Международная конференция IEEE по анализу, эволюции и реинжинирингу программного обеспечения 2021 года (SANER)

# Нейронная система ответов на вопросы для основных вопросов о подпрограммах

Аакаш Бансал∗, Закари Эберхарт∗, Лингфэй Ву† и Коллин Макмиллан∗ ∗депт компьютерных наук, Университет Нотр-Дам, Нотр-Дам, Штат Индиана, США

{abansal1, zeberhar, cmc} @nd.edu †IBM Research, Йорктаун Хайтс, Нью-Йорк, США {wuli}@us.ibm.edu

Система ответов на вопросы (QA) — это тип разговорного ИИ, который генерирует ответы на естественном языке на вопросы, задаваемые пользователями-людьми. Системы контроля качества часто составляют основу интерактивных диалоговых систем и широко изучаются для широкого спектра задач, начиная от рекомендаций ресторанов до медицинской диагностики. В последние годы был достигнут значительный прогресс, особенно благодаря использованию нейронных архитектур кодировщиков, обученных вводу больших данных. В этой статье мы предпринимаем первые шаги по внедрению современных нейронных технологий контроля качества в приложения программной инженерии путем разработки контекстно-ориентированной системы контроля качества для основных вопросов о подпрограммах. Мы курируем обучающий набор данных из 10,9 миллионов кортежей вопросов/контекст/ответов на основе правил, которые мы извлекаем из недавних эмпирических исследований. Затем мы обучаем пользовательскую нейронную модель контроля качества с помощью этого набора данных и оцениваем модель в исследовании с профессиональными программистами. Мы демонстрируем сильные и слабые стороны системы и закладываем основу для ее использования в возможных диалоговых системах для разработки программного обеспечения.

*Термины* индекса — нейронные сети, диалог вопросов и ответов

## I. ВВЕДЕНИЕ

Система ответов на вопросы (QA) — это тип разговорного ИИ, который фокусируется на генерации ответов на естественном языке на вопросы, задаваемые пользователями-людьми. QA определяется как однооборотный диалог, в котором в разговоре участвуют только два участника (человек и машина), и каждый участник говорит только один ход (человек задает вопрос, на который отвечает машина). На практике полный агент разговорной машины будет обсуждать несколько тем в течение произвольного количества ходов, обнаруживать, когда был задан вопрос, и использовать систему контроля качества для генерации ответа на вопрос. Таким образом, системы контроля качества являются ключевыми компонентами, необходимыми для создания полезных диалоговых агентов.

|  |
| --- |
| 978-1-7281-9630-5/21/$31.00 ©2021 IEEE  DOI 10.1109/SANER50967.2021.00015 |

В общем, системы контроля качества генерируют ответ с учетом контекста, о котором квестИон спрашивается. Для пример, Инь *и к.* [1] описывать а подход тот Анализирует a знание основа из данные около знаменитый народ Кому порождать Английский Ответы около Рождения, политический Офисы держать, Награды полученный, и так далее. Малиновский *и к.* [2] присутствующий a system тот Ответы Вопросы около Изображения, такой как который Объектов ар красный или зеленый в тем образ. Уэстон *и к.* [3] обеспечивать a набор данных из Двадцать Задачи для тренировка Контроль качества Системы (тем так называемый Свинья Задачи) определение дальности От позиционный рассуждение Кому путь обнаружение, для который тем контекст есть a знание основа из данные около Объектов и как они рассказывать Кому каждый другой (Например.. Контекст: 1. Лилия Передал тем младенец Кому Филип. 2.

2021

IEEE Международная конференция по анализу, эволюции и реинжинирингу программного обеспечения (SANER) | 978-1-7281-9630-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/SANER

50967.2021.00015

Филипп вышел на улицу. Вопрос: Где ребенок? Ответ: Снаружи с Филиппом.).

Как показывают приведенные выше примеры и как описано в нескольких обзорных работах [4]–[6], научная литература из области обработки естественного языка (НЛП) и ИИ изобилует системами контроля качества, предназначенными для ответа на вопросы о контексте. Общая структура этих подходов довольно последовательна: собирается большой набор данных, включающий вопросы, ответы и связанные с ними контексты. Затем модель обучается и тестируется с использованием набора данных. Как правило, используется нейронная сеть энкодера-декодера, в которой модель учится связывать признаки в вопросах с признаками в контексте с помощью механизма внимания. Тем не менее, для моделирования контекста всегда требуется множество настроек для конкретной области (как правило, вопрос и ответ могут быть смоделированы с использованием функций языка, таких как рекуррентная нейронная сеть). Например, работа Малиновского, связывающая слова в вопросах с признаками на изображениях, использует типичную модель вопросов и ответов на основе RNN, но зависит от пользовательской модели извлечения этих признаков из изображений [2]. Короче говоря, ключевая трудность в реализации систем контроля качества сводится к: 1) получению правильного набора данных и 2) разработке подходящей доменной модели контекста. Как спроектировать такую систему для исходного кода – вопрос активного исследования.

В этой статье мы представляем систему контроля качества для ответа на «базовые» вопросы программиста о подпрограммах в программах (подпрограммы — это контекст, о котором задаются вопросы). «Базовый» вопрос — это вопрос о небольшой детали метода, такой как «Каковы параметры метода convertWavToMp3?» Мы используем основной вопрос, потому что: 1) Эберхарт *и др.*  [7] обнаружил, что программисты задают эти вопросы о методах Java во время реальных задач программирования и выделил пять типов этих вопросов, и 2) «базовый» Вопросы позже приведут к отличной трамплинной проблеме к более крупным проблемам. Мы создали шаблоны вопросов и ответов и парафразы на основе Eberhart *et al.* Основные типы вопросов для построения набора данных для 1,56 млн методов Java. Затем мы разработали систему контроля качества, основанную на модели нейронного кодировщика-декодера.

Мы оценивали нашу работу двумя способами. Во-первых, мы использовали автоматизированные метрики в большом наборе тестирования, состоящем из примерно 67 КБ методов Java, чтобы оценить, как наш подход будет обобщать. Во-вторых, мы провели эксперимент с 20 экспертами-людьми, чтобы определить, насколько хорошо наша модель реагирует на фактический вклад человека для подмножества из 100 случайно выбранных методов из набора тестов 67k. Мы исследуем доказательства того *, как* наша модель учится распознавать соответствующие факты в исходном коде и генерировать ответы на английском языке (в духе объяснимого ИИ [8], [9]).

II. PROBLEM, SIGNIFICANCE, SCOPE

Этот документ задуман как ступенька к полноценной системе контроля качества для инженеров-программистов. Современные технологии в области НЛП/ИИ предназначены для ответа на вопросы из заданного контекста, как описано выше. Цель этого документа состоит в том, чтобы продемонстрировать, как адаптировать эти технологии, чтобы информацию об исходном коде можно было извлечь из больших наборов данных, и эту информацию использовали для ответа на вопросы об исходном коде. Неразумно ожидать создания системы контроля качества, способной отвечать на произвольные вопросы о программировании в одной статье. Существующая рецензируемая литература даже не продемонстрировала, что контекстно-зависимые системы контроля качества могут учиться на исходном коде для интерпретации и ответа на вопросы естественного языка. Настоящий документ является доказательством концепции в этом направлении.

Мы понимаем, что программисты, вероятно, не будут использовать систему контроля качества только для основных информационных вопросов об исходном коде. В конце концов, возвращаемый тип, список параметров и т. Д. Функции легко доступен при чтении исходного кода или обобщении документации. Тем не менее, важно признать, что система контроля качества обычно не предназначена для использования сама по себе. Вместо этого система контроля качества для этих вопросов является ключевым компонентом в общей картине диалоговых систем ИИ для программистов. Робийяр с тринадцатью ведущими соавторами в области понимания программ ясно обосновывает это в статье, обобщающей результаты соответствующего семинара в 2017 году [10]: они «выступают за новое видение удовлетворения информационных потребностей разработчиков», которое они называют *документацией разработчиков по требованию*. Идея состоит в том, что мы, как область исследований, должны двигаться к машинным ответам на информационные потребности программистов, которые настроены на контекст программного обеспечения и отдельные вопросы программистов. Но чтобы добраться до этой точки, нам (исследовательскому сообществу) необходимо решить несколько небольших проблем, которые в настоящее время являются барьерами для дальнейшего прогресса. Этот аргумент отражает те, которые неоднократно высказывались в исследовательском сообществе ИИ в целом [11], [12], что меньшие проблемы должны быть решены и использованы в качестве клина против более крупных, для достижения долгосрочной цели значимого разговорного ИИ.

Система контроля качества для базовой программной информации о подпрограммах является одной из тех проблем клина в понимании программы. Успешная система не только ответит на узкую проблему, но и даст представление о том, как моделировать и извлекать функции из исходного кода, как интерпретировать информационные потребности программиста и как понимать словарь, который используют программисты, который отличается от общего использования слов. В нашей статье показано, как нейронная сеть может быть использована для распознавания типа вопроса, поиска ответа в коде и комментариях, а также ответа с помощью удобочитаемого грамматически правильного формата. Следовательно, эффективность этой системы измеряется не только точностью полученной информации, но и способностью распознавать ранее невидимые вопросы человека-пользователя и находить, а также возвращать правильно отформатированные ответы из контекста. В долгосрочной перспективе мы планируем включить эту работу в более широкую интерактивную диалоговую систему, помогающую программистам читать и понимать исходный код. (*Некоторые цитаты не соответствуют политике двойного слепого обзора.* )

III.B ACKGROUND & RELATED WORK

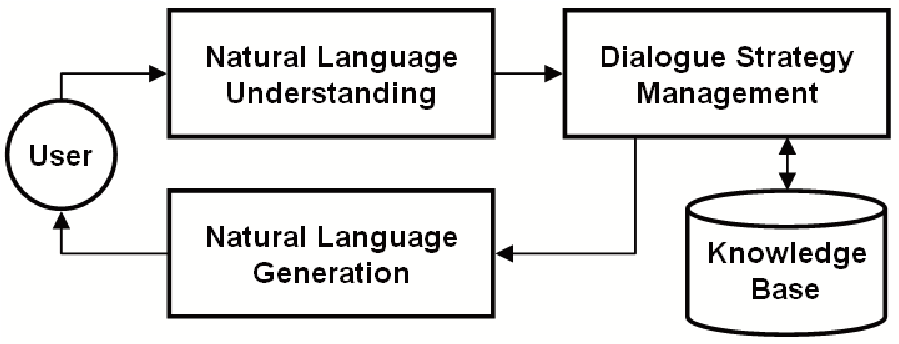
В этом разделе рассматриваются фоновые технологии и тесно связанная работа как в исследовательских центрах НЛП/ИИ, так и в области SE.

