную версию модели GPT-2 [35], основанной на декодировщике архитектуры Transformer. Она специальным образом обучена на большом

[3]Grazie: `https://plugins.jetbrains.com/plugin/16136-grazie-professional` (дата обращения: 10.05.2021)

[4]Веса и описание distilGPT-2: `https://huggingface.co/distilgpt2` (дата обращения: 10.05.2021)

неразмеченном наборе данных на английском языке с использованием оригинальной GPT-2 [20].

Входными данными для языковой модели являются история сообщений от того же пользователя и префикс текущего сообщения. На основе этого она возвращает несколько наиболее вероятных вариантов продолжения текущего сообщения (см. Рис 3).

С помощью истории сообщений в Grazie учитывается стиль письма каждого пользователя.

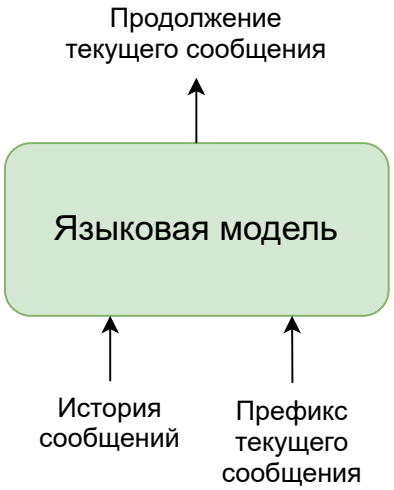


Рис. 3: Подход для автодополнения сообщений к коммитам в плагине Grazie.

1.2. Автоматическая генерация сообщений к коммитам

Рассмотрим существующие подходы к автоматической генерации сообщений к коммитам, чтобы выбрать лучший метод, который легко адаптировать под задачу автодополнения.

1.2.1. Методы

Ручное задание шаблонов

Ранние работы были направлены в первую очередь на генерацию сообщений с информацией о том, что изменилось. Большинство из них

использовали для генерации предварительно заданные наборы правил. Для извлечения изменений кода было предложено несколько способов, связанных с определением различий в AST. Примерами таких работ являются DeltaDoc [10] и ChangeScribe [13].

Данный подход позволяет развернуто описать, что и где изменилось в коде. Тем не менее, большие сообщения, сгенерированные по шаблону, не всегда удобны, в них обычно нет информации о том, почему было сделано изменение, и для задачи автодополнения этот подход не подходит.

Языковые модели

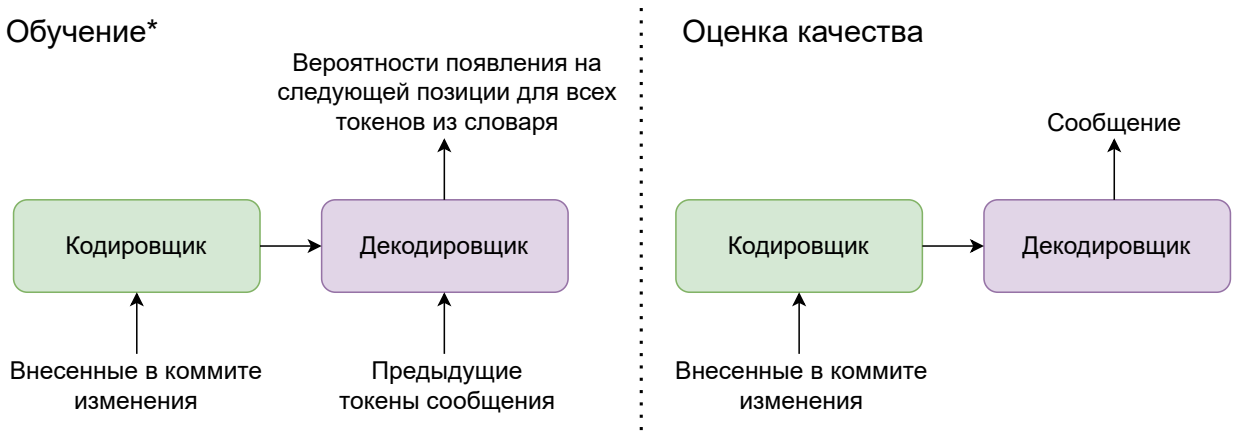
В дальнейшем появились работы, изучающих применение языковых моделей для задачи автоматической генерации сообщений к коммитам.

Большинство исследователей относят её к задачам «последовательность-последовательность» (sequence-to-sequence) и используют для её решения архитектуру кодировщик-декодировщик [37]: входными данными для кодировщика является последовательность токенов из внесенных в коммите изменений, на выходе декодировщик генерирует последовательность токенов из сообщения к коммиту.

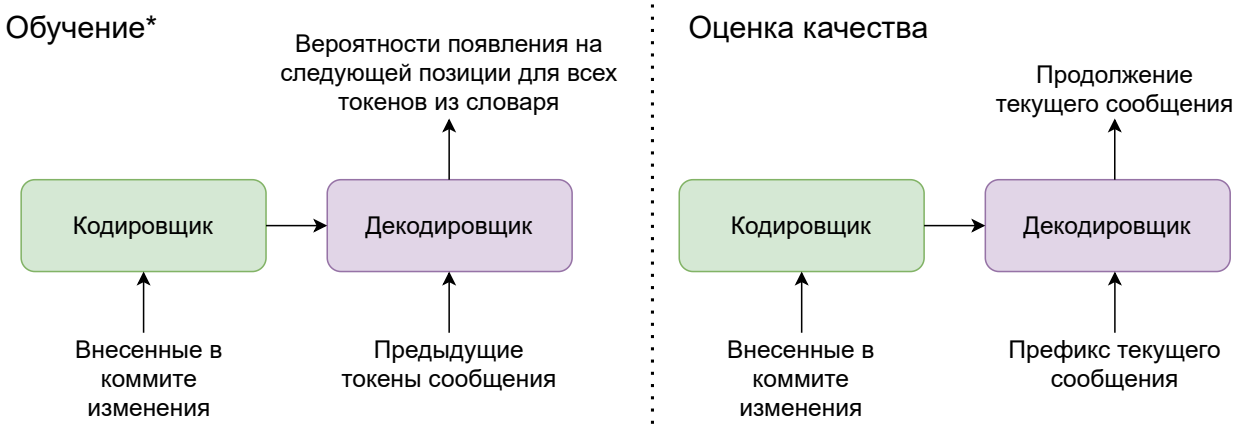
В такой постановке задача автоматической генерации практически не отличается от задачи автодополнения. Обучение (настройка параметров) проводится точно так же с использованием кросс-энтропийной функции потерь (см. Формулу 3), при оценке качества вместо генерации последовательности с нуля используется префикс (см. Рис 4).

В нескольких исследованиях по автоматической генерации сообщений к коммитам в качестве языковых моделей применялись рекуррентные нейронные сети. Так, в одной из первых работ с таким подходом [32] была рассмотрена архитектура кодировщик-декодировщик с LSTM и механизмом внимания [5]. Предложенный подход был назван NMT, так как авторы основывались на идеях из области нейронного машинного перевода (Neural Machine Translation, NMT).

