



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет»
(ДВФУ)

Институт математики и компьютерных технологий (Школа)
Академия цифровой трансформации

Петров Сергей Дмитриевич

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
Магистерская диссертация

**ИЗВЛЕЧЕНИЕ ПРИЗНАКОВОГО ПРЕДСТАВЛЕНИЯ ИСХОДНОГО КОДА С
ИСПОЛЬЗОВАНИЕМ МЕТОДОВ ОБУЧЕНИЯ БЕЗ УЧИТЕЛЯ ДЛЯ DOWNSTREAM
ОБУЧЕНИЯ МОДЕЛЕЙ**

по направлению подготовки
09.04.01 «Информатика и вычислительная техника»,
магистерская программа «Искусственный интеллект и большие данные»

Владивосток
2025

В материалах данной выпускной квалификационной работы не содержатся сведения, составляющие государственную тайну, и сведения, подлежащие экспортному контролю

Уполномоченный по экспортному контролю

Е. В. Сапрыкина
(подпись) (И.О.Ф.)
« 07 » июля 2021 г.

Автор работы _____
(подпись)
Группа М9119-09.04.01иид
« 07 » июля 2021 г.

Руководитель ВКР _____
(должность, уч. степень, уч. звание)
А. Г. Тыщенко
(подпись) (И.О.Ф.)
« 07 » июля 2021 г.

Консультант
А. Г. Тыщенко
(подпись) (И.О.Ф.)
« 07 » июля 2021 г.

Назначен рецензент
зав. лаб. НЦВИ ИОФ РАН, к.ф.-м.н.
(уч. степень, уч. звание)
Луньков Андрей Александрович
(фамилия, имя, отчество)

Защищена в ГЭК с оценкой

Секретарь ГЭК
Т. С. Тихонова
(подпись) (И.О.Ф.)
« 07 » июля 2021 г.

«Допустить к защите»

Академии цифровой трансформации, к.э.н.
Е. В. Сапрыкина
(подпись) (И.О.Ф.)
« 07 » июля 2021 г.

АННОТАЦИЯ

Аннотация

Данная выпускная квалификационная работа посвящена исследованию методов обучения без учителя для извлечения признаковых представлений исходного кода с целью их дальнейшего использования в downstream-задачах машинного обучения. В современных условиях разработки программного обеспечения анализ и обработка исходного кода играют ключевую роль в таких задачах, как предсказание дефектов, автоматический рефакторинг, классификация кода и поиск уязвимостей. Однако эффективное представление кода в машиночитаемом формате остается сложной задачей, требующей применения современных методов искусственного интеллекта.

Цель данной работы заключается в адаптации алгоритма самообучения DINO для работы с текстовыми данными, в частности с исходным кодом, и сравнительном анализе его эффективности с готовыми моделями представления кода. В рамках исследования был проведен анализ алгоритма, предложена его модификация для обработки текстовых последовательностей, обучены векторные представления исходного кода и выполнена их оценка на downstream-задачах, включая классификацию кода.

Результаты работы демонстрируют потенциал методов обучения без учителя для автоматического извлечения информативных признаков из исходного кода. Разработанные подходы могут быть интегрированы в инструменты статического анализа, системы контроля качества кода и другие решения, направленные на повышение эффективности разработки программного обеспечения.

СОДЕРЖАНИЕ

Введение	5
Актуальность задачи	5
Цель работы	5
Задачи исследования	6
Структура работы	6
1 Описание предметной области	7
1.1 Машинное обучение	7
1.2 Глубокое обучение	10
1.3 Обработка естественного языка	11
1.4 Обзор существующих аналогов	12
2 Методология работы	17
2.1 Требования к аппаратному обеспечению	17
2.2 Требование к программному обеспечению	17
2.3 Описание данных	17
2.4 Алгоритм обучения	18
3 Реализация и тестирование	23
3.1 Обучение модели	23
3.2 Сравнение моделей	24
4 Заключение	26
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	27

Введение

В современной разработке программного обеспечения исходный код является ключевым ресурсом, требующим эффективного анализа и обработки. С ростом сложности программных систем и увеличением объёмов кодовой базы традиционные методы анализа кода сталкиваются с рядом ограничений, связанных с масштабируемостью и точностью. В таких задачах, как автоматическое обнаружение уязвимостей, рефакторинг, поиск семантически схожих фрагментов кода и предсказание дефектов, критически важным становится наличие качественного признакового представления исходного кода, которое могло бы быть использовано в downstream-моделях машинного обучения.

Актуальность задачи

По мере роста объемов разработки ПО и старения кодовой базы корпоративных приложений, повышение продуктивности разработки и модернизация устаревших систем становятся критически важными задачами. Достижения в области глубокого обучения и машинного обучения уже привели к прорывам в компьютерном зрении, распознавании речи, обработке естественного языка и других областях, что вдохновило исследователей применять методы ИИ для повышения эффективности разработки программного обеспечения. Это способствовало стремительному развитию нового направления исследований - "ИИ для программирования".