*А. Интерактивные диалоговые системы*

Анатомия интерактивной диалоговой системы аккуратно сформулирована в недавней книге Ризера и Лемона [13] и обобщена на рисунке 1 ниже. По сути, существует четыре компонента. Во-первых, создается база знаний для хранения информации, относящейся к разговору, такой как изображения, о которых задаются вопросы [2], [14], или карты о том, какие направления могут быть получены [15]–[17], или рестораны, которые могут быть рекомендованы [18]. Во-вторых, компонент понимания естественного языка отвечает за преобразование входящего текста во внутреннее представление того, что было сказано. Часто это начинается с маркировки текста диалоговым актом типа [19]–[23] (например, в виде вопроса, последующего заявления, положительного или отрицательного комментария). Но это также включает в себя извлечение соответствующей информации, необходимой для формирования ответа. Например, хочет ли пользователь знать о возвращаемом типе или списке параметров подпрограммы.

Третий компонент – управление стратегией диалога. Этот компонент решает, как реагировать, а также как извлекать информацию, необходимую для ответа. Он использует базу знаний, чтобы помочь принять это решение, и ищет в базе знаний информацию, относящуюся к ответу. Обратите внимание, что понятие «стратегия» относится к процессу принятия решений, которому следует машина, и отличается от естественного языка в разговоре [24]. Например, если представить комментарий о погоде, некоторые агенты ответят резюме прогнозируемой погоды, некоторые ответят предложением взять зонтик, в то время как третьи зададут вопрос о предпочтениях пользователя лету или осени. Но решение о том, как реагировать, не связано со словами, фактически используемыми для вынесения ответа.

В-четвертых, методы генерации естественного языка лежат вдоль спектра, одной из крайностей которого является шаблонная, основанная на правилах.



1: Стереотипная диалоговая система, описанная Ризером и Лемоном [13]. В этой статье база знаний состоит из исходного кода подпрограмм, в то время как компоненты понимания и генерации изучаются через нейронную сеть из создаваемого нами набора данных. Мы предопределили стратегию, основанную на экспериментальных результатах, представленных Eberhart *et al.*  [7] подход [25], в то время как другой крайностью является чисто управляемый данными (обычно основанный на глубоком обучении) подход [26]. Примером гибридной системы является система, в которой законсервированные ответы используются для обучения нейронной сети (что позволяет более гибкие комбинации ответов) или выбор на основе данных из набора ответов шаблона-кандидата. Какое-то время существовало убеждение, что понимание языка, стратегия и генерация могут быть объединены в единый модуль, основанный на глубоком обучении, но эта вера находится в сильном упадке для большинства приложений [11], [16], [24].

Системы контроля качества вписываются в эту анатомию интерактивных диалоговых систем двумя способами. Во-первых, диалоговая система в постоянном обсуждении с пользователем может включать в себя несколько подсистем для обработки различных ситуаций и передавать управление подсистеме контроля качества по мере необходимости. Во-вторых, сама система контроля качества, как правило, следует той же конструкции. Компонент стратегии, как правило, проще, чем большинство систем, из-за предположения, что один вопрос от пользователя вызовет один ответ. Тем не менее, системе контроля качества может потребоваться справиться с различными типами вопросов и извлечь информацию из различных артефактов - оба решения, которые попадают в категорию стратегии. На практике стратегия, как правило, кодируется в модель с помощью проектирования набора данных, а не ручной модификации модели.

Исследования диалоговых систем для разработки программного обеспечения, как правило, являются либо фундаментальными / генерацией и анализом наборов данных, либо реализациями экспериментальных диалоговых систем. Ключевые фундаментальные работы по анализу и анализу наборов данных включают Maalej *et al* [27], Eberhart *et al.*  [7] и некоторые другие [19], [28]– [33]. В недавнем исследовании обсуждаются диалоговые системы в SE [34], которые включают в себя несколько экспериментальных систем [35]–[38]. Эти системы связаны с данной работой в том смысле, что они являются прототипами диалоговых систем для задач SE, но не являются непосредственно сопоставимыми, поскольку они решают очень специфические задачи и основаны на изучении узкоспециализированных знаний предметной области. Хотя в одной ситуации человек может работать довольно хорошо, он может быть неподходящим для других; Например, OpenAPI Bot [39] способен отвечать на вопросы программистов о компонентах API, но требует хорошо документированных спецификаций API в определенном формате. Надежная система контроля качества, которая опирается на ИИ, может быть лучше способна обобщать более широкий спектр форматов программного обеспечения и документации, автоматически идентифицируя соответствующие функции в артефактах разработки. Эта специализация характерна для диалоговых систем во всех областях [13], поэтому способ оценки подхода заключается в сравнении альтернатив реализации, а не различных диалоговых систем [40], [41].

### B. Нейронный энкодер-декодер

Модель энкодера-декодера является современным состоянием систем контроля качества, как описано в нескольких обзорах [5], [6], [26], [42]. Чтобы выбрать одну очень недавнюю и связанную статью, которая иллюстрирует, как работают диалоговые системы, основанные на модели кодировщик-декодер, рассмотрим Lin *et al.*  [43]. В статье представлена модель памяти для дополнения кодировщика типичной конструкции кодировщика-декодера и сравнивается с другими моделями кодировщик-декодер по общедоступным наборам данных. Эта статья аналогична, но вместо модели, настроенной на общие разговоры, мы предлагаем модель кодировщика для этой конкретной проблемы SE для получения знаний в конкретной области.

Конструкция кодировщика-декодера была четко описана в нескольких статьях, и мы обсуждаем детали в нашем разделе подхода. В общем, дизайн включает в себя кодировщик, который получает входные данные на естественном языке от пользователя и базы знаний. Энкодер выводит векторное представление входных данных, обычно через рекуррентную нейронную сеть (RNN). Декодер получает пример желаемого вывода во время обучения и учится генерировать этот вывод из соответствующего входа. Во время вывода модель выводит по одному слову за раз и использует выходные данные, предсказанные «до сих пор», чтобы помочь предсказать следующее слово.

Конструкция энкодера-декодера взлетела в популярности после Богданау *и др.*  [44] Введен «внимательный » вариант, который позволяет векторному представлению декодера фокусироваться на участках представления кодировщика во время обучения, т.е. создать словарь слов на одном языке в декодере на другой язык в кодировщике. Конкретные проекты, такие как знаменитая модель seq2seq, мотивировали тысячи статей, что далеко за пределами того, что мы можем описать в этом разделе. Таким образом, мы направляем читателей на несколько опросов [45]–[48]. В литературе по разработке программного обеспечения конструкция кодировщика-декодера все чаще используется для таких задач, как автозавершение кода [49], обобщение кода [50], [51] и автоматизированное восстановление [52].

## IV. АППРОАХ

Наш подход согласуется с соответствующей работой, описанной в предыдущем разделе: общая архитектура основана на дизайне диалоговой системы на рисунке 1, а реализация основана на модели нейронного кодировщика-декодера. Ключевой новинкой в модели является представление базы знаний. Ключевой новинкой в общей архитектуре является создание нашего набора данных для обучения нейронной модели. Они устанавливают новую оценку, которая показывает, как эти модели работают в системе контроля качества для программного понимания функций. В долгосрочной перспективе мы планируем, что эта система контроля качества станет компонентом гораздо более крупного диалогового агента, который будет разработан в будущем документе. Ниже приведен обзор компонентов нашей диалоговой системы:

Напомним, что управление стратегией диалога включает в себя решения как о том, как реагировать, так и о том, как извлечь информацию, необходимую для принятия ответа. Для (1) мы создаем набор данных, который включает восемь типов вопросов, которые мы нашли в недавно выпущенных экспериментах моделирования с программистами. В то время как эти эксперименты были выполнены другим исследованием, мы завершили анализ восьми вопросов для этой статьи. Дизайн набора данных представляет собой наши ручные усилия по разработке стратегии, которой должна следовать система, но сама стратегия будет изучена во время обучения и закодирована в нейронной модели. Для (2) мы используем механизм внимания в нашей нейронной модели между входным вопросом и базой знаний, чтобы узнать, какие компоненты базы знаний относятся к конкретным вопросам. Подробная информация о дизайне нашего набора данных приведена в разделе IV-A. Подробная информация о механизме внимания и нейронной модели приведена в разделе IV-C.

База знаний База знаний состоит из исходного кода подпрограмм. Мы используем коллекцию методов Java, предоставленных Linstead *et al.*  [53] и далее обработано LeClair *et al.*  [54]. Всего база знаний включает 2,1 млн Java-методов из более чем 10 тыс. проектов. Эти методы Java фильтруются для нетокенизированных символов и подаются в виде необработанного исходного кода и пар вопросы-ответы в модель 3 input seq2seq.