В последующей работе [27] обратили внимание, что в программном коде много уникальных идентификаторов (например, названия ме-



(a) Автоматическая генерация сообщений к коммитам



(b) Автодополнение сообщений к коммитам

Рис. 4: Схема обучения и оценки качества двух приложений основанного на изменениях метода.
* При обучении указанные действия проводятся для каждой позиции выходной последовательности (сообщения).

тодов, имена переменных и классов), и использовали сеть указатель-генератор (pointer-generator network) [53]: модификацию encoder-decoder LSTM, позволяющую не добавлять все редко встречающиеся токены в словарь и тем самым улучшить способность к обобщению. Предложенный подход был назван PtrGNCMsg.

В одном из последних исследований вместо рекуррентных нейронных сетей была опробована архитектура Transformer [18]. Также в данной работе было предложено особым способом предобучать модель на неразмеченных данных перед обучением для генерации сообщений. Обе идеи помогли улучшить качество генерации по сравнению с преды-

дущими работами на основе RNN. Предложенный подход был назван CoreGen.

Метод k ближайших соседей

Метод k ближайших соседей заключается в том, чтобы для каждого нового примера находить k наиболее близких примеров из обучающей выборки и агрегировать их значения целевой переменной для получения возвращаемого значения.

Этот подход был предложен для задачи автоматической генерации сообщений к коммитам в работе [43]: после анализа результата и данных из работы NMT [32] авторы обратили внимание, что в большинстве случаев, когда их модель генерирует сообщения, отмеченные экспертами как «хорошие», в обучающей выборке присутствуют очень похожие примеры. Предложенный подход был назван NNGen.

В приложении для задачи автоматической генерации сообщений к коммитам алгоритм выглядит так: изменения некоторым способом представляются в виде векторов, для каждого нового примера ищется k наиболее близких по косинусному расстоянию между векторными представлениями изменений примеров из обучающей выборки, из них выбирается пример с наибольшим значением метрики BLEU-4 [4] между исходными текстовыми представлениями изменений, как результат возвращается сообщение от него.

В работе NNGen для построения векторных представлений изменений использовался очень простой метод — мешок слов (bag-of-words). Существует также исследование CC2Vec [11], где для построения векторного представления изменений предлагается новый метод, учитывающий их структуру.

Данный подход достаточно прост и, если использовать мешок слов или другие готовые методы для построения векторных представлений, вообще не требует временных и вычислительных затрат на обучение. Тем не менее, он обладает несколькими недостатками: во-первых, велика вероятность, что уникальные идентификаторы из программного кода (например, названия методов, имена переменных и классов) для

нового примера будут отличаться от уже существующих, во-вторых, качество метода ближайших соседей сильно зависит от обучающей выборки, он не способен производить хорошие сообщения, если в ней не было похожих. Также его не так просто адаптировать под задачу автодополнения, так как необходимо учитывать не только внесенные изменения, но и введённый пользователем префикс.

1.2.2. Открытые наборы данных

Многие исследования основаны на открытом наборе данных[5] из работы из работы NMT [32]. Данные представляют собой пары изменений (выводов команды `git diff`) из коммитов и сообщений к ним. Дополнительной метаинформации (например, об авторе, дате и репозитории коммитов) в опубликованной отфильтрованной версии нет.

Авторы работы NMT собрали данные о коммитах из 1000 открытых Java репозиториев с GitHub с наибольшим количеством звезд. Перед дальнейшей работой они также тщательно отфильтровали данные. Кратко опишем процесс фильтрации.

1. Из сообщений к коммитам удаляются все предложения после первого.

2. Из сообщений к коммитам и изменений в коде удаляются уникальные идентификаторы (номер issue, хэш коммита).

3. Отбрасываются коммиты, в которых происходит слияние веток (merge) или отмена внесенных ранее изменений (rollback).

4. Отбрасываются коммиты, количество токенов в изменениях или сообщениях которых превосходит следующие значения:

    - 100 токенов для изменений;
    - 30 токенов для сообщений.

[5]NMT (Jiang et al.) online appendix: `ссылка` (дата обращения: 10.05.2021)

5. Отбрасываются коммиты, сообщения в которых не содержат грамматической структуры глагол-прямое дополнение (Verb-Direct Object). Пример предложения с такой грамматической структурой: *Remove redundant commands in travis config.*

   Пример предложения с отличающейся грамматической структурой: *When INFO_FORMAT_CHANGED event comes in before the renderer was initialized, go back to read from source again.*

Авторы работы NNGen [43] сообщили, что в наборе данных из исследования NMT осталось около 16% плохих примеров: сгенерированных специальными программами или просто неинформативных сообщений. Они отбросили такие примеры и опубликовали еще более отфильтрованную версию датасета[6].

Дальнейшие работы используют в основном набор данных из работы NNGen, поэтому приведем краткую статистику именно по этому датасету.

Раздел датасета	Количество примеров
Обучающая выборка	22 112
Валидационная выборка	2 511
Тестовая выборка	2 521
Всего	27 144

Таблица 1: Количество примеров в датасете из работы NNGen [43]

Статистика по количеству токенов	В изменениях	В сообщениях
Среднее значение	68	7
Стандартное отклонение	22	4
Медиана	75	6

Таблица 2: Статистика по количеству токенов в датасете из работы NNGen [43]. Токенизация проводилась по пробелам.

[6]NNGen (Liu et al.) online appendix: `ссылка` (дата обращения: 10.05.2021)

1.2.3. Метрики оценки качества

Для оценки качества автоматической генерации сообщений к коммитам используются метрики из области нейронного машинного перевода, оценивающие «близость» кандидата (машинного перевода) к одному или нескольким заданным эталонам (переводам, выполненным человеком-экспертом). Среди них наиболее популярны следующие:

1. BLEU (BiLingual Evaluation Understudy) [4].

   Для вычисления BLEU сначала для каждого n подсчитывается p_n — модифицированная точность n-грамм (modified n-gram precision).

$$p_n = \frac{\sum_{c \in \{\text{candidates}\}} \sum_{\text{n-gram} \in c} \min(\text{ref\_count(n-gram)}, \text{count(n-gram)})}{\sum_{c' \in \{\text{candidates}\}} \sum_{\text{n-gram}' \in c'} \text{count(n-gram}')} \tag{5}$$

   Здесь {candidates} — множество кандидатов для всех тестовых примеров, n-gram — N-грамма (последовательность из n токенов), count(n-gram) — сколько раз N-грамма n-gram встречается в кандидате c, ref_count(n-gram) — максимальное количество раз, которое N-грамма n-gram встречается в каком-либо из эталонов.

   Данное значение основано на метрике точности, применяемой для задачи бинарной классификации:

$$precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{6}$$

   Для одного примера оно представляет собой отношение количества совпадений N-грамм кандидата и эталона к количеству N-грамм в кандидате.