Цель работы

Адаптация алгоритма DINO для извлечения признаковых представлений исходного кода и сравнительный анализ его эффективности с готовыми моделями (CodeBERT, UnixCoder) на downstream-задачах, таких как классификация кода.

Задачи исследования

1. Провести обзор современных методов представления исходного кода и алгоритмов самообучения (self-supervised learning).
2. Модифицировать алгоритм DINO для работы с текстовыми данными.
3. Собрать и предобработать датасеты для обучения и оценки моделей.
4. Обучить модель на основе адаптированного DINO и сравнить её с существующими решениями и проанализировать результаты.

Структура работы

1. Описание предметной области
2. Описание технических требований
3. Реализация обучения
4. Оценка работы

1 Описание предметной области

1.1 Машинное обучение

Машинное обучение (ML) – это раздел искусственного интеллекта, изучающий методы построения алгоритмов, способных автоматически обучаться и улучшать свою работу на основе данных без явного программирования. В отличие от традиционных алгоритмов, где поведение системы жестко задаётся разработчиком, модели машинного обучения выявляют закономерности в данных и используют их для прогнозирования, классификации или принятия решений

Результат обучения алгоритма называется моделью – параметризованное отражение(функция), которое преобразует объекты из пространства входных признаков в пространство предсказаний.

Одним из главных требований к модели – ее способность к обобщению. Благодаря этому модель не просто запомнит данные на которых училась, а находит в них закономерности, что позволяет более точно делать отражение на новых объектах.

Алгоритмы ML делятся на три основные категории:

- Обучение с учителем
- Обучение с учителем
- Обучение с подкреплением

1.1.1 Обучение с учителем

Обучение с учителем – это вид машинного обучения, при котором модель обучается на примерах, где для каждого входного объекта известен правильный ответ. Цель такого метода обучения – построение модели которая будет способна предсказать ответ для ранее не встречавшихся примеров с заданной точностью.

Основное преимущество обучения с учителем:

- *Интерпретируемость* – модель работает с заранее определенным пространством выходных значений, так как разметка чаще составляется

человеком.

– *Интуитивная оценка качества* – для оценки часто используется интуитивно понятные метрики такие как точность или средний квадрат ошибки.

Основные недостатки обучения с учителем:

– *Зависимость от качества входных данных* – эффективность модели напрямую определяется качеством размеченных данных, процесс создания которых требует значительных временных и трудозатрат. Для сложных задач объём требуемых данных может возрастать экспоненциально, что создаёт существенные практические ограничения.

– *Проблема переобучения* – существует риск избыточной подгонки модели под особенности обучающей выборки – в таком случае алгоритм начинает воспроизводить не только значимые закономерности, но и случайные шумы, что резко снижает его способность к обобщению на новых данных.

Пример моделей которые обучаются с помощью обучения с учителем:

- Линейная регрессия
- Дерево решений
- Метод опорных векторов

1.1.2 Обучение без учителя

Обучение без учителя – это вид машинного обучения, при котором модель обучается на примерах для которых нет какой-либо разметки. Цель такого метода обучения – построение модели которая сама будет находить закономерности, не опираясь на внешние подсказки.

Задачи обучения без учителя включают в себя:

- *Кластеризацию* – задача разделения объектов на группы, которые имеют сходство между собой и отличаются от других.
- *Снижение размерности* – задача уменьшение количества признаков данных, сохраняя при этом информацию об объекте.
- *Поиск ассоциативных правил* – задача выявления устойчивых взаимосвязей между событиями в больших данных.

– *Генеративные модели* – задача генерации новых данных похожих на тренировочные.

Основное преимущество обучения без учителя:

– *Не требуется разметка данных* – работа с неразмеченными данными значительно облегчает процесс сбора данных.

– *Гибкость и универсальность* – применимо в разнообразных областях, а так же может использоваться для предобработки данных перед обучением с учителем

Основные недостатки обучения без учителя:

– *Сложность оценки качества* – отсутствие разметки затрудняет объективную оценку результатов.

– *Проблема интерпретируемости* – так как пространство выходных значений не известно, сложно их интерпретировать.

Пример моделей которые обучаются с помощью обучения с учителем:

– К средних

– Генеративные состязательные сети

1.1.3 Обучение с подкреплением

Обучение с подкреплением – это вид машинного обучения, при котором агент(модель) обдается на основе опыта взаимодействия со средой, принимая решения которые максимизируют награду.

В отличие от прошлых методов, агенты ориентированы на последовательное принятие решений в условиях неопределенности.

Основное преимущество обучения с подкреплением:

– *Подходит для задач с отложенной наградой* – Может учитывать долгосрочные последствия действий, а не только мгновенную выгоду.

– *Возможность обучения без размеченных данных* – Не требует готовых ”правильных ответов”.

Основные недостатки обучения с подкреплением:

– *Высокие вычислительные затраты* – Требует миллионов (иногда миллиардов) попыток для обучения.