Понимание естественного языка / Генерация Мы используем рекуррентные нейронные сети со словом, встраивающим векторные пространства для реализации кодировщика и декодера. Кодировщик — это, по сути, компонент, реализующий понимание естественного языка, а декодер реализует генерацию языка. Эта структура тесно связана с подавляющим большинством новейших систем контроля качества, основанных на данных (см. Раздел III-B).

### А. Подготовка набора данных

Мы подготовили набор данных для обучения нейронной модели, описанной в разделе 4.3. В этом разделе описывается, как мы структурируем наш набор данных, чтобы представить знания о том, как программисты задают вопросы и как отвечать. Обратите внимание, что мы явно не вписываем правила в наше управление стратегией диалога; этот набор данных неявно содержит те правила, из которых нейронная модель учится позже. Чтобы было ясно, мы не просто передаем в сеть все данные, собранные эмпирически, и ожидаем, что модель научится правильному поведению и оправдает общую позицию по проектированию набора данных: решения о создании набора данных *- это* решения, которые будут закодированы как управление стратегией диалога.

Мы строим правила генерации нашего набора данных на основе эмпирических данных, предоставленных нам в предварительном выпуске. Эберхарт *и др.*  [7] провел эксперимент, в ходе которого 30 программистов решали задачи программирования с помощью моделируемого интерактивного диалога agent (дизайн исследования «Волшебник страны Оз»). Затем авторы аннотировали каждый вопрос одним из двенадцати типов информационных потребностей API (эти потребности в информации ОБ API были определены в более ранней статье TSE Мааледжа и Робийяра [27]). Эберхарт *и др.*  обнаружили, что более 90% вопросов имели три информационные потребности: функциональность, шаблоны или основные. В долгосрочной перспективе диалоговый агент должен будет справиться со всем этим. Но масштаб этой проблемы слишком велик для одной статьи. В качестве ранней попытки решить проблему мы сосредотачиваемся на основных вопросах, которые, как правило, более самодостаточны, имеют конкретные одновитковые ответы и в целом на которые легче ответить.

Базовый вопрос — это тот, в котором программист запрашивает ключевую информацию о компонентах кода. «Компоненты» почти всегда были подпрограммами, а не классами и т.д. «Ключевая информация» включала в себя такие вещи, как возвращаемый тип, параметры функции или высокоуровневое описание (например, сводный комментарий от JavaDocs). Примерно 20% вопросов, заданных программистами в исследовании Eberhart *et al.*  [7] были основными вопросами.

Мы (независимо от анализа Eberhart *et al.* ) рассмотрели вопросы, которые были помечены как основные. Авторы создали восемь категорий основного вопроса. Процедура представляла собой открытый процесс кодирования, в котором авторы маркировали каждый вопрос конкретной информационной потребностью из подпрограммы. Авторы работали вместе, чтобы разрешить разногласия, а не работать независимо и вычислять метрику соглашения, чтобы повысить надежность данных.[[1]](#footnote-1)

В конце концов, у нас было восемь типов основных вопросов. Важным отличием является то, что шесть вопросов касались известных подпрограмм, т.е. программист уже знал правильный метод Например, что такое возвращаемый тип метода X. Три вопроса касались неизвестных подпрограмм, т.е. программист не знал правильного метода. Например, какой метод принимает int в качестве параметра и возвращает строку. Мы называем вопросы с известной подпрограммой «тип K» и вопросы с неизвестной подпрограммой «тип U». Вопросы типа K (известна подпрограмма):

1. Каков тип *метода* возврата?
2. Каковы параметры *метода*?
3. Дайте мне определение *метода*.
4. Что такое сигнатура *метода*.
5. Что делает *метод*?
6. Может *метод*, *краткое описание задачи*?

Вопросы типа U (подпрограмма неизвестна):

1. Как кратко описать *задачу*?
2. Какой метод использует тип параметра *P* и возвращает тип *R*?

Область применения нашей системы контроля качества включает только вопросы типа K. Вопросы типа K включают в себя вопрос, ответ и известный контекст, который соответствует тому, что модели QA в НЛП оснащены для решения (хотя эти модели не были адаптированы к исходному коду). Вопросы типа U включают в себя процесс поиска правильной подпрограммы, который будет включать в себя поиск кода и даже диалог между машиной и программистом, чтобы принять решение о правильной подпрограмме. Эти поисковые задачи являются обширными исследовательскими проблемами с автономными исследовательскими работами. Поэтому мы ограничиваемся проблемой ответа на основные вопросы об известных подпрограммах. Интеграция поиска кода, заземляющего диалога и т. Д. Является областью для нашей будущей работы, основанной на этом документе. *В. Создание наборов данных*

Следующим шагом является создание набора данных с изолированными типами вопросов. Для этого мы получили большой репозиторий методов Java, затем сгенерировали примеры вопросов и ответов для каждого типа вопросов, используя эвристику для автоматического извлечения информации из методов. Репозиторий методов Java представляет собой набор из 2,1 млн методов, уже отфильтрованных на наличие дубликатов и других ошибок и сопряженных с краткими описаниями, представленными на NAACL'19 [54]. Далее мы отфильтровали методы с дублирующимися и неописательными комментариями, оставив 1,5 млн методов.

Для генерации текста для вопросов (1-4) мы извлекли информацию из необработанного кода для каждого метода, например, возвращаемого типа. Для вопроса (5) мы использовали краткое описание, отформатированное грамматически. Для типа вопроса (6) мы использовали краткое описание и название метода в вопросе, и ответ был просто «да» или «нет». Только для типа вопроса (6) мы включили положительный и отрицательный пример для поддержания сбалансированного набора данных. Этот отрицательный пример состоял из случайного суммарного описания из другого метода (с другим именем, чтобы избежать выбора перегруженного имени метода) в том же проекте в паре с методом.

Чтобы ограничить размер словарного запаса, мы заменили некоторую информацию токенами, которые направляют выходной интерфейс на непосредственное копирование информации из контекста, а не научитесь прогнозировать информацию как часть модели. В частности, мы используем *маркер <funcode>* для замены определения метода, а не требуем, чтобы система копировала код метода по одному токену за раз. Это позволяет пользователю иметь тот же опыт, сокращая словарный запас, который модель должна выучить.

Последним шагом было перефразирование каждого вопроса и ответа. Вопросы типа K, перечисленные в Разделе 4.1, являются архетипическими примерами запросов пользователей, выражающих конкретные информационные потребности; на практике программист не будет использовать именно этот язык при формировании запроса. Чтобы учесть эту изменчивость, мы написали 15-25 перефразированных макетов каждого вопроса и случайным образом выбрали один из них при генерации вопросов и ответов. Вместо реального набора данных QA для программиста-помощника мы стремимся к тому, чтобы нейронная сеть училась на парафразах, достаточных для распознавания большинства, если не всех вариаций этих вопросов, задаваемых участниками. Все замены и парафразы доступны через наше онлайн-приложение (см. Раздел VII). Подводя итог, наша процедура выглядит следующим образом:

### для каждого из 1,56 млн методов M do для каждого типа вопроса T do

1. случайным образом выбрать шаблон парафраза для T
2. сгенерировать вопрос и ответ с помощью шаблона
3. препроцессный код M для использования в качестве context4. создать 3-кортеж: (вопрос, ответ, контекст), если *T == 6* , то
   1. случайным образом выбрать сводку различных методов
   2. создать 3-кортеж: (вопрос, "нет", контекст)

Результатом генерации нашего набора данных является набор из 10,88 миллиона 3-кортежей. Каждый 3 кортеж содержит вопрос, ответ и контекстный Java-метод. Для каждого из 1,56 млн. методов Java мы сгенерировали 7 вопросов и ответов типа K (один для типов вопросов 1-5, два для вопроса 6). Чтобы наилучшим образом улучшить воспроизводимость, мы сохранили разделение обучения / проверки / тестирования 90/5/5, которое обеспечивает отсутствие загрязнения между проектами или методами, предоставленными LeClair *et al.*  [54].

### С. Нейронная модель

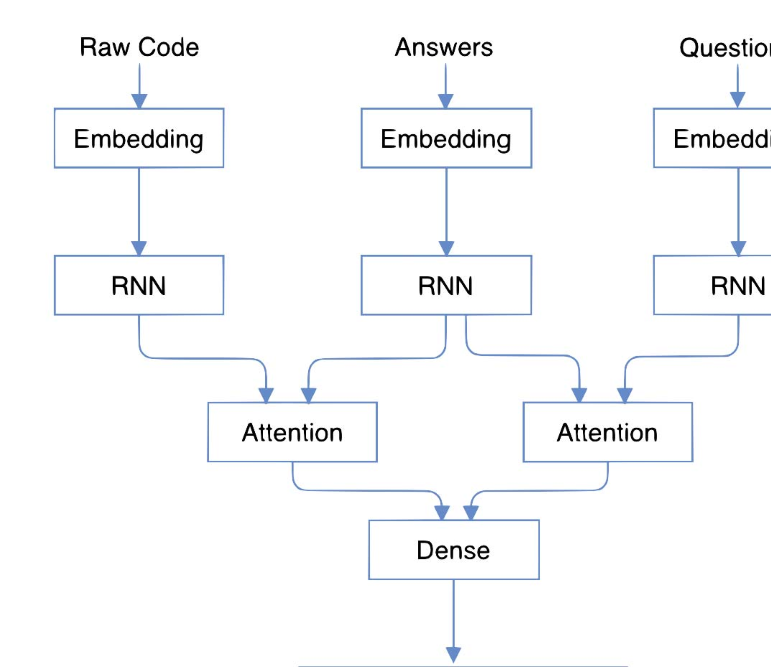
Почему нейронные сети? Нейронные модели обеспечивают гибкое понимание и генерацию естественного языка за несколько шагов, без необходимости ручного написания правил для извлечения информации из контекста. Традиционной альтернативой нейронной модели является простой подход, основанный на классификации входящих вопросов и правил извлечения информации. Тем не менее, эта, казалось бы, очевидная альтернатива не соответствует недавней работе из области исследований НЛП для контекстно-зависимых систем вопросов и ответов. Как Визе *и др.*  [56] отмечают, что недавние изменения в нейронных моделях привели к «впечатляющему повышению производительности по сравнению с более традиционными системами».