Итоговое значение метрики получается следующим образом:

$$BLEU = BP \cdot \exp(\sum_{i=1}^{N} \log(p_n))$$
$$BP = \begin{cases} 1, \text{ если } c > r \\ e^{(1-r/c)}, \text{ если } r > c \end{cases} \quad (7)$$
$$r \text{ — сумма длин эталонов}$$
$$c \text{ — сумма длин кандидатов}$$

Часто используется BLEU-4, то есть вариант, где при подсчете метрики учитывается количество совпадений между n-граммами для $n = 1, 2, 3, 4$.

2. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [40].

   ROUGE представляет собой набор метрик для оценивания качества генерации текста и машинного перевода. Обратим внимание на часть из них:

   - ROUGE-N

     Данная метрика, в отличие от BLEU, основана не на точности (см. Формулу 6), а на полноте — другой метрике для бинарной классификации:

     $$recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad (8)$$

     Для вычисления ROUGE-N подсчитывается отношение количества совпадений N-грамм кандидата и эталона к количеству N-грамм в эталонах.

   - ROUGE-L

     Данная метрика основана на наибольшей общей подпоследовательности (Longest Common Subsequence, LCS) и F-мере —

еще одной метрике для бинарной классификации:

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \tag{9}$$

В случае ROUGE-L F-мера подсчитывается следующим образом:

$$P_{LCS} = \frac{len(LCS(X, Y))}{len(X)} \tag{10}$$

$$R_{LCS} = \frac{len(LCS(X, Y))}{len(Y)} \tag{11}$$

$$F_{LCS} = \frac{(1 + \beta^2) \cdot P_{LCS} \cdot R_{LCS}}{R_{LCS} + \beta^2 \cdot P_{LCS}} \tag{12}$$

Здесь X — эталонное предложение, Y — кандидат.

3. METEOR (Metric for Evaluation of Translation with Explicit ORdering) [6].

   В данной метрике в отличие от предыдущих, где учитывается только точное совпадение между словами в n-граммах, учитываются также совпадения основ слов и синонимы. Для сравнения униграмм в эталоне и кандидате считаются и полнота, и точность, затем они комбинируются, используя формулу F_3-меры:

$$F_3 = (1+3^2) \cdot \frac{Precision \cdot Recall}{(3^2 \cdot Precision) + Recall} = \frac{10 \cdot Precision \cdot Recall}{Recall + 9 \cdot Precision} \tag{13}$$

   Чтобы учитывать более длинные совпадения, F_3 мера умножается на специально заданный штраф.

1.2.4. Сравнение методов

Сравнение рассмотренных выше методов приведено в Таб. 3. Здесь NMT — кодировщик-декодировщик с LSTM и механизмом внимания из работы [32], PtrGNCMsg — сеть указатель-генератор [27], NNGen —

метод k ближайших соседей [43], CoreGen $_{\text{II}}$ — Transformer без предобучения [18], CoreGen — Transformer с предобучением [18].

Модель	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
NMT	14.17	21.29	12.19	20.85	12.99
PtrGNCMsg	9.78	23.66	9.61	23.67	11.41
NNGen	16.43	25.86	15.52	24.46	14.03
CoreGen $_{\text{II}}$	18.74	30.65	18.06	28.86	15.18
CoreGen	**21.06**	**32.87**	**20.17**	**30.85**	**16.53**

Таблица 3: Сравнение методов для автоматической генерации сообщений к коммитам на наборе данных из работы NNGen [43] по метрикам для нейронного машинного перевода (таблица взята из работы [18]).

1.3. Предобученные модели

В области обработки естественного языка в настоящее время распространены модели, основанные на архитектуре Transformer, обучение которых предполагает две стадии:

1. Предобучение на большом неразмеченном наборе данных.
2. Дообучение под конкретную задачу на небольшом размеченном наборе данных.

Большу́ю роль в этом сыграла модель BERT [3], основанная на кодировщике архитектуры Transformer. Для неё существует множество модификаций (например, RoBERTa [51], StructBERT [56], ALBERT [1]), которые и сейчас показывают очень хорошее качество для нескольких решаемых в NLP задач. Также существует несколько работ, исследующих обучение модели BERT на исходном коде [22, 38, 16]. При последующем применении к различным задачам, связанных с исходным кодом, такие модели показывают неплохие результаты. Например, в работе CodeBERT [16] модель RoBERTa [51] предобучают на большом датасете с примерами на нескольких языках программирования (Ruby, Javascript, Go, Python, Java, PHP). Авторы подают на вход модели как

пары естественный язык-исходный код (natural language-programming language, NL-PL), так и только исходный код. При этом используется специальная целевая функция.

Помимо BERT и ее различных модификаций подобным образом применяются также модели от OpenAI, основанные на декодировщике архитектуры Transformer: GPT, GPT-2, GPT-3 [30, 35, 34]. В случае этих моделей часто можно и без дообучения под конкретную задачу получить очень хорошее качество генерации.

Для автоматической генерации сообщений к коммитам с помощью языковых моделей в большинстве работ используется архитектура кодировщик-декодировщик. Исследования по предобучению целых кодировщик-декодировщик моделей Transformer на больших датасетах с программным кодом начали появляться только в 2021 году, и на данный момент нет завершенных работ с опубликованными предобученными весами [61, 58, 17].

Обычно предобученные модели, основанные на кодировщике или декодировщике, применяются сами по себе, но существует исследование, в котором подробно обозревается, как инициализация кодировщика или декодировщика целой модели Transformer весами различных предобученных моделей (BERT, RoBERTa, GPT-2) влияет на качество [52]. Авторы сообщают, что таким образом для нескольких задач из области NLP можно добиться лучшего качества, чем при обучении модели Transformer с нуля.

1.4. Выводы

Лучшим на данный момент методом для автоматической генерации сообщений к коммитам является модель CoreGen архитектуры Transformer [18]. Архитектура Transformer используется и в системах автодополнения, таким образом, хорошо подходит для нашей задачи, поэтому в данной работе будут рассмотрены методы на основе этой архитектуры.

Предложенная авторами CoreGen целевая функция для предобуче-

ния позволяет улучшить качество, но, во-первых, весов предобученной модели из этой работы нет в открытом доступе, а во-вторых, авторы явно различают обычные изменения и изменения бинарных файлов (которые выглядят следующим образом: `Binary files x and y differ`), что для простоты задачи на данном этапе можно опустить. В качестве альтернативы предлагается рассмотреть несколько доступных моделей, предобученных с более общей целевой функцией, но на гораздо бóльшем количестве данных.

Стоит заметить, что в существующих работах по автоматической генерации сообщений к коммитам для обучения и оценки качества чаще всего используется набор данных из работы NMT [32] или его более отфильтрованная версия из работы NNGen [43]. У этого набора данных есть несколько ограничений:

1. Набор данных содержит всего около 30 тысяч примеров.

   В области обработки естественного языка для обучения моделей архитектуры Transformer обычно используется гораздо большее количество данных: например, популярные датасеты для нейронного машинного перевода WMT 2014 English-German и WMT 2014 English-French [26] состоят из 4.5 миллионов и 36 миллионов примеров соответственно.

2. Ограничения по максимальному количеству токенов и в изменениях, и в сообщениях в наборе данных достаточно небольшие: 100 и 30 токенов соответственно.