– *Проблема исследования-эксплуатации* – Агент должен балансировать между исследованием и эксплуатацией. Исследование – проба новых действий, для поиска лучшей стратегии. Эксплуатация – использование уже известных лучших действий.

Пример моделей которые обучаются с помощью обучения с учителем:

- К средних
- Генеративные состязательные сети

1.2 Глубокое обучение

Глубокое обучение – это подраздел машинного обучения, основанный на использовании искусственных нейронных сетей (NN). Эти модели способны автоматически извлекать признаки из данных, имитируя работу человеческого мозга в упрощённой форме.

Свое название оно получило благодаря многослойной архитектуре нейронных сетей, что позволяет моделям находить более сложные закономерности. Например, в задаче классификации изображений первый слой находит базовые признаки, такие как края и прямые линии. Второй слой используя уже обработанные данные находит углы. Таким образом при достаточном количестве слоев нейронная сеть сможет отличать котов от собак.

К сожалению у нейронных сетей есть и недостатки:

- *Сложность интерпретации* – так как NN работает как ”черный ящик“ (мы знаем вход и выходы модели, но не знаем что происходит внутри).
- *Вычислительная сложность* – современные архитектуры могут иметь миллионы и миллиарды параметров, которые необходимо хранить в оперативной памяти.

В настоящее время глубокое обучение стало неотъемлемой частью нашей жизни. Благодаря ему работают голосовые помощники, системы рекомендаций и системы распознавания лиц.

Основные направления в которых применяют глубокое обучение:

- *Компьютерное зрение* – Детекция объектов, классификация и генерация изображений.
- *Обработка естественного языка* – Машинный перевод, языковое моделирование.
- *Обучение с подкреплением* – Робототехника, Игровые AI.

1.3 Обработка естественного языка

Обработка естественного языка – направления искусственного интеллекта, объединяющее лингвистику и компьютерные науки для анализа, понимания и генерации человеческого языка. Современные NLP-системы способны обрабатывать текст не только на уровне слов, но и понимать контекст, иронию и даже культурные отсылки.

В задачах обработки естественного языка входные данные по своей природе не имеют жестких ограничений по длине и структуре. По этому для эффективной обработки таких данных требуются архитектуры способные учитывать произвольные зависимости между элементами последовательности.

Чтобы представить последовательность текста в виде числа используют токенизаторы. Токенизатор – инструмент, который разбивает текст на отдельные элементы. Элементы могут быть как словами, так и отдельными символами. Этот процесс помогает стандартизировать данные и сокращает объем данных. В современных токенизаторах токен может описывать самые часто встречающиеся слова, отдельные части слов(корни, суффиксы, окончания) так и отдельные символы.

Для того чтобы легче менять значимость токена, используется слой Embedding. Он принимает на вход максимальное количество токенов, и размерность векторов представления. Далее создается матрица в которой каждому токену присваивается своя строка. В процессе работы модели последовательность токенов преобразуется в последовательность из строк.

Для обучения векторов embedding делают предобучение модели.

Предобучение – процесс первоначального обучения нейросети на большом объеме неразмеченных данных с использованием методов Self Supervised Learning (SSL). На этом этапе модель так же учится понимать синтаксис, семантику и контекст слов, что позволяет ей в дальнейшем эффективно адаптироваться к узким задачам. Основными методами предобучения являются MLM[6] и RTD.

После предобучения модель дообучают на меньших размеченных датасетах под конкретную задачу. Например, дообученная сеть может быть использован для классификации спама или генерации ответов в чат боте. Такой подход экономит время и улучшает качество модели, поскольку предобучение закладывает в модели базовое понимание языка. Таким образом даже маленький набор целевых данных может демонстрировать высокую точность.

1.4 Обзор существующих аналогов

Современные подходы к извлечению признаковых представлений исходного кода можно условно разделить на две основные категории: специализированные языковые модели для программного кода и универсальные модели, адаптированные для работы с кодом.

Среди наиболее значимых представителей первого направления выделяется CodeBERT[1] – двуязычная трансформерная модель, разработанная Microsoft Research, которая обучается на парных данных ”код-описание” с использованием модифицированных задач маскированного языкового моделирования (MLM) и обнаружения заменённых токенов (RTD).

Второе направление ярко представлено моделью UniXcoder[7], которая предлагает унифицированный подход к обработке кода через совместное использование различных модальностей (последовательность токенов, абстрактное синтаксическое дерево и граф потока данных), что позволяет достичь более полного понимания структурных и семантических особенностей программного кода.

1.4.1 Маскированное языковое моделирование

Masked Language Modeling (MLM) – это ключевая задача предобучения в современных языковых моделях, где модель учится предсказывать специально замаскированные токены в исходном тексте или коде на основе контекста. Этот подход позволяет нейросетям глубоко усваивать синтаксические и семантические зависимости в данных.