В отличие от этого, наша модель четко согласуется с соответствующей работой из области исследований НЛП по контекстно-зависимым системам вопросов и ответов (см. Раздел III-B). С точки зрения машинного обучения, одним из новых аспектов этой статьи является то, что мы показываем, как нейронная модель может изучать особенности в исходном коде, когда ей дают только этот код в качестве контекста и вопросы / ответы о контексте. Это важная новизна, по аналогии с Wiese *et al.*  [56], когда они показали , как нейронные модели вопросов и ответов могут учиться на биомедицинских текстовых данных по сравнению с другими высокоспецифичными областями, например. техническое сопровождение бесед [57] или даже религиозных текстов [58]. Дело в том, что адаптация предметной области считается важным вкладом и не просто применяет технологию X к данным Y. Нейронные сети добавляют гибкость и дисперсию как входным данным, которые они могут обрабатывать в таком исследовании, как наше, так и ответам, которые будут субъективно оцениваться программистами.

На высоком уровне наша нейронная модель похожа на контекстно-зависимые системы Q/A, описанные в соответствующей работе (см. Раздел III-B). Эти системы используют вопрос и контекст в качестве входных данных кодировщика и ответ в качестве входных данных декодера (для обучения). Модель учится предсказывать ответы по одному слову за раз. Наша модель следует той же базовой структуре; вопрос и ответ генерируются для каждой функции, а контекст является исходным кодом функции.

Наш подход основан на модели, выпущенной LeClair *et al.*  [50] на выставке ICSE 2019. Мы выбрали эту модель, потому что: 1) она была разработана для размещения исходного кода в качестве входных данных, а не только текста, 2) доступен полный пакет воспроизводимости и 3) он не полагается на предварительную обработку данных в семантические форматы или графики, как другие современные модели, такие как [59], [60], [61]. Эта модель была разработана для генерации описаний исходного кода на естественном языке (так называемое «обобщение исходного кода»). Входные данные для кодировщика модели представляли собой предварительно обработанный исходный код, сведенное абстрактное синтаксическое дерево. Исходным материалом для обучения декодеру послужил пример резюме.

Наши модификации, вкратце, заключаются в том, чтобы сделать кодировщик модели вводящим исходный код (не предварительно обработанным), добавить входные данные для запроса / вопроса пользователя в кодировщик и изменить обучающий ввод декодера на примеры ответов на вопросы. We использовал необработанный исходный код вместо предварительно обработанного исходного кода code



2: Нейронная модель и конвейер данных.

потому что нас интересует способность модели узнавать, где находятся функции кода, такие как возвращаемый тип, параметры и т. Д., В отличие от LeClair *et al.*  которые были больше заинтересованы в извлечении текстовых функций, таких как имена идентификаторов. Их шаги по предварительной обработке удалили информацию, которую мы сочли важной для того, чтобы помочь модели узнать особенности кода. Это имеет решающее значение для будущих приложений VA для извлечения информации из любого исходного кода, найденного в Интернете, без задержек или зависимостей. Подробности Мы предоставляем реализацию Keras нашей модели для максимальной ясности и воспроизводимости, следуя примеру LeClair *et al.*  [50], через наше онлайн-приложение (Section VII).

Соответствующий код модели находится в файле qamodel.py.

### Процедуры профессиональной подготовки

Наша процедура обучения основана на технике «принуждения учителя» [62]–[64], в которой модель получает только правильные примеры из обучающего набора и не подвергается собственным ошибкам. Напомним, что архитектура кодировщика-декодера обычно предсказывает выходные последовательности по одному элементу за раз. Например, задавая вопрос «что такое возвращаемый тип функции X?», модель будет генерировать ответ, предсказывая первое слово ответа:

[ вопрос ] + [ код ]

=> [ "the" ]

Затем он будет использовать предсказание первого слова в качестве нового входа в декодер, чтобы предсказать второе слово и так далее:

[ вопрос ] + [ код ] + [ "the" ]

=> [ "метод" ]

[ вопрос ] + [ код ] + [ "метод" ]

=> [ "возвращает" ]

[ вопрос ] + [ код ] + [ "метод возвращает" ]

=> [ "a" ]

[ вопрос ] + [ код ] + [ "метод возвращает a" ]

=> [ "длинный" ]

Тем не менее, именно так модель ведет себя во время вывода. Чтобы обучить модель, следуя процедуре принуждения учителя, мы предоставляем модели каждому примеру по одному слову за раз. Итак, в приведенном выше примере мы предоставим модели «the», за которым следует ссылочный вывод «method», затем «method» со ссылочным выводом «returns» и так далее. Если модель делает неправильное предсказание, мы используем обратное распространение, чтобы исправить модель, а затем подставляем правильный эталонный вывод на следующий шаг - модели не разрешается использовать свое собственное ошибочное предсказание в качестве следующего входа.

## V. ОЦЕНКА

Мы проводим эксперимент с пользователями-людьми для оценки нашей системы контроля качества. Обратите внимание, что наша конечная цель для этой системы контроля качества состоит в том, чтобы служить компонентом гораздо более крупного разговорного ИИ (см. Разделы II и III-A). Таким образом, наша экспериментальная установка представляет собой контролируемую среду, в которой мы тестируем конкретные входы и выходы, генерируемые пользователями-людьми. Мы *не* пытаемся оценить систему «в дикой природе», потому что система не предназначена для автономного использования, и потому что более крупной диалоговой системы ИИ еще не существует.

### A. Вопросы исследования

Цель нашего исследования состоит в том, чтобы определить степень, в которой наша система контроля качества способна ответить на восемь вопросов о подпрограммах, которые мы определили в разделе IV-A. Мы задаем следующие исследовательские вопросы (RQ) для достижения этой цели:

RQ1 Насколько хорошо наша система контроля качества работает с точки зрения актуальности, точности, полноты и краткости?

RQ2 Как производительность зависит от шести типов вопросов, для которых мы разработали систему?

RQ3 Какие функции в контексте наиболее важны для модели при ответе на вопрос?

Обоснование RQ1 заключается в том, что хорошие ответы любой системы контроля качества должны хорошо оцениваться по этим степеням: релевантность, точность, полнота и лаконичность. Релевантность, чтобы наблюдать, распознала ли модель вопрос и попыталась получить запрашиваемую информацию Например, если задается Q1 и возвращаемый тип int, но ответ, данный моделью, «Тип возврата двойной», это неточно, но актуально. Точность, поскольку независимо от каких-либо других факторов ответ не должен содержать ложной информации. Полнота, потому что ответы должны содержать *всю* информацию, необходимую для ответа на вопрос. Краткость, поскольку ответы должны содержать только необходимую необходимую информацию. Всем участникам были приведены примеры ситуаций, когда оценка этих баллов может привести к путанице, чтобы провести различие между релевантностью и точностью. Мы вывели эти четыре степени качества генерации текста из соответствующей литературы SE по генерации описания кода [65], [66]. Обоснование RQ2заключается в том, что система может хорошо работать по некоторым вопросам, но не по другим. В частности, он может хорошо работать при извлечении информации, такой как тип возврата подпрограммы, но бороться за другие вопросы, такие как возврат описания подпрограммы. Мы просим RQ3получить представление о поведении модели. Нейронные модели, как правило, очень эффективны для задач понимания и генерации текста, но печально известны тем, что производят ответы черного ящика, которые трудно понять. *В. Методология*

Наша методология ответа на RQ1 и RQ2заключается в проведении пользовательского исследования, в котором программисты задают вопросы системе контроля качества и оценивают ответы. Чтобы ограничить масштаб эксперимента, мы контролируем условия исследования, чтобы программисты задавали только вопросы, связанные с шестью вопросами типа K, выделенными в разделе IV-A. Мы набрали профессиональных программистов из США и Европы через онлайн-платформу вакансий (демография исследуемого населения приведена в следующем разделе). Мы также создали веб-интерфейс для программистов для связи с системой контроля качества. Снимок экрана этого интерфейса показан на рисунке 3. Затем интерфейс предлагает программистам оценить ответы по шкале от 0 до 4, начиная от «Полностью согласен», «Согласен», «Нейтральный», «Не согласен» или «Категорически не согласен» для Подсказки качества, показанные в таблице I. Наш дизайн исследования похож на  предыдущие эксперименты Шридхары *и др.*  [65] и Макберни *и др.*  [66]. Мы использовали аналогичную формулировку наших подсказок для изучения участников и те же четыре варианта. Единственное отличие, которое мы сделали, состояло в том, чтобы добавить еще один вариант для Neutral в случае, если модель возвращает ТАБЛИЦУ I: Подсказки качества для исследования пользователя, соответствующие критериям качества (релевантность, точность, полнота и лаконичность). Ответы для P1-P4: «Полностью согласен», «Согласен», «Нейтральный», «Не согласен» и «Категорически не согласен».

|  |  |
| --- | --- |
| П1 | Независимо от других факторов, ответ имеет отношение к моему вопросу, даже если содержащаяся в нем информация неточна. |
| П2 | Ответ точен, даже если он не имеет отношения к моему вопросу. |
| П3 | В ответе отсутствует важная информация, и это может помешать моему пониманию. |
| П4 | Ответ содержит много ненужной информации. |
| П5 | Есть ли у вас какие-либо общие замечания по поводу ответа? |

бессмысленный ответ (который может произойти для нашей нейронной модели, но был очень маловероятен в шаблонных системах генерации комментариев кода в тех предыдущих исследованиях). Мы добавили нейтральный вариант в качестве золотой середины, чтобы избежать принуждения участников к принятию решений о возможных бессмысленных ответах.