   При реальном использовании систем контроля версий в одном коммите может быть внесено много изменений, что приведет к длинному выводу команды `git diff`. Сама по себе архитектура Transformer способна обрабатывать и более длинные последовательности, типичные ограничения по количеству токенов для неё составляют 512 или 1024 токена [2]. Но обученная только на коротких последовательностях модель, скорее всего, не очень хорошо покажет себя при обработке более длинных.

3. Сообщения в наборе данных имеют единообразную грамматическую структуру.

   Одной из причин авторов работы NMT [32] для фильтрации сообщений к коммитам по грамматической структуре стало большое количество разных стилей письма в открытых источниках — при обучении на разнородных данных качество их метода могло ухудшиться.

   Альтернативой приведению всех сообщений к одной грамматической структуре может стать подход из плагина Grazie, позволяющий учитывать стиль письма пользователей.

Таким образом, можно выделить следующие направления для улучшения метода автодополнения сообщений к коммитам на основе архитектуры Transformer:

- сбор набора данных, не обладающего ограничениями уже существующего;
- использование истории сообщений каждого пользователя в качестве дополнительного контекста при автодополнении;
- использование для инициализации весов моделей, предобученных на больших наборах данных.

2. Базовый метод автодополнения на основе изменений

В данном разделе приведено описание метода автодополнения сообщений к коммитам, полученного прямой адаптацией лучшего подхода для автоматический генерации.

2.1. Описание метода

В качестве метода для автодополнения сообщения к коммитам на основе изменений была выбрана языковая модель архитектуры Transformer, так как она показывает лучшие результаты по автоматической генерации сообщений к коммитам и легко адаптируется под задачу автодополнения: обучение проводится идентичным способом, отличается только процесс оценки качества (см. Рис 4). В частности, для оценки качества систем автодополнения используются другие метрики (см. раздел 1.1.2)

Среди доступных открытых наборов данных для обучения был выбран наиболее отфильтрованный датасет из работы NNGen [43]. Более подробное описание этого набора данных приведено в разделе 1.2.2.

2.2. Инфраструктура для обучения и оценки качества

Среди фреймворков для работы с нейронными сетями наиболее популярны TensorFlow [59] и PyTorch [47]. В данной работе для реализации инфраструктуры для обучения и оценки качества нейронных сетей был выбран PyTorch как более удобный и простой для понимания.

Также использовалась библиотека PyTorch Lightning [23], позволяющая собрать всю основную логику модели в одном модуле и во многом автоматизировать её.

Для работы с моделями архитектуры Transformer использовалась библиотека HuggingFace's Transformers [60].

Репозиторий с инфраструктурой находится в открытом доступе[7].

2.3. Воспроизведение результатов автоматической генерации

В рамках данной работы было проведено сравнение качества автоматической генерации сообщений к коммитам с моделями из работы CoreGen [18].

Для обучения модели использовался графический процессор (GPU) NVIDIA Tesla T4, для оптимизации параметров использовался алгоритм AdamW на основе градиентного спуска [41]. Было проведено несколько экспериментов с разными гиперпараметрами, значения гиперпараметров, с которыми реализованная модель показала наилучшее качество, приведены в Таб. 4.

Информация о проведенных экспериментах доступна в Weights & Biases [8] проекте[8].

Количество слоев в кодировщике	4
Количество слоев в декодировщике	4
Размер пакета (batch size)	8
Коэффициент скорости обучения (learning rate)	0.0001*
Количество эпох	40

Таблица 4: Основные гиперпараметры обучения модели Transformer для задачи автоматической генерации.
* Коэффициент скорости обучения первые 4000 шагов линейно повышался до значения 0.0001, затем линейно понижался до конца обучения.

Результаты реализованной модели приведены в Таб. 5.

Лучше всего качество у модели CoreGen, которая была дополнительно предобучена с использованием специальной целевой функции.

[7]Открытый GitHub репозиторий: `https://github.com/JetBrains-Research/commit_message_generation` (дата обращения: 10.05.2021)

[8]Проект Weights & Biases: `https://wandb.ai/saridormi/commit_message_generation` (дата обращения: 10.05.2021)

Модель	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
CoreGen $_{\text{II}}$	18.74	30.65	18.06	28.86	15.18
CoreGen	**21.06**	**32.87**	**20.17**	**30.85**	**16.53**
(наша) Transformer 4+4	18.12	30.28	17.78	29.83	28.43

Таблица 5: Результаты автоматической генерации сообщений к коммитам на открытом датасете из работы NNGen [43] в сравнении с моделями из работы CoreGen [18].

Модель CoreGen $_{\text{II}}$, для которой предобучения не приводилось, аналогична обученному в рамках данной работы методу. Небольшие различия в значениях метрик могут быть вызваны тем, что гиперпараметры при обучении моделей совпадали не полностью. Для дальнейшего применения для задачи автодополнения они не так важны.

3. Предлагаемые улучшения

В данном разделе приведено описание предлагаемых улучшений к методу автодополнения сообщений к коммитам, полученному прямой адаптацией лучшего подхода для автоматический генерации.

3.1. Сбор данных с GitHub

Так как открытый набор данных имеет достаточно много ограничений, для последующих экспериментов предлагается собрать новый датасет, более реалистичный для рассматриваемого в данной работе сценария использования.

В качестве источника для сбора данных была выбрана платформа GitHub[9]: она содержит большое количество программных проектов с открытым исходным кодом и часто используется в различных исследованиях [19].

Выбор репозиториев для сбора данных

Для выбора открытых репозиториев из всех доступных на GitHub было установлено несколько критериев (см. Таб. 6):

- В качестве основного языка программирования был выбран Java, так как, во-первых, в дальнейшем метод планируется использовать в IDE IntelliJ IDEA, обычно применяемой для разработки на Java/Koltin, а во-вторых, в большинстве открытых датасетов также используются примеры на языке Java.

- Чтобы не допустить какого-либо нарушения авторских прав, для сбора данных были выбраны репозитории с подходящими открытыми лицензиями (Apache 2.0, MIT, BSD 3-Clause).

- Количество звезд — распространенный критерий оценки популярности при анализе открытых GitHub-репозиториев [21], который используется для фильтрации нерелевантных проектов.

[9]GitHub: `https://github.com/` (дата обращения: 10.05.2021)

- Критерии по количеству коммитов, контрибьюторов и дате создания репозиториев были установлены с целью оставить проекты с содержательной и разнообразной историей [21].

Основной язык программирования	Java
Лицензия	Apache 2.0, MIT, BSD 3-Clause
Количество звезд	≥ 50
Количество коммитов	≥ 1000
Количество контрибьюторов	≥ 10
Как давно создан	не менее 2 лет назад
Является ли форком	нет
Наличие не-ASCII символы в описании	нет

Таблица 6: Критерии для выбора репозиториев

Был собран список удовлетворяющих данным критериям репозиториев, для этого использовались инструмент GitHub Search [19] и официальный GitHub REST API. Затем, чтобы обработать случаи переименования репозиториев и не допустить повторения одного и того же репозитория несколько раз, из списка были отброшены совпадения по полному названию (full_name). Итоговое количество репозиториев составило 995.