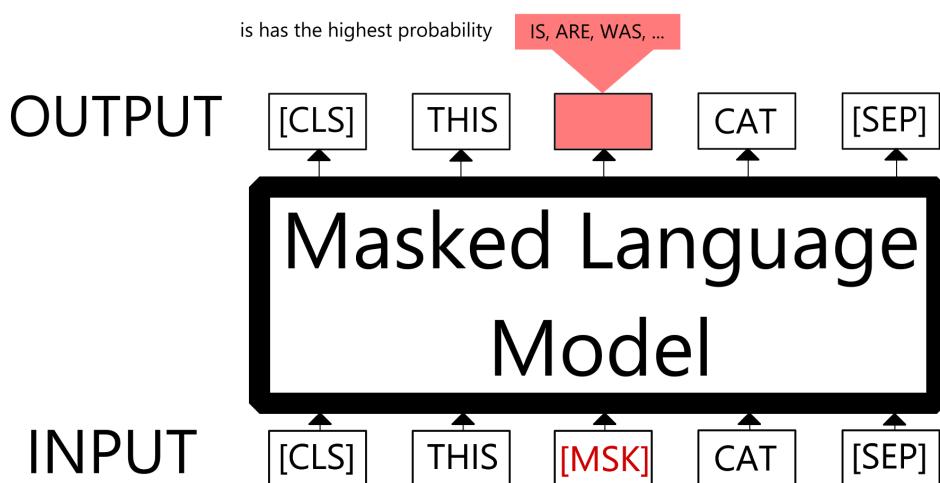


Рисунок 1 – Пример задачи MLM

На рисунке 1 изображен классический процесс маскированного языкового моделирования. Обучение происходит в 3 шага:

1. Маскирование – выбирается случайное количество токенов. Из них большая часть заменяется на токен [MASK], а оставшиеся либо не меняются, либо заменяются на случайное слово
2. Предсказание – модель анализирует контекст вокруг маски и вычисляет вероятность токенов кандидатов.
3. Функция потерь – ошибка считается только для замаскированных позиций.

Почему же используют MLM

- Контекстное обучение – модель учится находить взаимосвязи токенов, а не только статистику.

– Универсальность – подходит для любых последовательностей, не требуя разметку.

– Подготовка к downstream-задачам – навык востановления контекста полезно для автопролонгации текста или же исправления ошибок.

1.4.2 Обнаружения заменённых токенов

Replaced Token Detection (RTD) – это вспомогательная задача предобучения, используемая в современных языковых моделях для более эффективного обучения представлений.

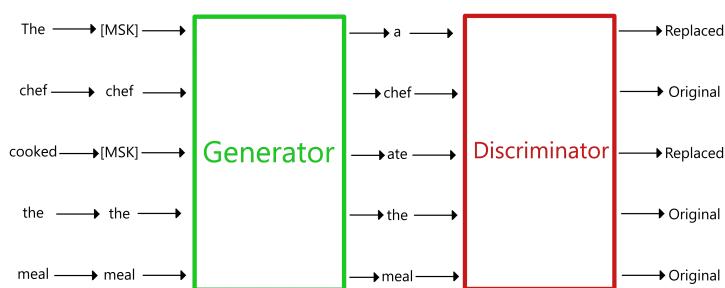


Рисунок 2 – Пример задачи RTD

На рисунке 2 изображен классический процесс обучения модели через задачу обнаружения заменённых токенов. Обучение происходит в 3 шага:

1. Замена токенов – маленькая модель генератора заменяет токены на новые, схожие по смыслу. К примеру running заменится на jogging
2. Предсказание – модель анализирует контекст вокруг замены выявить является ли токен замененным.
3. Функция потерь – ошибка считается для замененных токенов и оригинальных.

Почему же используют RTD

- Эффективность данных – все токены используются в момент обучения.
- Вычислительная сложность – низкая у RTD потому что используется бинарная классификация.

1.4.3 DINO

Интуитивно кажется что решение задачи MLM не способствует тому что модель будет понимать структуру кода или какие-либо отличительные черты.

В ходе исследования внимание было обращено внимание на метод обучения [4][3][8] разработанный FacebookResearch. Он позволяет эффективно обучать модели на изображениях без разметки, извлекая универсальные визуальные представления.

Ключевая идея — обучение через "самодистилляцию", когда модель учится согласовывать разные измененные версии одного изображения, не требуя предварительно размеченных данных.

Для обучения используется две модели — учитель и ученик. Учитель генерирует эталонные представления изображения, тогда как ученик пытается предсказать выход учителя.

В процессе обучения изображения разделяются на участки. Сначала из изображения выделяются два глобальных участка(изображение включающее 90% исходного). Затем из изображения выделяются шесть локальных участков (изображения включающие около 15% исходного). После этого к каждому набору участок применяются аугментации(например отражение, изменение цветов и так далее) и глобальные участки передаются в учителя, а локальные в ученика.