Другое сходство заключается в том, что мы оказываемся в той же ситуации, что и Шридхара *и др.*  [67] в своей статье ASE: не существует базового уровня для сравнения. Даже междоменные исследования, которые могут быть использованы в качестве базового уровня, используют реальные данные, поэтому они *не* сопоставимы с нашим исследованием с синтезированным набором данных. Насколько нам известно, ни одна система контроля качества не была разработана для ответа на эти конкретные вопросы в формате естественного языка. Для некоторых вопросов существуют различные инструменты. Например, вопрос (5) можно рассматривать как вопрос обобщения кода, в то время как на вопросы (1-4) можно ответить, просто прочитав саму подпрограмму. Тем не менее, помните, что мы не стремимся к оценке «в дикой природе» - нам нужно оценить входные и выходные данные модели in situ с помощью компонентов понимания естественного языка и генерации подхода. Поэтому мы следуем примеру этих предыдущих работ и фокусируемся на более глубоком анализе ответов по нескольким критериям качества, а не сравниваем метрики между конкурирующими подходами (поскольку они еще не существуют).

Обратите внимание, что мы не используем оценки BLEU или другие автоматизированные метрики. Исследование на людях жизненно важно по двум причинам. Во-первых, нам нужно оценить конкретные субъективные качества, а не сходство с основной истиной. Во-вторых, основная истина в нашем наборе данных (т.е. компонент ответа вопроса, контекст, кортежи ответов, см. Раздел IV-A) генерируется нами. Мы используем его в качестве обучающих данных, но было бы неуместно использовать его в качестве тестовых данных, поскольку он будет включать наши собственные предубеждения.

Процедура эксперимента В эксперименте мы дали каждому программисту «викторину» для заполнения с помощью системы контроля качества (см. Рисунок 3). На каждой странице викторины давалось имя определенного метода Java. Было показано только имя метода, а не тело метода. Для каждого метода случайным образом были выбраны три вопроса типа K (см. Раздел IV-A). Под именем метода были три подсказки, производные от выбранных вопросов типа K. Мы сформулировали подсказки как императивные операторы (например, «Предоставить возвращаемый тип этой функции»), чтобы избежать инициализации программистов определенным форматом вопроса. Мы проинструктировали программистов использовать свои собственные слова, чтобы запросить информацию из системы контроля качества. Мы попросили программистов не копировать вопросы, но разрешили им копировать ответы из системы контроля качества для викторины. Программист может задавать системе контроля качества столько запросов, сколько захочет. После ответа на вопрос о конкретном методе программисты были переведены на новую страницу, где их попросили оценить каждое из их взаимодействий с системой контроля качества для этого метода. Для каждого взаимодействия (состоящего из запроса пользователя и ответа системы контроля качества) мы попросили программистов ответить на пять подсказок качества, перечисленных в таблице I. Когда они были закончены, они могли нажать кнопку, чтобы вызвать следующий метод и новый набор подсказок.

Короче говоря, мы использовали формат викторины, чтобы побудить программистов задавать системе контроля качества определенные типы вопросов своими словами. Затем они оценили ответы, используя подсказки качества, чтобы мы могли определить, получили ли они правильную информацию в конце. Материалы, относящиеся к дизайну викторины, можно найти в нашем онлайн-приложении (Раздел VII).

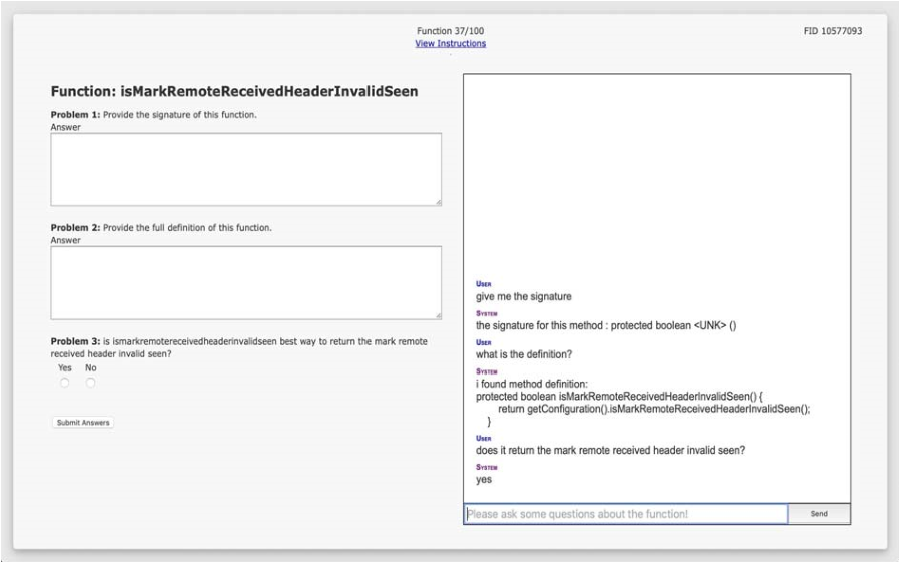
Для ясности в разделе экспериментальных результатов мы используем следующий словарь: 1) «вопрос», Q1-6, является одним из шести типов вопросов типа K, которые мы используем в нашем эксперименте и описаны в разделе IV-A, 2) «запрос» - это текст, вводимый пользователем в интерфейс эксперимента до клавиши возврата, нажатие клавиши возврата запускает интерфейс для отправки текста в модель прогнозирования и получения ответа, и 3) «подсказка качества», P1-5, является одним из запросов, которые мы делаем к пользователям, чтобы оценить ответ модели. Пользователи видят три вопроса на функцию. Они могут написать столько запросов, сколько пожелают. Затем они отвечают на пять качественных подсказок для ответа на каждый запрос.

### С. Участники

Мы набрали 20 участников для нашего эксперимента. Эти участники имели профессиональный опыт от трех до 15 лет. Программисты получили компенсацию по фиксированной ставке 60 долларов США в час, рыночной ставке в нашем регионе, независимо от скорости производительности. Каждый программист потратил в общей сложности 40 минут на столько страниц викторины, сколько смог.

### D. Предметные методы Java

В нашем эксперименте мы использовали в общей сложности 100 методов Java. Мы взяли эти методы из тестового сплита — модель не видела их во время обучения. Мы выбрали случайным образом, чтобы нет



3: Интерфейс, который программисты использовали для связи с системой контроля качества во время нашего эксперимента.

программист видел один и тот же метод более одного раза, и каждый метод был показан по крайней мере трем программистам. Но учитывая перипетии любого исследования на людях (усталость, разная скорость), не все методы заканчивались тремя рейтингами.

### Е. Угрозы действительности

Как и любая статья, этот эксперимент несет в себе угрозу обоснованности своих выводов. Одна угроза в основном внешняя, как определено Инь [68], а не внутренняя или случайная надежность из-за единообразного процесса и объективного анализа с использованием нейронной сети. Количество функций, которые мы могли бы попросить оценить любого человека, ограничено. Мы выбрали случайный выбор из большого, кураторского набора данных, гарантируя, что каждую функцию увидят более одного человека, но все же возможно, что другой выбор приведет к другому результату. Точно так же другой набор программистов может дать разные ответы. Мы попытались смягчить этот риск, опросив более 20 участников.

Одной из угроз для построения валидности [68] являются экспериментальные инструкции, объясненные администратором, которые мы пытаемся смягчить с помощью описания сценария одним администратором. Техническая надежность является риском за пределами дизайна исследования, мы смягчаем различные результаты от самой модели, обеспечивая последовательные случайные семена и экспериментальные условия (доступные через наше онлайн-приложение), однако случайные факторы производительности графического процессора или программного обеспечения могут привести к различным результатам.

## VI. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ

В этом разделе описываются результаты эксперимента: наши ответы на наши RQ и подтверждающие доказательства.

ТАБЛИЦА II: Последовательное взаимодействие одного участника. Мы заметили некорректный ответ с невыполненными словами типа «что»

|  |  |
| --- | --- |
| *запрос* пользователя | что он возвращает? |
| *выход* модели | Тип возвращаемого значения для этого метода — void |
| *запрос* пользователя | устанавливает ли он доменное имя? |
| *выход* модели | да |
| *запрос* пользователя | какова его цель? |
| *выход* модели | Нет |
| *запрос* пользователя | что он делает? |
| *выход* модели | Этот метод задает имя домена |

### A. RQ1: Общая производительность

В целом, модель имеет тенденцию генерировать разумные ответы. Таблица II показывает взаимодействие, модель правильно реагирует, когда слова находятся в словарном запасе, но для запроса вне словаря. На рисунке 5 представлен обзор. Рисунок представляет собой гистограмму всех пользовательских ответов на качественные подсказки из таблицы I. Напомним, что 1="Полностью согласен", 2="Согласен", 3="Нейтральный", 4="Не согласен" и 5="Категорически не согласен" на текст подсказки. Подсказки 1 и 2 сформулированы положительно (поэтому согласие лучше), в то время как подсказки

3 и 4 сформулированы отрицательно (поэтому разногласия лучше). Для

ТАБЛИЦА III: Статистика результативности участников эксперимента. Каждый участник работал по 40 минут. Мы задали три вопроса по каждому методу. Участники работали со своей скоростью, и им было разрешено задавать несколько запросов.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Значить | Мин | Макс |
| Методы, оцениваемые на одного участника | 19 | 7 | 38 |
| Запросы на участника | 70 | 37 | 117 |
| Запросы на метод | 3.8 | 2 | 10 |
| Запросы до "Правильного" ответа | 1.2 | 1 | 8 |

Например, для P1 об актуальности подавляющее большинство ответов получили оценку «Полностью согласен» или «Согласен». Что касается П3, то подавляющее большинство ответов указывало на несогласие, т.е. важная информация *не* отсутствовала. Также следует отметить, что лишь немногие ответы были оценены как нейтральные; в целом ответы были достаточно четкими, чтобы участники могли составить свое мнение. Большинство ответов, оцененных как нейтральные, были результатом модели тарабарщины.

