Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ГЕНЕРАЦИЯ KOTLIN КОДА С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ АРХИТЕКТУРЫ TRANSFORMER ДЛЯ ФАЗЗИНГА КОМПИЛЯТОРА

Автор: Тихонов Виталий Андреевич _____

Направление подготовки: 01.03.02 Прикладная математика и информатика

Квалификация: Бакалавр

Руководитель ВКР: Фильченков А.А., канд. физ.-мат. наук _____

Санкт-Петербург, 2021 г.

Обучающийся Тихонов Виталий Андреевич
Группа М3436 Факультет ИТиП

Направленность (профиль), специализация
Математические модели и алгоритмы в разработке программного обеспечения

Консультанты:

а) Петухов В.А., без степени, без звания _____

ВКР принята «_____» _____ 20___ г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты «15» июня 2021 г.

Секретарь ГЭК Павлова О.Н. _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

УТВЕРЖДАЮ

Руководитель ОП

проф., д.т.н. Парфенов В.Г. _____

«____» _____ 20___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Обучающийся Тихонов Виталий Андреевич

Группа M3436 **Факультет** ИТиП

Квалификация: Бакалавр

Направление подготовки: 01.03.02 Прикладная математика и информатика

Направленность (профиль) образовательной программы: Математические модели и алгоритмы в разработке программного обеспечения

Тема ВКР: Генерация Kotlin кода с помощью нейронной сети архитектуры Transformer для фаззинга компилятора

Руководитель Фильченков А.А., канд. физ.-мат. наук, доцент, научный сотрудник Университета ИТМО

2 Срок сдачи студентом законченной работы до: «31» мая 2021 г.

3 Техническое задание и исходные данные к работе

Требуется попробовать применить нейронные сети, построенные на архитектуре Transformer для генерации кода на языке Kotlin. В архитектуру сети должна быть заложена генерация кода не только в соответствии с грамматикой языка (правила синтаксиса), но и в соответствие с некоторыми правилами семантики.

4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)

а) Провести обзор существующих архитектур нейронных сетей Transformer и сделать аргументированный выбор трансформера для генерации кода.

б) Изучить существующие способы делать code embedding и выбрать наиболее подходящий для решаемой задачи.

в) Обучить сеть на наборе тестов компилятора и/или фрагментов кода в багтрекере.

г) Применить построенный генератор кода для поиска проблем в компиляторе Kotlin: выбрасываемых исключений и проблем с производительностью.

5 Перечень графического материала (с указанием обязательного материала)

Графические материалы и чертежи работой не предусмотрены

6 Исходные материалы и пособия

Исходные материалы и пособия работой не предусмотрены

7 Дата выдачи задания «01» сентября 2020 г.

Руководитель ВКР _____

Задание принял к исполнению _____ «01» сентября 2020 г.

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Обучающийся: Тихонов Виталий Андреевич

Наименование темы ВКР: Генерация Kotlin кода с помощью нейронной сети архитектуры Transformer для фаззинга компилятора

Наименование организации, в которой выполнена ВКР: Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: Разработать модель поддерживающую обучение элементам семантики языка для более качественной генерации кода

2 Задачи, решаемые в ВКР:

а) реализуется модель

б) сравнивается с чем-то там

в) используется для генерации кода и выявления проблем компилятора

3 Число источников, использованных при составлении обзора: 0

4 Полное число источников, использованных в работе: 0

5 В том числе источников по годам:

Отечественных			**Иностранных**		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
0	0	0	0	0	0

6 Использование информационных ресурсов Internet: нет

7 Использование современных пакетов компьютерных программ и технологий:

Пакеты компьютерных программ и технологий	**Раздел работы**
Пакет `tabularx` для чуть более продвинутых таблиц	**??**, Приложения **??**, **??**
Пакет `biblatex` и программное средство `biber`	Список использованных источников

8 Краткая характеристика полученных результатов

Получился, надо сказать, практически неплохой стилевик. В 2015–2018 годах его уже использовали некоторые бакалавры и магистры. Надеюсь на продолжение.

9 Гранты, полученные при выполнении работы

Автор разрабатывал этот стилевик исключительно за свой счет и на добровольных началах. Однако значительная его часть была бы невозможна, если бы автор не написал в свое время кандидатскую диссертацию в LaTeX, а также не отвечал за формирование кучи научно-технических отчетов по гранту, известному как «5-в-100», что происходило при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01).

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы

По теме этой работы я (к счастью!) ничего не публиковал. Однако покажу, как можно ссылаться на свои публикации из списка литературы:

1 *Буздалов М. В.* Генерация тестов для олимпиадных задач по программированию с использованием генетических алгоритмов // Научно-технический вестник СПбГУ ИТМО. — 2011. — 2(72). — С. 72–77.

2 *Buzdalov M.*, *Shalyto A.* Hard Test Generation for Augmenting Path Maximum Flow Algorithms using Genetic Algorithms: Revisited // Proceedings of IEEE Congress on Evolutionary Computation. — 2015. — P. 2121–2128.

Обучающийся Тихонов В.А. _____

Руководитель ВКР Фильченков А.А. _____

«____» _____ 20___ г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Процесс тестирования очень важен при разработке приложений. Причина в том, что продукт, работающий некачественно может доставлять пользователям дискомфорт, приводить к потере времени и денег, а это в свою очередь может стать поводом для отказа от продукта. Еще важнее тестировать приложения, не являющиеся конечным продуктом, ведь проблемы в них могут затронуть еще большее число пользователей. Одним из таких приложений являются компиляторы языков программирования.