Сбор и обработка данных

Для сбора данных использовалась библиотека PyDriller [55].

Из репозиториев из полученного списка были собраны данные обо всех коммитах, изменения в которых затрагивали `.java` файлы. Данные о каждом коммите включают в себя внесенные изменения, сообщение, имя и адрес электронной почты автора, дату и репозиторий.

Также собранные данные были предобработаны:

1. Для того, чтобы убрать выбросы по количеству измененных файлов и количеству токенов в сообщениях и в изменениях, были отброшены примеры со значениями этих признаков, выходящими за посчитанные по всему набору данных перцентили. Конкретные перцентили и их значения приведены в Таб. 7.

Признак	Перцентиль	Значение
Количество измененных файлов	$> 95\%$	> 32 файлов
Количество токенов в изменениях	$< 1\%$	< 73 токенов
Количество токенов в изменениях	$> 90\%$	$> 11\ 402$ токенов
Количество токенов в сообщении	$< 5\%$	< 3 токенов
Количество токенов в сообщении	$> 99\%$	> 130 токенов

Таблица 7: Значения перцентилей для нескольких признаков, по которым отбрасывались примеры

2. Так как в большинстве существующих моделей есть ограничения по количеству токенов, были отброшены примеры с количеством токенов в изменениях, большим 2048. Число было выбрано так, чтобы остались последовательности длиннее типичных для моделей архитектуры Transformer ограничений в 512 или 1024 токенов, но при этом занимаемая датасетом память сильно уменьшилась.

3. Из изменений и сообщений были удалены уникальные идентификаторы (номер issue, хэш коммита), адреса электронной почты и ссылки.

4. Набор данных был разделен на обучающую, валидационную и тестовую выборки следующим образом: дважды случайным образом выбрано по 10 репозиториев, примеры из них отнесены к валидационной и тестовой выборкам, все примеры из оставшихся репозиториев — к обучающей. Также из обучающей выборки были отброшены примеры от авторов, примеры от которых присутствуют в валидационной или в тестовой.

5. Чтобы не допустить совпадений между обучающей, валидационной и тестовой выборками, которые могут привести к смещению метрик качества моделей, с помощью инструмента SourcererCC [54] была проведена дедупликация по изменениям и по сообщениям (всего отброшено 169 947 клонов).

Сравнение с существующим открытым набором данных

В Таб. 8 приведена статистика по собранному датасету в сравнении с открытым датасетом из работы NNGen [43].

	Собранный датасет	Датасет NNGen
Дата сбора данных	март 2021	сентябрь 2016
Количество репозиториев	995	1 000
Количество примеров (всего)	1 264 851	27 144
Количество примеров (в обучающей выборке)	1 228 239	22 112
Количество примеров (в валидационной выборке)	24 605	2 511
Количество примеров (в тестовой выборке)	12 007	2 521
Максимальное количество токенов в изменениях*	2 048	121
Среднее количество токенов в изменениях*	665 ± 527	68 ± 22
Максимальное количество токенов в сообщениях*	130	30
Среднее количество токенов в сообщениях*	17 ± 19	7 ± 4

Таблица 8: Статистика по собранному набору данных в сравнении с открытым набором данных из работы NNGen [43]
* токенизация проводилась по пробелам

Данные собирались из практически одинакового количества репозиториев, но стоит заметить, что и время сбора данных, и критерии, по которым выбраны репозитории, отличаются: в открытом датасете учитывался только порядок в рейтинге открытых репозиториев на языке Java по количеству звезд, в данной работе используется несколько критериев, что позволяет более тщательно отобрать источники.

В собранном датасете установлены бóльшие ограничения по количеству токенов. В результате в собранном датасете гораздо больше примеров, а изменения и сообщения в нем в среднем длиннее. Также, в отличие от открытого датасета, в собранном есть информация о датах и авторах коммитов и присутствуют сообщения, не имеющие грамма-

тическую структуру глагол-прямое дополнение. В целом, собранный набор данных более реалистичен для нашего сценария использования.

3.2. Использование истории сообщений

Так как в собранном наборе данных не проводилось фильтрации по грамматической структуре сообщений, сообщения от разных авторов могут существенно отличаться. По внесенным в коммите изменениям и префиксу сообщения может быть сложно понять, какой стиль письма обычно предпочитает пользователь, поэтому предлагается адаптировать идею из плагина Grazie под архитектуру кодировщик-декодировщик: подавать историю сообщений от того же автора как дополнительный контекст декодировщику (см. Рис. 5).

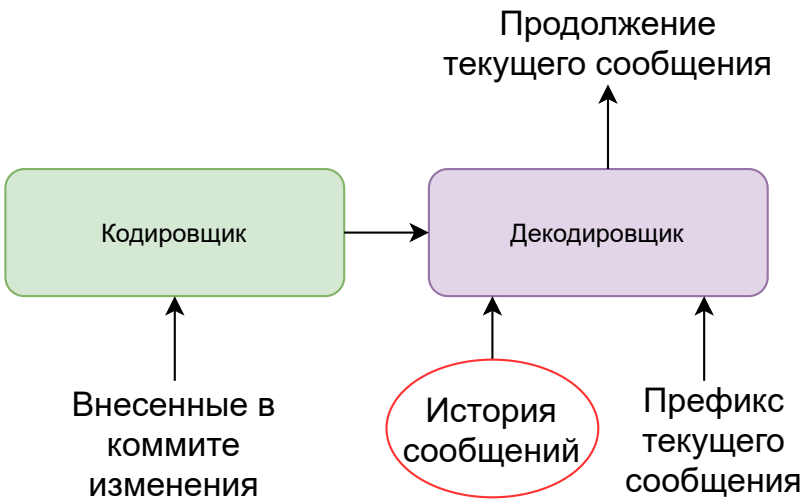


Рис. 5: Автодополнение сообщений к коммитам с использованием истории сообщений

3.3. Использование предобученных моделей

Предварительное обучение моделей на большом неразмеченном наборе данных позволяет улучшить качество решения многих задач, связанных как с тестами на естественном языке, так и с исходным кодом программ (см. раздел 1.3).

Специфичная целевая функция, как в работе CoreGen, вероятно, может лучше подойти для задачи автодополнения сообщений к ком-

митам, но обученных с ней моделей нет в открытом доступе, а подобное предобучение требует больших вычислительных ресурсов. В данной работе предлагается воспользоваться доступными предобученными моделями для инициализации весов кодировщика или декодировщика модели Transformer.

Как модель для инициализации весов кодировщика было решено выбрать CodeBERT [16]: из доступных на данный момент обученных на исходном коде моделей на основе BERT только у неё в датасете для предобучения присутствовали примеры на языке программирования Java. CodeBERT состоит из 12 слоев и может занимать достаточно много памяти, а также достаточно долго обрабатывать входные данные. Для будущего использования в плагине к IntelliJ IDEA необходимо иметь небольшую и быструю модель, поэтому в данной работе рассматривается также уменьшенная версия, полученная равномерным выбором 6 слоев из 12 слоев исходной модели [44]. В дальнейшем она упоминается как «smallCodeBERT».