Следует понимать два предостережения. Во-первых, разные участники работали с разной скоростью, поэтому некоторые участники представлены в данных больше, чем другие. В таблице III приведены количественные показатели этих различий. Большинство участников оценивали от 15 до 20 методов, но было несколько выбросов, что естественно в выборках человеческих популяций (средняя скорость 19 методов на 40 минут исследования составляет около 2 минут на метод, в то время как 38 - это скорость около 1 минуты / метод). Тем не менее, мы обнаружили, что количество запросов, необходимых для ответа на каждый вопрос, довольно стабильно, причем одного запроса обычно достаточно, а два или более запросов довольно редки. Время, необходимое каждому участнику, зависело от их чтения, набора текста и скорости оценки, больше, чем количество заданных запросов.

Во-вторых, ответы на каждую подсказку качества не зависят от других подсказок. Таким образом, возможно, что ответ получает хороший балл для P2 и плохой балл для P1(см. Раздел 5.1). Чтобы изучить это предостережение, мы вывели двоичную метрику, которую мы называем «правильностью», объединив оценкиP1 и P2. Ответ получает 1 тогда и только тогда, когда баллы P1 и P2равны [1,2] - т.е. ответ является «правильным» только в том случае, если участник полностью согласен или согласен с тем, что он является релевантным и точным. Мы обнаружили, что 79% ответов были «правильными» и что обычно требуется только один запрос, чтобы получить правильный ответ.

Мы обнаружили, что ключевым фактором в 21% неправильных ответов является размер словарного запаса. Как упоминалось в разделе IV-C, ограничения памяти графического процессора ограничивают как входной, так и выходной размер воскоба, несмотря на наши попытки расширить их, используя графические процессоры с 16 ГБ видеопамяти и низкими размерами обучающих пакетов. Этот предел повлиял на наши результаты. Подавляющее большинство ответов, которые были релевантными, но не точными, были с токенами UNK в ответе (т.е. модель могла найти ответ правильно, но токен UNK был важной частью ответа). Аналогичным образом, ответы, которые были точными, но не релевантными, почти всегда имели токены UNK в вопросе (т. Е. Участник написал запрос с невыполненными словами в нем) - эти токены UNK, вероятно, заставили модель неправильно понять вопрос и дать точный ответ, который, тем не менее, был нерелевантным, как в таблице II. Как и во всех исследованиях на людях, неожиданные или неправильные запросы, т.е. человеческая ошибка, также могут привести к результатам, поскольку мы позволили участникам относительно свободно взаимодействовать с моделями после коротких рекомендаций.

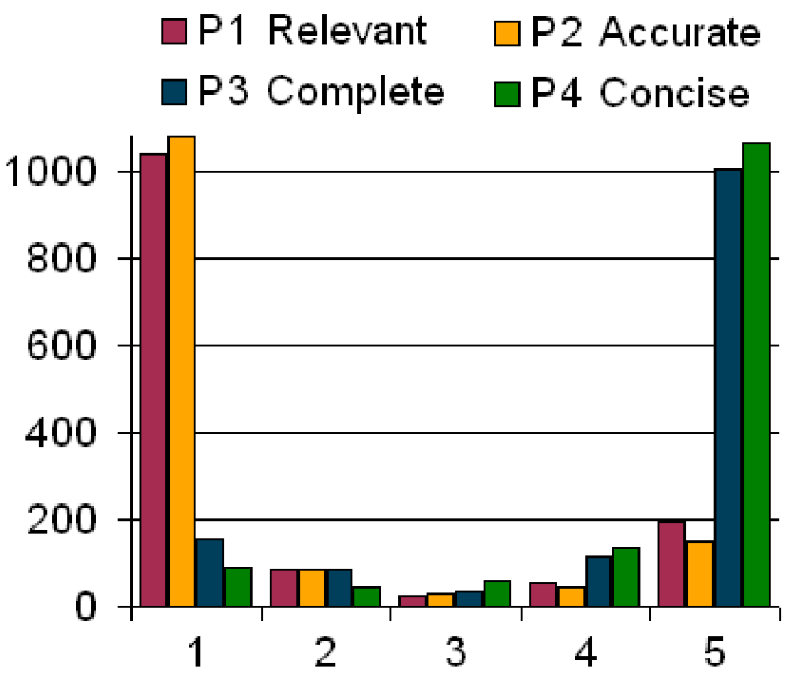
### B. RQ2: Различия между типами вопросов

|  |
| --- |
| а) Релевантность P1 b) Точность P2  с) P3 Полнота(d) P4Краткость  4: Коробочные графики ответов на подсказки качества (релевантность, точность, полнота и краткость) для каждого из типов вопросов из раздела IV-A. Для P1 и P2 более низкий балл лучше, однако для P3 и P4 более высокие баллы лучше, поскольку эти вопросы поставлены отрицательно. Производительность самая высокая для Q1 и Q4 и худшая для Q5: рейтинги релевантности и полноты, как правило, |

Мы наблюдаем небольшую степень вариации между типами вопросов в нашем эксперименте. Вспомните из раздела IV-A, что у нас есть множество шаблонов вопросов, которые мы получили из шести различных типов вопросов, соответствующих шести ключевым потребностям программистов. (В эксперименте мы ограничили участников этими информационными потребностями, но у нас не было ограничений на язык, который они могли использовать для постановки вопроса.) быть хуже для Q5, чем для других типов вопросов.

Напомним, что обоснование этого RQ заключается в том, что модель может лучше понимать некоторые информационные потребности, чем другие.

На рисунке 4 показаны квадратные графики ответов на каждую качественную подсказку, разделенные по каждому типу вопросов. Например, столбец Q2 на рисунке 4(a) показывает, что для вопроса 2 среднее значение всех ответов на запросы составляет около 2 (красная линия), межквартильный диапазон — от 1 до 2. Способ интерпретации этого заключается в том, что среди всех запросов, написанных для Q2, участники либо Сильно



5: Гистограммы ответов пользователя на подсказки качества в таблице I. Напомним, что P1 и P2 задаются положительным тоном (поэтому 1-2 балла лучше), в то время как P3 и P4 находятся в отрицательном тоне (поэтому 4-5 лучше). Участники, как правило, находили ответы модели хорошего качества.

Согласились или согласились с тем, что запрос актуален примерно в половине случаев. Обратите внимание, что выбросы исключены для удобочитаемости, но у нас был по крайней мере один экземпляр каждой оценки.

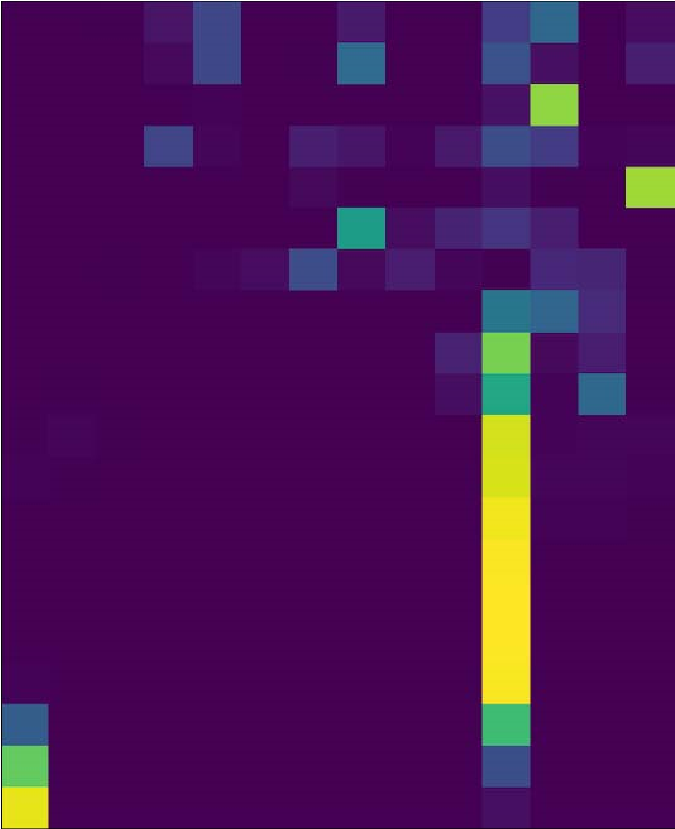
В целом, модель хорошо работает для Q1 и Q4. Для обоих ответов преобладают оптимальные оценки (1 для релевантности и полноты, 5 для точности и краткости). Этот результат подразумевает, что модель успешно учится распознавать, когда участники запрашивали эти информационные потребности, а также учится извлекать эту информацию из исходного кода и давать ответ на естественном языке. Q1 и Q4 соответствуют возвращаемому типу и сигнатуре метода.

Модель показала худшие результаты для Q2 и Q3, в которых модель определяет параметры метода и определение. Эти информационные потребности могут быть более трудными для изучения, потому что они больше различаются по размеру и словарному запасу. Возвращаемый тип (Q1) всегда можно найти в том же месте в начале подписи метода, обычно он состоит из одного слова, а словарный запас ограничен именами типов. Список параметров также всегда можно найти в одном и том же месте, но он различается по длине и включает имена идентификаторов, которые могут быть специфичными для этого метода.

Модель работает наименее хорошо на Q5, особенно с точки зрения актуальности и полноты. Однако этот результат можно ожидать, поскольку ожидается, что модель предоставит краткое описание поведения метода. Мы даем модели краткое описание в контексте, и модель действительно учится использовать это в своем ответе. Но размер, качество и словарный запас описания значительно различаются, так как источником является JavaDocs, что делает модель склонной к использованию неправильных слов.