Рисунок 3 – Пример multicrop augmentation

Так же в loss функции используется техника заострения(Sharpening),

которая делает выходное вероятностное распределение более пиковым, уменьшая энтропию и подчеркивая наиболее вероятные классы. Это помогает избежать тривиальных решений, когда модель вырождается и предсказывает равномерное распределение для всех входов. Sharpening выглядит как Softmax с параметром τ .

$$P(x)^{(i)} = \frac{\exp\left(\frac{g_\theta(x)^{(i)}}{\tau}\right)}{\sum_{k=1}^K \exp\left(\frac{g_\theta(x)^{(k)}}{\tau}\right)} \quad (1)$$

В формуле 1 показана функция sharpening для i элемента модели g с параметрами θ .

$$\text{Loss} = -P_t(x) \log \mathbb{P}_s(x) \quad (2)$$

В формуле 2 показана функция потерь для учителя и студента.

В качестве Loss функции используется стандартная кросс энтропия.

В процессе обучения ученик изменяет свои веса таким образом чтобы уменьшить разницу предсказания с предсказанием учителя. Веса учителя постепенно изменяются обновляясь как экспоненциальное скользящее среднее весов ученика.

2 Методология работы

2.1 Требования к аппаратному обеспечению

Для обучения нейронной сети требуется:

1. Графический процессор (GPU) с высокой производительностью и тензорными ядрами. Рекомендуется использовать GPU от Nvidia не менее чем с 16 ГБ видеопамяти. Меньший объем памяти может существенно увеличить необходимое время для обучения модели.
2. Хранилище данных которое обеспечит высокую скорость доступа к файлам. Рекомендуется использовать SSD.
3. Оперативная память для хранения датасета. Рекомендуется 16 ГБ.

Так же можно увеличить количество графических процессоров, так как код позволяет обучать модель на нескольких GPU одновременно.

2.2 Требование к программному обеспечению

Для обучения нейронной сети требуется:

1. Операционная система на базе Linux, в связи с тем что libuv не может работать на Windows.
2. Язык программирования python 3.11 или выше.
3. Библиотека глубокого обучения PyTorch
4. Библиотека машинного обучения Scikit Learn
5. Среда разработки: Pycharm, VSCode, Jupyter Notebook
6. Параллельная вычислительная платформа и программный интерфейс CUDA

Важно убедиться что программные средства совместимы между собой.

2.3 Описание данных

Для обучения модели было решено использовать код только на одном языке, таким образом уменьшится разнообразие в данных и модель сможет обучиться легче.

Из-за распространенности было решено остановиться на языке

программирования C, в связи с его популярностью и так как множество языков программирования С подобные.

Тренировочный набор данных был собран с помощью репозитория CodeforcesYalink. Он предоставляет инструмент для автоматизированного сбора примеров кода с платформы codeforces. С помощью этого инструмента можно извлекать код участников по определенным параметрам, таким как язык программирования или идентификатору задания.

Были собраны данные из 104 задач. Количество тренировочных данных: 36489

Для оценки результата работы использованы данные из Project CodeNet[2] от IBM. Это масштабный датасет, содержащий более 14 миллионов образцов кода на 55 языках программирования. Датасет включает разнообразные задачи, метаданные(например статус выполнения, время работы кода, потребление памяти).

Из этого набора данных было выбрано 48 задач, в которых было больше всего примеров кода на C и у которых статус выполнения принят.

2.4 Алгоритм обучения

Для обучения было решено модернизировать алгоритм dino для работы с текстом.

Для этого было необходимо сделать

- Переработать модель из ViT в Transformer
- Преобразовать multicrop augmentation
- Выбрать подходящие аугментации

2.4.1 Описание архитектуры самой маленькой нейронной сети из представленных

Изначально архитектура разбивала исходное изображение на кусочки размером 16×16 пикселей, потом при помощи линейного преобразования преобразовывались в вектора embedding. Затем к ним прибавлялись Positional Embedding и эти данные передавались дальше в блоки модели.