Классический подход в разработке тестов - написание тестов программистами, сразу после обновления функционала программы. Очевидно, что таким способом сложно протестировать такую большую программу как компилятор достаточно хорошо, поэтому необходимы и другие методы тестирования.

Примером такого метода может быть фаззинг. В процессе фаззинга для тестируемой программы генерируется большое количество входных данных. Для каждого примера входных данных программа запускается независимо, и исследуются некоторые характеристики ее работы, такие, например, как затрачиваемая память и время. Входные данные на которых поведение программы аномально (например, большое количество затраченной памяти или долгое время работы) выделяются для дальнейшего изучения программистом.

В данной работе в качестве тестируемого приложения выбран компилятор языка Kotlin. В случае с фаззингом компитлятора в качестве входных данных будут использоваться программы, написанные на соответствующем языке, а критериями аномальности могут служить время компиляции, затраченная память, выброшенные исключения, отличающееся поведение откомпилированных входных программ на разных бэкэндах.

Таким образом задача тестирования сводится к задаче генерации кода. Есть два основных аспекта, которые следует учитывать при генерации кода - это синтаксис и семантика языка. С синтаксисом языка программирования не должно возникать серьезных проблем - он описывается формальной грамматикой, и нет трудностей в генерации кода на ее основе. С семантикой все сложнее - у нее нет формального описания. В случае Kotlin’а она описывается спецификацией на естественном языке. Реализовать генератор, полностью поддерживающий спецификацию - задача сопоставимая с разработкой само-

го компилятора. Более того, генератор будет иметь неформальную основу, а значит высоки шансы допустить ошибки при его разработке.

При этом опираться только на синтаксис нельзя - доля семантически некорректных, а значит некомпилируемых программ слишком высока. Поэтому возникает желание изучить семантику с помощью машинного обучения, вместо того чтобы формализовывать ее или писать сложный генератор, описывая всю спецификацию.

Таким образом формулируется цель работы - разработать модель нейронной сети, обучаемую семантике языка Kotlin для генерации корректного кода для фаззинга компилятора.

описание разделов

ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

говорим о представлении кода

Модели машинного обучения отлично умеют работать с числами. Однако работать напрямую с текствой информацией они не могут. Тут на помощь приходит механизм эмбедингов - данные разбиваются на структурные компоненты (например, текст можно разбить на слова), и каждой такой компоненте сопостовляется вектор чисел, называемый эмбедингом. Таким образом каждый элемент кодируется точкой в некотором пространстве. Обработанные таким образом данные уже можно подавать на вход модели.

Существуют способы предлагать эти вектора по-умному, сохраняя некоторую семантическую информацию. Например, может получиться так, что вектор между точками, соответствующим словам ”Москва”и ”Россия”будет коллинеарен аналогичному вектору для слов ”Прага”и ”Чехия”. То есть такой вектор будет иметь смысл ”x является столицей y”.

Текст имеет достаточно простую структуру - это по сути просто последовательность слов. Для работы с последовательностями существуют рекурентные нейронные сети, принимающие данные фармент за фрагментом. Исходный же код имеет более сложную древовидную структуру. Это приводит к некоторым интересным спецэффектам. Например, на уровне файла мы можем поменять местами определеня двух функций, и код от этого никак не изменится. Однако, если в тексте поменять местами два абзаца, то смысл может значительно поменяться. какой-нибудь еще пример Выходит, что рассматривать код как обычный текст - не лучшая затея, так как часть информации из входных данных просто не будет использоваться.

Другое хорошо изученное представление данных - изображения. Для работы с ними так есть целый класс нейронных сетей - сверточные. Изображения рассматриваются как двумерные объекты, полностью заполняя некоторый прямоугольник.

Графовое предствление данных находится где-то по середине, не умещаясь в одномерное пространство, при этом не заполняя полностью двумерное. Поэтому для графов как правило используются специальные подходы.

Про то как работать с графами???

За последнее время тема популярна, есть статьи, ляляля. Рассмотрим подробнее несколько подходов по представлению исходного кода и выберем наиболее подходящий для генерации кода.

1.1. Графовые нейронные сети

AST = граф для GNN

1.2. Дерево разбора как список правил

AST = список правил;

1.3. Дерево разбора как набор путей

AST = paths

Почему-то используем AST = paths (мб он поновее, GNN - ресурсозатратный, подход неоднократно применялся)

Сказать что нигде нет семантики и надо бы добавить

Задача: внедрить механизмы работы с семантикой в модель

Подход для эмбедингов типов

Выводы по главе 1

вфвф

ГЛАВА 2. ОБРАБОТКА СЕМАНТИКИ

Во второй главе исследовательской работы описывается: Предлагаемое теоретическое решение (подход/метод/алгоритм/схема) Обоснование, почему оно удовлетворяет требованиям, сформулированным в первой главе. Теоретическое сравнение с существующими решениями

Выводы по главе 2

В конце каждой главы желательно делать выводы. Вывод по данной главе — нумерация работает корректно, ура!

ГЛАВА 3. РЕАЛИЗАЦИЯ

Имплементация модели

Извлечение путей

Архитектура генератора

Интеграция сематники

Результаты генерации

Выводы по главе 3

фывфв

ЗАКЛЮЧЕНИЕ

В данном разделе размещается заключение.

fff