Ответы на Q6 являются особым случаем, поскольку они всегда либо «да», либо «нет», что позволяет модели легко получить хорошую оценку с точки зрения, например, релевантности. Тем не менее, модель иногда ошибочна, что отражено в колонке Q6 рисунка 4(b).

|  |  |
| --- | --- |
| *запрос* пользователя | Дайте мне тип возврата |
| *выход* модели | Возвращаемым типом для этого метода является vertex |
| *контекст* | <st> возвращает следующую вершину многоугольника nl публичной вершины nextvertex ( vertex v ) { int ind = вершины .  индекс ( v ) ; возврат ( вершина ) ( ind  == -1 ? null : вершины . get ( ( ind  + 1 ) % вершин . размер () ) ); } <эт> |

*<*st*>* 1 значение return3 type4 for5 this6 способ 7 is8

9

⏐ 10

⏐ *прогнозирование*

11

*следующее слово*

12

13

14

15

16

17

18

19

20

12345678910 11 12 13 14

Пример 1: Участник запрашивает тип возвращаемого метода. Тепловая карта сети внимания показывает, как модель учитывает правильное положение «вершины» в контексте (позиция 11), чтобы предсказать последнее слово в выходных данных.

### С. RQ3: Влияние контекстных особенностей

Мы приводим доказательства влияния признаков в контексте на примере поведения модели. «Объяснимый ИИ» — спорная тема, с большим согласием в том, что это необходимо, но мало консенсуса по лучшим стратегиям — нейронные сети, в частности, имеют репутацию производящих результаты, которые трудно объяснить [8], [9]. Однако одним из источников доказательств является сеть внимания. Механизмы внимания в большинстве моделей энкодера-декодера отвечают за соединение частей входов декодера с частями входов энкодера. Часто внимание дает подсказки относительно того, почему модель принимает то или иное решение. Например, в NMT немецкому слову «hund» будет уделяться большое внимание английскому слову «dog», в то время как в компьютерном зрении слово «dog» может получить высокое внимание к области на изображении, где появляется собака.

В нашем подходе механизм внимания связывает выходные слова со словами во входной контекстной последовательности. Рассмотрим пример 1. Участник исследования пользователя пишет запрос с просьбой вернуть тайп. Тепловая карта показывает состояние сети внимания непосредственно перед предсказанием слова «вершина» (слой code\_attn из раздела IV-C, вспомните из раздела IV-D, что модель предсказывает выход по одному слову за раз). Этот пример типичен почти для всех запросов о возвращаемом типе: модель узнала, где найти возвращаемый тип в коде. Он не всегда находится в позиции 11, но модель научилась искать подпись, и где искать в подписи возвращаемый тип. Обратите внимание, что модель не учитывает более раннее использование слова vertex в описании метода, поскольку эта формулировка может измениться. Точно так же он не обращает внимания на слово vertex после фактического возврата в коде, поскольку это имя переменной, которое может не быть фактическим типом возвращаемого значения.

Мы включаем еще несколько примеров в наше онлайн-приложение, приведенное ниже. Поведение довольно последовательное: для запросов, например, о параметрах, модель обращает внимание на область параметров подписи и выводит соответствующую информацию.

## VII.C ВКЛЮЧЕНИЕ

Мы представили систему контроля качества для вопросов программистов о подпрограммах. Мы разработали нейронную модель, основанную на структуре кодировщика-декодера, которая может извлекать информацию о методах Java непосредственно из исходного кода. Наша система различает и отвечает на вопросы по шести различным информационным потребностям, полученным из недавней работы над диалоговыми системами для программистов. В эксперименте с 20 профессиональными программистами мы показываем, что наш подход способен достоверно ответить на эти типы вопросов.

На протяжении всей нашей статьи мы отмечаем, что эта система контроля качества не предназначена для использования сама по себе. Вместо этого он будет служить компонентом гипотетической гораздо более широкой системы интерактивного диалога. Виртуальные агенты предназначены для многих задач, в том числе в качестве помощников по разработке программного обеспечения. Однако неразумно рассчитывать на создание такой системы за один шаг – в первую очередь требуется исследование подсистем и вспомогательных компонентов. В этом документе эта роль рассматривается по отношению к виртуальным агентам для задач SE. Важные следующие шаги включают в себя как проектирование других подсистем, так и расширение числа типов вопросов, которые может обрабатывать эта система контроля качества.

Чтобы способствовать продолжению исследований, мы публикуем все наши данные, исходный код подхода и рабочую интерактивную демонстрацию через наше онлайн-приложение: https://github.com/paqs2020/paqs2020

## ACKNOWLEDGMENTS

Эта работа частично поддерживается грантами NSF CCF-1452959 и CCF-1717607. Любые мнения, выводы и выводы, выраженные в настоящем документе, являются авторами и не обязательно отражают мнения авторов.

## РЕФЕРЕНСЫ

1. J. Yin, X. Jiang, Z. Lu, L. Shang, H. Li и X. Li, "Neural generative question answering", in *Proceedings of the Twenty Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, 2016, стр. 2972–2978.
2. М. Малиновски, М. Рорбах и М. Фриц, «Спросите свои нейроны: нейронный подход к ответам на вопросы об изображениях», в *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1–9.
3. J. Weston, A. Bordes, S. Chopra, A.M. Rush, B. van Merrienboer, ̈ A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.
4. W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu и F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.
5. H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems: Recent advances and new frontiers," *Acm Sigkdd Explorations Newsletter*, vol. 19, no. 2, pp. 25–35, 2017.
6. Гао, М. Галлей, Л. Ли *и др.* , «Нейронные подходы к разговорному ии», *Foundations and TrendsR in Information Retrieval*, vol. 13, no. 2-3, pp. 127–298, 2019.
7. З. Эберхарт, А. Бансал и К. Макмиллан, «Корпус апиза: диалоги об использовании Api с симулированным виртуальным помощником», 2020.
8. В. Самек, Т. Виганд и К.-Р. Мюллер, «Объяснимая искусственная теллигенция: понимание, визуализация и интерпретация моделей глубокого обучения», *препринт arXiv arXiv: 1708.08296*, 2017.
9. Г. Рас, М. ван Гервен и. Хаселагер, «Методы объяснения в глубоком обучении: пользователи, ценности, проблемы и проблемы» в *книге «Объяснимые и интерпретируемые модели в компьютерном зрении и машинном обучении*». Шпрингер, 2018, с. 19–36.
10. М.. Робийяр, А. Маркус, К. Треуде, Г. Бавота, О. Чапарро, Н. Эрнст, М. А. Героза, М. Годфри, М. Ланца, М. Линарес-Васкес *и др.* , «Документация разработчика по требованию», в *2017 году IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2017, стр. 479–483.
11. Н. Г. Уорд и Д. ДеВолт, «Проблемы построения высокоинтерактивных диалоговых систем», *AI Magazine*, том 37, No 4, стр. 7–18, 2016.
12. М. Джонсон и А. Вера, «Нет ай — это остров: аргументы в пользу объединения разведки», *AI Magazine*, vol. 40, no. 1, pp. 16–28, 2019.
13. В. Ризер и О. Лемон, *Обучение с подкреплением для адаптивных диалоговых систем: методология управления диалогом и генерации естественного языка на основе данных*. Springer Science & Business Media, 2011.
14. Y. Chen, L. Wu и M. J. Zaki, «Двунаправленные сети внимательной памяти для ответов на вопросы над базами знаний», в *Трудах Североамериканского отделения Ассоциации компьютерной лингвистики 2019 года: технологии человеческого языка, том 1 (Long and Short Papers*, 2019, pp. 2913–2923.
15. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks", *arXiv preprint arXiv:1511.05493*, 2015.
16. А. Грейвс, Г. Уэйн, М. Рейнольдс, Т. Харли, И. Данихелька, А. ГрабскаБарвинска, С. Г. Колменарехо, Э. Грефенстетт, Т. Рамальо, Дж*.* , «Гибридные вычисления с использованием нейронной сети с динамической внешней памятью», *Nature*, vol. 538, no. 7626, p. 471, 2016.
17. C. Xiong, S. Merity и R. Socher, «Динамические сети памяти для визуального и текстового ответа на вопросы», в *Международной конференции по машинному обучению*, 2016, стр. 2397–2406.
18. О. Лемон, «Учимся тому, что говорить и как это говорить: совместная оптимизация управления разговорным диалогом и генерации естественного языка», *Computer Speech & Language*, vol. 25, no. 2, pp. 210–221, 2011.
19. A. Wood, P. Rodeghero, A. Armaly и C. McMillan, «Обнаружение типов речевых актов в разговорах разработчиков вопросов и ответов во время исправления ошибок» в *материалах 26-го совместного заседания ACM 2018 года на Европейской конференции по разработке программного обеспечения и симпозиума по основам разработки программного обеспечения*. АКМ, 2018, с. 491–502.
20. Х. Кумар, А. Агарвал, Р. Дасгупта, С. Джоши и А. Кумар, «Маркировка последовательности актов диалога с использованием иерархического кодировщика с crf», *AAAI*, 2018.
21. З. Чэнь, Р. Ян, З. Чжао, Д. Цай и X. Он, «Признание акта диалога через crf-внимательную структурированную сеть» *на 41-й Международной конференции ACM SIGIR по исследованиям и разработкам в области информационного поиска*. ACM, 2018, с. 225–234.
22. . Блунсом, Н. Кальчбреннер и Н. Кальчбреннер, «Рекуррентные сверточные нейронные сети для композиционности дискурса» в *трудах семинара 2013 года по моделям непрерывного векторного пространства и их композиционности*. Материалы семинара 2013 года по моделям непрерывного векторного пространства и их композиционности, 2013 год.
23. S. Burger, K. Weilhammer, F. Schiel и H. G. Tillmann, "Verbmobil data collection and annotation", in *Verbmobil: Foundations of speechto-speech translation*. Спрингер, 2000, с. 537–549.
24. Х. Хэ, Д. Чен, А. Балакришнан и. Лян, «Стратегия разъединения и генерация в переговорных диалогах», в *Трудах Конференции 2018 года по эмпирическим методам в обработке естественного языка*, 2018, стр. 2333–2343.
25. Э. Рейтер и Р. Дейл, *Построение систем генерации естественного языка*.