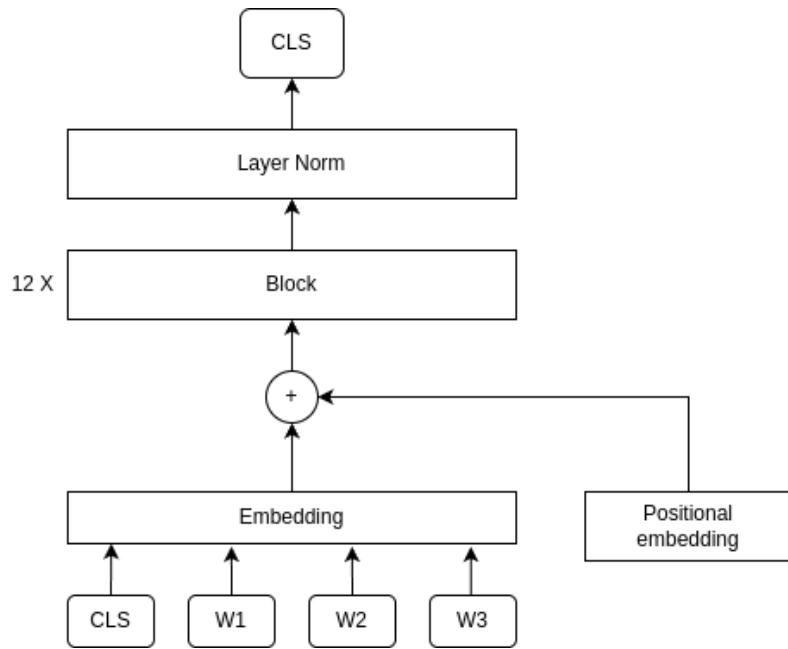


Рисунок 4 – Архитектура нейросети

Для изменения линейного преобразования был использован Embedding слой, который каждому токену присваивает свой вектор представления.

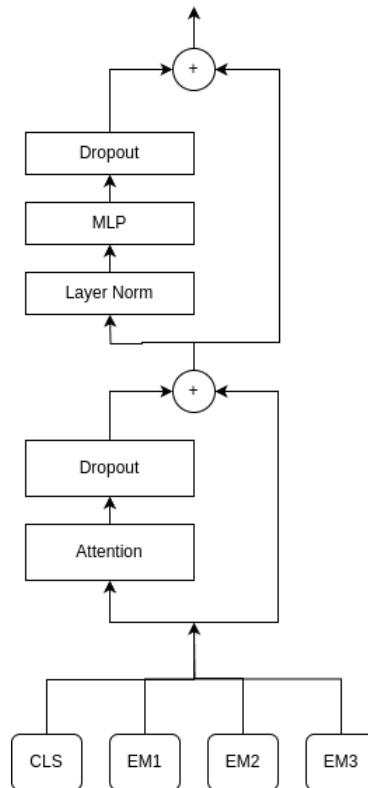


Рисунок 5 – Архитектура блока нейросети

Блок состоит из слоя нормирования, внимания, dropout, еще раз нормирования и MLP. Из них самым полезным и ресурсозатратным является

слой внимания. Его вычисление зависит от квадрата числа токенов в последовательности.

Так как изначальное изображение было размером 224×224 , то в Vit модели было всего 196 токенов. В текстовой же реализации токенов значительно больше.

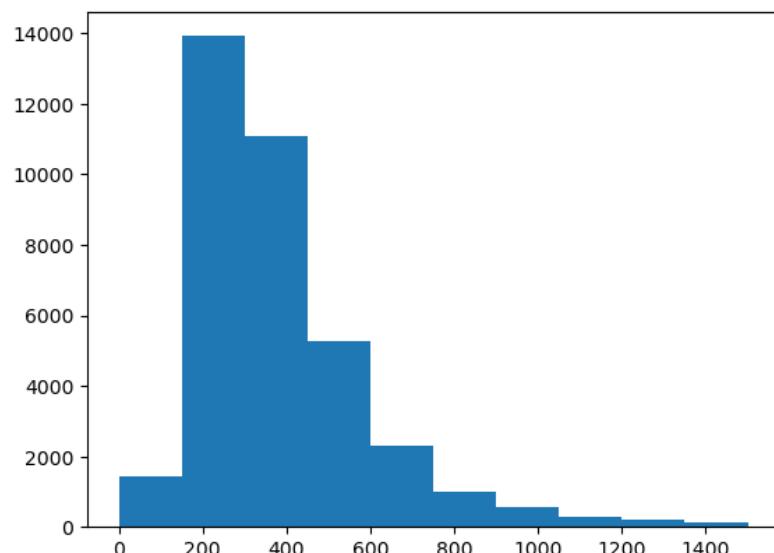


Рисунок 6 – Гистограмма распределения количества токенов в объектах тренировочного набора

Из рисунка 6 видно что оптимальным количеством токенов для рассмотрения будет 1000, что в 5 раз больше привычного количества токенов у блока. Соответственно вычисление attention становится медленнее в 25 раз.

В связи с неэффективностью стандартного attention слоя, было решено использовать встроенный в PyTorch FalshAttention[5]. Таким образом удалось существенно увеличить скорость работы и уменьшить потребление памяти.

После этих изменений модель смогла работать с большими последовательностями токенов.

2.4.2 Multicrop augmentation

В изначальной реализации участки выбирались случайным наложением границ на изображение. Для текста программ это не совсем подходит.

Было предложено 3 основных способа как делать код на блоки:

- Блоки отделяются по токенам

– Блоки делятся по строкам кода - так как конец строки это конец выражения

– Блоки делятся по ветвям синтаксического дерева - таким образом участок кода будет оконченным с точки зрения логики

Наилучшим из предложенных вариантов кажется разделение по ветвям синтаксического дерева, таким образом получится в качестве локальных участков брать завершенные логически блоки.