В качестве вариантов для инициализации весов декодировщика было решено выбрать модель distilGPT-2, так как она небольшая и показывают хорошее качество генерации текстов на английском языке.

4. Эксперименты

В данном разделе приведено описание экспериментов по оценке качества предложенного подхода.

4.1. Методология экспериментов

Цели экспериментов

Для оценки предложенного в данной работе подхода были поставлены следующие вопросы.

1. Каково качество базового метода автодополнения сообщений к коммитам на основе изменений по сравнению с существующей в IntelliJ IDEA системой?

2. Как предложенные в данной работе улучшения влияют на качество базового метода?

3. Каково качество метода автодополнения сообщений к коммитам на основе изменений с лучшими из предложенных в данной работе улучшений по сравнению с существующей в IntelliJ IDEA системой?

Набор данных

Оценка качества проводится на тестовой выборке из собранного с GitHub набора данных, так как он более реалистичен для нашего сценария использования. Подробное описание этого набора данных приведено в разделе 3.1.

Метрики

Для оценки качества используются метрики Accuracy@1, Accuracy@5 и MRR@5, оценивающие качество предсказания следующего токена для каждой позиции и часто применяющиеся для оценки систем автодополнения. Их подробное описание приведено в разделе 1.1.2.

Параметры

Для обучения модели Transformer 4+4 используется 1 графический процессор (GPU) NVIDIA Tesla T4. Для остальных моделей, обучаемых на собранном датасете, используются 8 GPU NVIDIA Tesla V100. Для оптимизации параметров используется алгоритм AdamW на основе градиентного спуска [41]. Для всех экспериментов коэффициент скорости обучения (learning rate) первые 4000 шагов линейно повышается до значения 0.0001, затем линейно понижается до конца обучения — подобные подходы часто применяются при обучении моделей архитектуры Transformer [2].

4.2. Результаты экспериментов

Результаты проведенных экспериментов приведены в Таб. 9.

Модель	Accuracy@1, % (95% дов. интервал)	Accuracy@5, % (95% дов. интервал)	MRR@5, % (95% дов. интервал)
distilGPT-2 (инициализированная предобученными весами)	(21.8, 22.48)	(36.08, 36.79)	(27.13, 27.78)
distilGPT-2 (дообученная)	(30.48, 31.11)	(48.17, 48.84)	(37.15, 37.77)
Transformer 4+4 (базовый, обученный на открытом датасете)	(6.88, 7.22)	(13.86, 14.32)	(9.51, 9.87)
Transformer 4+4 (без истории)	(32.95, 33.61)	(52.83, 53.54)	(40.39, 41.01)
Transformer 4+4 (с историей)	**(37.91, 38.61)**	**(57.5, 58.21)**	**(45.34, 46.01)**
CodeBERT + distilGPT-2 (с историей)	(28.55, 29.16)	(45.81, 46.47)	(35.08, 35.69)
smallCodeBERT + distilGPT-2 (без истории)	(31.09, 31.72)	(50.79, 51.48)	(38.47, 39.08)
smallCodeBERT + distilGPT-2 (с историей)	(35.2, 35.87)	(54.85, 55.55)	(42.65, 43.3)

Таблица 9: Качество автодополнения сообщений к коммитам моделей в проведенных экспериментах. Все модели, кроме distilGPT-2 (инициализированная предобученными весами) и Transformer 4+4 (базовый, обученный на открытом датасете), обучены на собранном наборе данных.

1. **Каково качество базового метода автодополнения сообщений к коммитам на основе изменений по сравнению с существующей в IntelliJ IDEA системой?**

   Результаты автодополнения сообщений к коммитам базового метода на основе изменений и модели distilGPT-2 из плагина Grazie приведены в Таб. 9.

   Несмотря на то, что Transformer 4+4 использует информацию о внесенных в коммитах изменениях и, в отличие от инициализированной предобученными весами distilGPT-2, обучается на специфичных для задачи автодополнения сообщений к коммитам данных, значения метрик у неё гораздо ниже.

   Было выдвинуто предположение, что это связано не с плохой работой метода, а с сильным отличием открытого набора данных из работы NNGen [43], на котором была обучена модель Transformer 4+4, от собранного набора данных, на котором проводилось сравнение. Для его проверки дополнительно было проведено сравнение качества автодополнение на тестовой выборке из открытого набора данных из работы NNGen [43]. Результаты приведены в Таб. 10. Ввиду отсутствия в открытом наборе данных информации об авторах в качестве контекста для distilGPT-2 использовалась история сообщений из того же репозитория.

Модель	Accuracy@1, % (95% дов. интервал)	Accuracy@5, % (95% дов. интервал)	MRR@5, % (95% дов. интервал)
Transformer 4+4 (базовый)	**(35.77, 38.27)**	**(49.98, 52.49)**	**(41.18, 43.66)**
distilGPT-2 (инициализированная предобученными весами)	(15.48, 16.97)	(28.11, 29.86)	(20.08, 21.60)

Таблица 10: Качество автодополнения базового метода на основе изменений и модели distilGPT-2 из плагина Grazie *на тестовой выборке из открытого набора данных из работы NNGen* [43].

На открытом датасете качество автодополнения Transformer 4+4 выше, чем у distilGPT-2, так что наше предположение о влиянии структуры данных на результат работы модели подтвержается. Но, так как наиболее важным является качество на тестовой выборке из собранного набора данных, можно сделать вывод, что подход Grazie в данном случае выигрывает.

2. **Как предложенные в данной работе улучшения влияют на качество базового метода?**

   Результаты автодополнения сообщений к коммитам базового метода на основе изменений и нескольких вариантов улучшенных приведены в Таб. 9.

   Обученные на собранном наборе данных модели Transformer 4+4 показывают более высокое качество автодополнения, чем модель идентичной архитектуры, обученная на открытом.

   Использование в контексте декодировщика истории сообщений от того же автора во время обучения позволяет улучшить результаты для двух архитектур: Transformer 4+4 и CodeBERT (6 слоев) + distilGPT-2.

   В то же время эксперименты с инициализацией весов различными предобученными моделями в целом показали качество ниже, чем при обучении модели Transformer 4+4 с нуля. Это может быть связано с тем, что модель CodeBERT обучалась на программном коде, а в данной задаче на вход поступают внесенные в коммитах изменения, формат которых всё же отличается.

3. **Каково качество метода автодополнения сообщений к коммитам на основе изменений с лучшими из предложенных в данной работе улучшений по сравнению с существующей в IntelliJ IDEA системой?**

   Результаты всех экспериментов по автодополнению сообщений к коммитам приведены в Таб. 9. Лучшие результаты показала мо-

дель Transformer 4+4, обученная на собранном наборе данных с использованием истории сообщений.

Чтобы проверить предположение, что изменения — действительно существенная информация для автодополнения сообщений к коммитам, дополнительно было проведено дообучение distilGPT-2 на собранном наборе данных. Дообучение на собранном наборе данных позволяет повысить качество автодополнения сообщений к коммитам модели distilGPT-2, но качество основанной на изменениях модели выше.