Нью-Йорк, Нью-Йорк, США: Cambridge University Press, 2000.

1. Л. Дэн и Ю. Лю, *Глубокое обучение в обработке естественного языка*. Спрингер, 2018.
2. W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.
3. М. Мэн, С. Стейнхардт и А. Шуберт, «Документация по интерфейсу прикладного программирования: чего хотят разработчики программного обеспечения?» *Журнал технического письма и коммуникации*, том 48, No 3, стр. 295–330, 2018.
4. W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the understanding of program comprehension," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, p. 31, 2014.
5. М. Монперрус, М. Эйхберг, Э. Текес и М. Мезини: «О чем должны знать разработчики? эмпирическое исследование директив документации API», *Empirical Software Engineering*, vol. 17, no. 6, pp. 703–737, 2012.
6. Э. Агаджани, К. Надь, Г. Бавота и М. Ланца, «Крупномасштабное эмпирическое исследование лингвистических антипаттернов, влияющих на API», в *2018 году IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2018, стр. 25–35.
7. A. Head, C. Sadowski, E. Murphy-Hill и A. Knight, «Когда не комментировать: вопросы и компромиссы с документацией api для проектов C++» в *Трудах 40-й Международной конференции по программной инженерии*. АКМ, 2018, с. 643–653.
8. M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.
9. В. Арнаудова, С. Гайдук, А. Маркус и Г. Антониол, «Использование поиска текста и обработки естественного языка в программной инженерии» в *трудах 37-й Международной конференции по программной инженерии - том 2*. IEEE Press, 2015, стр. 949–950.
10. Y. Tian, F. Thung, A. Sharma и D. Lo, "Apibot: Question answering bot for api documentation", в *2017 году 32-я Международная конференция IEEE/ACM по автоматизированной программной инженерии (ASE)*. IEEE, 2017, стр. 153–158.
11. A. J. Ko и B. A. Myers, "Designing the whyline: a debugging interface for asking questions about program behavior", in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2004, с. 151–158.
12. P. Pruski, S. Lohar, W. Goss, A. Rasin, and J. Cleland-Huang, "Tiqi: answering unstructureed natural language trace queries," *Requirements Engineering*, vol. 20, no. 3, pp. 215–232, 2015.
13. Н.C. Брэдли, Т. Фриц и Р. Холмс, «Контекстно-зависимые разговорные помощники разработчиков», в *трудах 40-й Международной конференции по программной инженерии*. ACM, 2018, с. 993–1003.
14. H. Ed-Douibi, G. Daniel, and J. Cabot, "Openapi bot: A chatbot to help you understand rest apis", in *Web Engineering*, M. Bielikova, T. Mikkonen, and C. Pautasso, Eds. Чам: Springer International Publishing, 2020, стр. 538–542.
15. Н. Берингер, У. Картал, К. Лука, Ф. Шил, У. Турк *и др.* , «Обещание: процедура оценки мультимодальных интерактивных систем» в *трудах семинара «Оценка мультимодальных ресурсов и мультимодальных систем*». Citeseer, 2002, с. 90–95.
16. M. A. Walker, D. J. Litman, C. A. Kamm и A. Abella, "Evaluationing interactive dialogue systems: Extending component evaluation to integrated system evaluation", in *Interactive Spoken Dialog Systems: Bringing Speech and NLP Together in Real Applications*, 1997.
17. Y. Chen, L. Wu и M. J. Zaki, "Graphflow: Exploiting conversation flow with graph neural networks for conversational machine comprehension", *arXiv preprint arXiv:1908.00059*, 2019.
18. Z. Lin, X. Huang, F. Ji, H. Chen и Y. Zhang, «Task-oriented conversation generation using heterogeneous memory networks» в *трудах Конференции 2019 года по эмпирическим методам обработки естественного языка*, 2019 год.
19. Д. Богданау, К. Чо и Ю. Бенджио, «Нейронный машинный перевод путем совместного обучения выравниванию и переводу», *препринт arXiv: 1409.0473*, 2014.
20. T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.
21. А. Шрестха и А. Махмуд, «Обзор алгоритмов и архитектур глубокого обучения», *IEEE Access*, том 7, стр. 53040–53065, 2019.
22. С. Пуянфар, С. Садык, Ю. Ян, Х. Тянь, Ю. Тао, М.. Рейес, М.-Л. Шю,

S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, p. 92, 2018.

1. W. Yu, L. Wu, Q. Zeng, Y. Deng, S. Tao, and M. Jiang, «Crossing variational autoencoders for answer retrieval», in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 5635–5641.
2. С. А. Хаяти, Р. Оливье,. Аввару,. Инь, А. Томасич и Г. Нойбиг, «Генерация нейронного кода на основе поиска», в *Трудах Конференции 2018 года по эмпирическим методам обработки естественного языка*, 2018, стр. 925–930.
3. А. Леклер, С. Цзян и К. Макмиллан, «Нейронная модель для генерации резюме подпрограмм на естественном языке» в *трудах 41-й Международной конференции по программной инженерии*. IEEE Press, 2019, стр. 795–806.
4. A. LeClair, S. Haque, L. Wu и C. McMillan, "Improved code summarization through a graph neural network," *arXiv preprint arXiv:2004.02843*, 2020.
5. З. Чен, С. Я. Коммруш, М. Туфано, Л.-Н. Pouchet, D. Poshyvanyk и M. Monperrus, «Sequencer: Sequence-to-sequence learning for endto-end program repair», *IEEE Transactions on Software Engineering*, 2019.
6. E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: mining and search internet-scale software repositories," *Data Mining and Knowledge Discovery*, vol. 18, pp. 300–336, 2009.
7. А. Леклер и К. Макмиллан, «Рекомендации по наборам данных для обобщения исходного кода», в *Трудах конференции Североамериканского отделения Ассоциации компьютерной лингвистики 2019 года: технологии человеческого языка, том 1 (длинные и короткие документы),* 2019, стр. 3931–3937.
8. Р. Крэггс и М.M. Вуд, «Оценка схем кодирования дискурса и диалога», *Computational Linguistics*, vol. 31, no. 3, pp. 289–296, 2005.
9. G. Wiese, D. Weissenborn, and M. Neves, "Neural domain adaptation for biomedical question answering," *arXiv preprint arXiv:1706.03610*, 2017.
10. В. Кастелли, Р. Чакраварти, С. Дана, А. Ферритто, Р. Флориан, М. Франц, Д. Гарг, Д. Ханделвал, С. Маккарли, М. Маккоули *и др.* , «Набор данных techqa», *препринт arXiv arXiv: 1911.02984*, 2019.
11. Чжао и Й. Лю, «Поиск ответов из слова Божьего: адаптация домена для нейронных сетей в библейских ответах на вопросы», в *2018 году Международная совместная конференция по нейронным сетям (IJCNN).* IEEE, 2018, стр. 1–8.
12. U. Alon, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," *CoRR*, vol. abs/1808.01400, 2018. [Онлайн]. Доступно: http://arxiv.org/abs/1808.01400
13. U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *CoRR*, vol. abs/1803.09473, 2018. [Онлайн]. Доступно: http://arxiv.org/abs/1803.09473
14. Y. Wan, J. Shu, Y. Sui, G. Xu, Z. Zhao, J. Wu и P. S. Yu, "Multimodal attention network learning for semantic source code retrieval", in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ser. АСЭ '19. IEEE Press, 2019, с. 1325. [Онлайн]. Доступно: https://doi.org/10.1109/ASE.2019.00012
15. A.M. Lamb, A. G. A. P. Goyal, Y. Zhang, S. Zhang, A.C. Courville и Y. Bengio, «Professor forcing: A new algorithm for training recurrent networks», in *Advances In Neural Information Processing Systems*, 2016, pp. 4601–4609.
16. А.M. Логар, Э.M. Корвин и У. Дж. Олдхэм, «Сравнение рекуррентных алгоритмов обучения нейронных *сетей» на Международной конференции IEEE по нейронным сетям*. IEEE, 1993, стр. 1129–1134.
17. К. Дойя, «Рекуррентные сети: алгоритмы обучения», *The Handbook of Brain Theory and Neural Networks*, pp. 955–960, 2003.
18. Г. Шридхара, Л. Поллок и К. Виджай-Шанкер, «Автоматическое обнаружение и описание действий высокого уровня в рамках методов» в *трудах 33-й Международной конференции по программной инженерии*. АКМ, 2011, с. 101–110.
19. . У. Макберни и К. Макмиллан, «Автоматическое обобщение исходного кода контекста для методов Java», *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.
20. Г. Шридхара, Э. Хилл, Д. Муппанени, Л. Поллок и К. Виджай-Шанкер, «К автоматическому созданию сводных комментариев для методов Java» в *трудах международной конференции IEEE/ACM по автоматизированной разработке программного обеспечения*. АКМ, 2010, с. 43–52.
21. Р. К. Инь, *Тематические исследования и приложения: Дизайн и методы*.

Публикации Sage, 2017.

1. Показатели соглашения количественно оценивают надежностьy, но не разрешать разногласия. Поскольку в конечном итоге нам пришлось принимать решения о создании набора данных, мы решили разрешить разногласия за счет метрики надежности, как предложили Крэггс и Макги [55]. [↑](#footnote-ref-1)