В конечной реализации было решено использовать разделение блоков по токенам, таким образом получится больше уникальных блоков, что на небольшом датасете наборе данных, что значительно увеличит разнообразие в данных.

После определения разделения, был реализован класс MulticropAugmentations, который на вход получает объект класса, который определяет как данные будут делиться на участки, и так же был реализован класс _Separate_by_tokens, который во время вызова получает список количеств токенов, которые нужно выбрать случайным образом.

2.4.3 Аугментация данных

Аугментация текстовых данных – не тривиальный процесс. Существует несколько основных подходов.

– Синтаксические методы – изменение структуры предложения с сохранением смысла. Основные способы: замена на синонимы, перестановка слов, добавление/удаление стоп слов.

– Семантические методы – использование моделей для генерации новых вариантов текста. Основные способы: обратный перевод, генерация текста с помощью LLM, замена слов через предобученные MLM модели.

– Методы основанные на шуме – добавление шума для повышения устойчивости модели. Основные способы: случайные вставки/удаления слов, обмен местами соседних букв или слов, имитация ошибок клавиатуры.

В конкретном случае в связи с тем что данные являются кодом на языке C, то синтаксические методы не подойдут из-за невозможности определения поведения

методов и функций. Семантические методы не подходят из-за специфики данных. А методы на основе шума не подходят из-за того что такие ошибки чаще всего отмечаются спелчекером из современных IDE.

После определения необходимых аугментаций все остальные были удалены.

3 Реализация и тестирование

3.1 Обучение модели

Следующим шагом работы было обучение моделей машинного обучения.

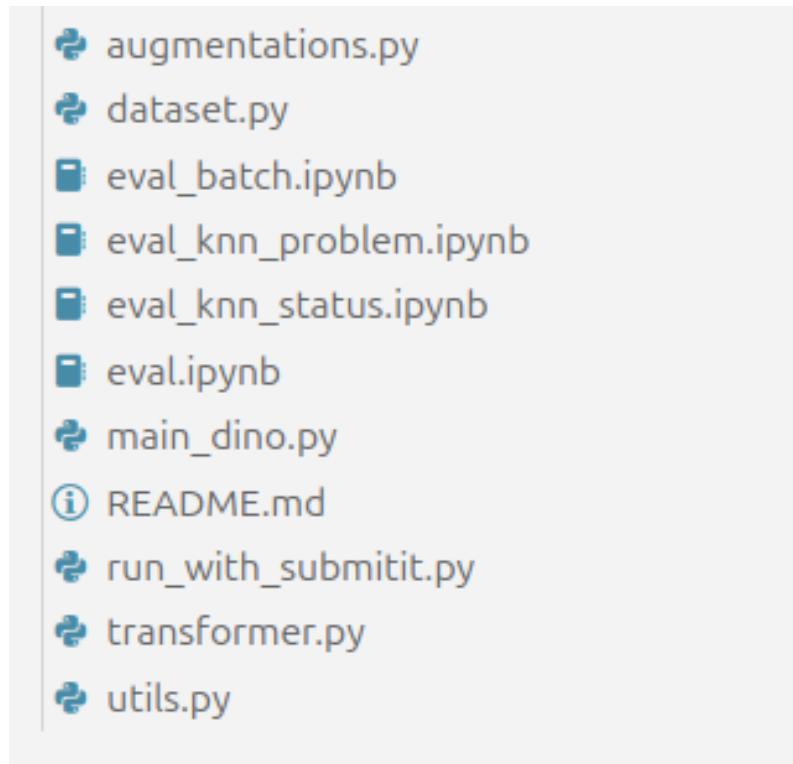


Рисунок 7 – Архитектура проекта

На рисунке 7 изображена архитектура проекта. Здесь для вызова обучения необходимо запустить файл `main_dino.py`, с необходимыми аргументами.

Перечень основных аргументов

- `arch` – название архитектуры, которая будет использована для создания модели, изначально выбран `t_tiny`.
- `token_slicer` – метод по отделения участков. Изначально стоит `Tokens` – разделение на участки по токенам.
- `max_token_count` – максимальное число токенов, в использованном токенизаторе это значение равняется 65536.
- `out_dim` – размерность пространства выхода Dino head. Изначально равно 65000.

- use_fp16 – использовать ли float 16 во время обучения или нет.

Изначально флаг равен Истине.

- batch_size_per_gpu – количество примеров которые будут загружены на одной видеокарте. Изначально стоит 16.

- epochs – количество эпох обучения. Изначально равно 100.

- local_crops_number – количество локальных участков на одно наблюдение. Изначально равно 8.

Для запуска был использован персональный компьютер с ОС Ubuntu, GPU Nvidia 3090 с 16гб видеопамяти, и 256 гигабайтами оперативной памяти. При этом, со всеми выставленными настройками было занято 14 из 16 гигабайт видеопамяти.

Весь процесс обучения занял 25 часов. Результатом работы был файл checkpoint.pth, в котором находились веса модели учителя и ученика, а так же данные из конфигурации процесса обучения.