Заключение

В рамках данной работы были получены следующие результаты.

1. Проведен обзор предметной области. Описана существующая система автодополнения для IntelliJ IDEA, распространенные подходы к автодополнению кода и текстов на естественном языке (ранжирование, языковое моделирование). Рассмотрены методы (kNN, encoder-decoder RNN, Transformer) и открытые наборы данных (из работ NMT, NNGen) для автоматической генерации сообщений к коммитам.

2. Воспроизведены результаты метода для автоматической генерации сообщений к коммитам на основе архитектуры Transformer. Реализована инфраструктура для обучения и оценки качества моделей для задачи автодополнения сообщений к коммитам (использовался язык программирования Python и фреймворк PyTorch), код находится в открытом доступе[10].

3. Создан датасет для экспериментов: с помощью библиотеки PyDriller собрано 1.3 миллиона коммитов из 995 GitHub-репозиториев на языке Java. Датасет отличается от существующих наличием информации об авторах коммитов, также он содержит в **46** раз больше примеров, а среднее количество токенов в изменениях и сообщениях в нем больше в **9** раз и в **2** раза соответственно[11].

4. Проведены эксперименты для оценки качества предложенного в данной работе метода. По результатам экспериментов предложенный метод на основе архитектуры Transformer достигает значения метрики MRR@5 **45.68**, для существующей в IntelliJ IDEA системы автодополнения сообщений к коммитам значение составляет **27.46**. Улучшение статистически значимо.

[10]Открытый GitHub-репозиторий: `https://github.com/JetBrains-Research/commit_message_generation` (дата обращения: 10.05.2021)

[11]Датасет доступен на Google Диске: `ссылка` (дата обращения: 10.05.2021)

Список литературы

[1] ALBERT: A Lite BERT for Self-supervised Learning of Language Representations / Zhenzhong Lan, Mingda Chen, Sebastian Goodman et al. — 2020. — 1909.11942.

[2] Attention is All You Need / Ashish Vaswani, Noam Shazeer, Niki Parmar et al. NIPS'17. — Red Hook, NY, USA : Curran Associates Inc., 2017. — P. 6000–6010. — ISBN: 9781510860964.

[3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). — Minneapolis, Minnesota : Association for Computational Linguistics, 2019. — Jun. — P. 4171–4186. — Access mode: `https://www.aclweb.org/anthology/N19-1423`.

[4] BLEU: a Method for Automatic Evaluation of Machine Translation / Kishore Papineni, Salim Roukos, Todd Ward, Wei Jing Zhu. — 2002. — 10.

[5] Bahdanau Dzmitry, Cho Kyunghyun, Bengio Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. — 2015. — Vol. abs/1409.0473.

[6] Banerjee Satanjeev, Lavie Alon. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. — Ann Arbor, Michigan : Association for Computational Linguistics, 2005. — Jun. — P. 65–72. — Access mode: `https://www.aclweb.org/anthology/W05-0909`.

[7] Bar-Yossef Ziv, Kraus Naama. Context-sensitive query auto-completion // Proceedings of the 20th International Conference on World Wide Web (WWW). — 2011. — P. 107–116.

[8] Biewald Lukas. Experiment Tracking with Weights and Biases. — 2020. — Software available from wandb.com. Access mode: https://www.wandb.com/.

[9] Bruch Marcel, Monperrus Martin, Mezini Mira. Learning from Examples to Improve Code Completion Systems // Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. — ESEC/FSE '09. — New York, NY, USA : Association for Computing Machinery, 2009. — P. 213–222. — Access mode: https://doi.org/10.1145/1595696.1595728.

[10] Buse Raymond P.L., Weimer Westley R. Automatically Documenting Program Changes. ASE '10. — New York, NY, USA : Association for Computing Machinery, 2010. — P. 33–42. — ISBN: 9781450301169. — Access mode: https://doi.org/10.1145/1858996.1859005.

[11] CC2Vec / Thong Hoang, Hong Jin Kang, David Lo, Julia Lawall // Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. — 2020. — Jun. — Access mode: http://dx.doi.org/10.1145/3377811.3380361.

[12] Cai Fei, Liang Shangsong, de Rijke Maarten. Time-Sensitive Personalized Query Auto-Completion // Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. — CIKM '14. — New York, NY, USA : Association for Computing Machinery, 2014. — P. 1599–1608. — Access mode: https://doi.org/10.1145/2661829.2661921.

[13] ChangeScribe: A Tool for Automatically Generating Commit Messages / M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, D. Poshyvanyk. — 2015. — Vol. 2. — P. 709–712.

[14] Code Completion with Neural Attention and Pointer Networks / Jian Li, Yue Wang, Michael R. Lyu, Irwin King // Proceedings of the Twenty-Seventh International Joint Conference on Artificial

Intelligence. — 2018. — Jul. — Access mode: http://dx.doi.org/10.24963/ijcai.2018/578.

[15] Kim Seohyun, Zhao Jinman, Tian Yuchi, Chandra Satish. Code Prediction by Feeding Trees to Transformers. — 2021. — 2003.13848.

[16] CodeBERT: A Pre-Trained Model for Programming and Natural Languages / Zhangyin Feng, Daya Guo, Duyu Tang et al. // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings. — 2020. — P. 1536–1547.

[17] Elnaggar Ahmed, Ding Wei, Jones Llion et al. CodeTrans: Towards Cracking the Language of Silicone's Code Through Self-Supervised Deep Learning and High Performance Computing. — 2021. — 2104.02443.

[18] Nie Lun Yiu, Gao Cuiyun, Zhong Zhicong et al. CoreGen: Contextualized Code Representation Learning for Commit Message Generation. — 2021. — 2007.06934.

[19] Dabic Ozren, Aghajani Emad, Bavota Gabriele. Sampling Projects in GitHub for MSR Studies. — 2021.

[20] Sanh Victor, Debut Lysandre, Chaumond Julien, Wolf Thomas. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. — 2020. — 1910.01108.

[21] Wen Feng-Cai, Nagy Csaba, Lanza Michele, Bavota Gabriele. An Empirical Study of Quick Remedy Commits. — 2020. — 09.

[22] Exploring Software Naturalness through Neural Language Models / Luca Buratti, Saurabh Pujar, Mihaela Bornea et al. // arXiv e-prints. — 2020. — P. arXiv–2006.

[23] Falcon et al. William. PyTorch Lightning // GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning. — 2019. — Vol. 3.

[24] Svyatkovskiy Alexey, Lee Sebastian, Hadjitofi Anna et al. Fast and Memory-Efficient Neural Code Completion. — 2021. — 2004.13651.

[25] Hard Andrew, Rao Kanishka, Mathews Rajiv et al. Federated Learning for Mobile Keyboard Prediction. — 2019. — 1811.03604.

[26] Findings of the 2014 Workshop on Statistical Machine Translation / Ondřej Bojar, Christian Buck, Christian Federmann et al. // Proceedings of the Ninth Workshop on Statistical Machine Translation. — Baltimore, Maryland, USA : Association for Computational Linguistics, 2014. — Jun. — P. 12–58. — Access mode: https://www.aclweb.org/anthology/W14-3302.