3.2 Сравнение моделей

Сравнение было решено провести с моделями CodeBert и UnixCoder, так как они считаются лучшими на данный момент.

Изначально весь выбранный датасет был пропущен через модели DinoNLP, CodeBert и UnixCoder, а полученные вектора сохранены в пакетные файлы.

В качестве базового решения (baseline) был использован наивный баесовский классификатор(Naive Bayes Classifier). Наивный баесовский классификатор был выбран в качестве baseline-модели, поскольку он является классическим методом машинного обучения, широко применяемым для задач обработки текстовых данных.

3.2.1 Классификация номера задания

Было выбрано 48 заданий из набора данных от IBM в которых больше всего принятых решений. Размер тренировочного набора - 65059. Размер тестового набора - 16265. После этого было решено использовать KNN классификатор и MLP классификатор предсказания номера задачи.

Для KNN был выставлен параметр - 10 ближайших соседей.

MLP имело 11 слоев, с функциями активации ReLU.

	BaseLine		DinoNLP	CodeBert	UnixCoder
Acc, %	76	KNN	80	80	89
		MLP	85	81	95

Таблица 1 – Точность в процентах решения задачи определения номера задания

Решение при помощи модели DinoNLP значительно выше чем baseline, не уступает решению модели использующей представления модели CodeBert и хуже, чем UnixCoder.

3.2.2 Классификация статуса задачи

Из прошлого набора данных было выбрано одно задание в котором было больше всего примеров. В качестве целевой переменной было выбрано булевое значение, равен ли задача принятой.

Размер тренировочного набора - 7838, из них 44% приняты. Размер тестового набора - 1960, из них 44% приняты.

	BaseLine		DinoNLP	CodeBert	UnixCoder
Acc, %	72	KNN	72	72	73
		MLP	77	79	85

Таблица 2 – Точность в процентах решения задачи определения статуса решения

Решение при помощи модели DinoNLP значительно выше чем baseline, в некоторых случаях уступает решению модели использующей представления модели CodeBert и хуже, чем UnixCoder.

4 Заключение

В данной работе было исследовано извлечение признакового представления исходного кода с использованием методов обучения без учителя для последующего применения в downstream-задачах. Основной фокус был направлен на адаптацию подхода для обработки программного кода, что привело к созданию модели DinoNLP.

Несмотря на значительно меньший объем тренировочных данных по сравнению с CodeBERT, модель DinoNLP продемонстрировала близкое качество в задачах классификации кода.

Если MLM ориентированы на предсказание конкретных токенов, то фокусируется на извлечении обобщённых признаковых представлений данных, что делает его более предпочтительным для выбранных задач, так как модель DinoNLP учится извлекать признаки инвариантные к аугментациям.

Для улучшения стоит произвести обучение с большим количеством тренировочных данных и вычислительных блоков.

Модель DinoNLP подтвердила свою жизнеспособность как альтернатива feature extraction моделям. Хотя она и не превосходит CodeBERT в абсолютных метриках, ее эффективность при малых данных открывает новые возможности для внедрения ИИ в разработку ПО.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. CodeBERT: A Pre-Trained Model for Programming and Natural Languages / Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou. — 2020. — arXiv: 2002.08155 [cs.CL]. — URL: <https://arxiv.org/abs/2002.08155> (дата обращения: 11.03.2025).
2. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks / R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker, V. Thost, L. Buratti, S. Pujar, S. Ramji, U. Finkler, S. Malaika, F. Reiss. — 2021. — arXiv: 2105.12655 [cs.SE]. — URL: <https://arxiv.org/abs/2105.12655>.
3. DINOV2: Learning Robust Visual Features without Supervision / M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, P. Bojanowski. — 2024. — arXiv: 2304.07193 [cs.CV]. — URL: <https://arxiv.org/abs/2304.07193>.
4. Emerging Properties in Self-Supervised Vision Transformers / M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, A. Joulin. — 2021. — URL: <https://arxiv.org/abs/2104.14294> (дата обращения: 18.02.2025).
5. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness / T. Dao, D. Y. Fu, S. Ermon, A. Rudra, C. Ré. — 2022. — arXiv: 2205.14135 [cs.LG]. — URL: <https://arxiv.org/abs/2205.14135>.
6. Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little / K. Sinha, R. Jia, D. Hupkes, J. Pineau, A. Williams, D. Kiela. — 2021. — arXiv: 2104.06644 [cs.CL]. — URL: <https://arxiv.org/abs/2104.06644>.

7. UniXcoder: Unified Cross-Modal Pre-training for Code Representation / D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, J. Yin // arXiv preprint arXiv:2203.03850. — 2022. — (Дата обращения: 11.03.2025).
8. Vision Transformers Need Registers / T. Darcet, M. Oquab, J. Mairal, P. Bojanowski. — 2024. — arXiv: 2309.16588 [cs.CV]. — URL: <https://arxiv.org/abs/2309.16588>.