[27] Generating Commit Messages from Diffs using Pointer-Generator Network / Qin Liu, Zihe Liu, Hongming Zhu et al. — 2019. — 05. — P. 299–309.

[28] Gmail Smart Compose: Real-Time Assisted Writing / Andrew Dai, Benjamin Lee, Gagan Bansal et al. — 2019. — Access mode: https://arxiv.org/pdf/1906.00080.pdf.

[29] Hochreiter Sepp, Schmidhuber Jürgen. Long Short-term Memory // Neural computation. — 1997. — 12. — Vol. 9. — P. 1735–80.

[30] Improving Language Understanding by Generative Pre-Training / Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever. — 2018.

[31] Svyatkovskiy Alexey, Deng Shao Kun, Fu Shengyu, Sundaresan Neel. IntelliCode Compose: Code Generation Using Transformer. — 2020. — 2005.08025.

[32] Jiang Siyuan, Armaly Ameer, McMillan Collin. Automatically Generating Commit Messages from Diffs Using Neural Machine Translation. ASE 2017. — Urbana-Champaign, IL, USA : IEEE Press, 2017. — P. 135–146. — ISBN: 9781538626849.

[33] Jing Kun, Xu Jungang. A Survey on Neural Network Language Models. — 2019. — 1906.03591.

[34] Brown Tom B., Mann Benjamin, Ryder Nick et al. Language Models are Few-Shot Learners. — 2020. — 2005.14165.

[35] Language models are unsupervised multitask learners / Alec Radford, Jeffrey Wu, Rewon Child et al. // OpenAI blog. — 2019. — Vol. 1, no. 8. — P. 9.

[36] Cho Kyunghyun, van Merrienboer Bart, Gulcehre Caglar et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. — 2014. — 1406.1078.

[37] Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation / Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre et al. // CoRR. — 2014. — Vol. abs/1406.1078. — 1406.1078.

[38] Learning and Evaluating Contextual Embedding of Source Code / Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, Kensen Shi // arXiv e-prints. — 2019. — P. arXiv–2001.

[39] Jiang Jyun-Yu, Ke Yen-Yu, Chien Pao-Yu, Cheng Pu-Jen. Learning user reformulation behavior for query auto-completion. — 2014. — 07.

[40] Lin Chin-Yew. ROUGE: A Package for Automatic Evaluation of Summaries. — Barcelona, Spain : Association for Computational Linguistics, 2004. — Jul. — P. 74–81. — Access mode: `https://www.aclweb.org/anthology/W04-1013`.

[41] Loshchilov Ilya, Hutter Frank. Decoupled Weight Decay Regularization. — 2019. — 1711.05101.

[42] Marasoiu Mariana, Church L., Blackwell A. An empirical investigation of code completion usage by professional software developers // PPIG. — 2015.

[43] Neural-Machine-Translation-Based Commit Message Generation: How Far Are We? / Zhongxin Liu, Xin Xia, Ahmed E. Hassan et al. ASE 2018. — New York, NY, USA : Association for Computing Machinery, 2018. — P. 373–384. — ISBN: 9781450359375. — Access mode: https://doi.org/10.1145/3238147.3238190.

[44] Sajjad Hassan, Dalvi Fahim, Durrani Nadir, Nakov Preslav. On the Effect of Dropping Layers of Pre-trained Transformer Models. — 2021. — 2004.03844.

[45] Park Dae Hoon, Chiba Rikio. A Neural Language Model for Query Auto-Completion // Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. — SIGIR '17. — New York, NY, USA : Association for Computing Machinery, 2017. — P. 1189–1192. — Access mode: https://doi.org/10.1145/3077136.3080758.

[46] Personalized Query Auto-Completion for Large-Scale POI Search at Baidu Maps / Ying Li, Jizhou Huang, Miao Fan et al. // ACM Trans. Asian Low-Resour. Lang. Inf. Process. — 2020. — Jun. — Vol. 19, no. 5. — Access mode: https://doi.org/10.1145/3394137.

[47] PyTorch: An Imperative Style, High-Performance Deep Learning Library / Adam Paszke, Sam Gross, Francisco Massa et al. ; Ed. by H. Wallach, H. Larochelle, A. Beygelzimer et al. — Curran Associates, Inc., 2019. — P. 8024–8035.

[48] Pythia: AI-Assisted Code Completion System / Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, Neel Sundaresan // Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. — KDD '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 2727–2735. — Access mode: https://doi.org/10.1145/3292500.3330699.

[49] Realtime Query Completion via Deep Language Models. / Po-Wei Wang, Huan Zhang, Vijai Mohan et al. // eCOM@SIGIR / Ed.

by Jon Degenhardt, Giuseppe Di Fabbrizio, Surya Kallumadi et al. — Vol. 2319 of CEUR Workshop Proceedings. — CEUR-WS.org, 2018.

[50] Recurrent neural network based language model / Tomas Mikolov, Martin Karafiát, Lukas Burget et al. — Vol. 2. — 2010. — 01. — P. 1045–1048.

[51] RoBERTa: A Robustly Optimized BERT Pretraining Approach / Y. Liu, Myle Ott, Naman Goyal et al. // ArXiv. — 2019. — Vol. abs/1907.11692.

[52] Rothe Sascha, Narayan Shashi, Severyn Aliaksei. Leveraging Pre-trained Checkpoints for Sequence Generation Tasks // Transactions of the Association for Computational Linguistics. — 2020. — 06. — Vol. 8. — P. 264–280.

[53] See Abigail, Liu Peter J., Manning Christopher D. Get To The Point: Summarization with Pointer-Generator Networks. — 2017. — 1704.04368.

[54] SourcererCC: Scaling Code Clone Detection to Big-Code / Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko et al. // 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). — 2016. — P. 1157–1168.

[55] Spadini Davide, Aniche Maurício, Bacchelli Alberto. PyDriller: Python framework for mining software repositories // Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018. — New York, New York, USA : ACM Press, 2018. — P. 908–911. — Access mode: `http://dl.acm.org/citation.cfm?doid=3236024.3264598`.

[56] StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding / Wei Wang, Bin Bi, Ming Yan et al. — 2019. — 1908.04577.

[57] A Study of Visual Studio Usage in Practice / S. Amann, S. Proksch, S. Nadi, M. Mezini // 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). — Vol. 1. — 2016. — P. 124–134.

[58] Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks / Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper et al. // arXiv preprint arXiv:2102.02017. — 2021.

[59] TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems / Martín Abadi, Ashish Agarwal, Paul Barham et al. — 2015. — Software available from tensorflow.org. Access mode: `https://www.tensorflow.org/`.

[60] Transformers: State-of-the-Art Natural Language Processing / Thomas Wolf, Lysandre Debut, Victor Sanh et al. — Online : Association for Computational Linguistics, 2020. — Oct. — P. 38–45. — Access mode: `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[61] Unified Pre-training for Program Understanding and Generation / Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, Kai-Wei Chang // arXiv preprint arXiv:2103.06333. — 2021.