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

Уполномоченый по экспортному контролю

_____ _____
(подпись) (И.О.Ф.)
«__07__»_____июля_____ 2021 г.

Автор работы _____
(подпись)
Группа М9119-09.04.01иибд
«__07__»_____июля_____ 2021 г.

Руководитель ВКР _____
(должность, уч. степень, уч. звание)

_____ _____
(подпись) (И.О.Ф.)
«__07__»_____июля_____ 2021 г.

Консультант

_____ А. Г. Тыщено
(подпись) (И.О.Ф.)
«__07__»_____июля_____ 2021 г.

Назначен рецензент _____
(уч. степень, уч. звание)

_____
(фамилия, имя, отчество)

Защищена в ГЭК с оценкой

_____

Секретарь ГЭК

_____ _____
(подпись) (И.О.Ф.)
«__07__»_____июля_____ 2021 г.

«Допустить к защите»

Академии цифровой трансформации,

_____ _____
(подпись) (И.О.Ф.)
«__07__»_____июля_____ 2021 г.

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 Distillation with No Labels (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

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.

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”.

Таким образом целью является адаптация алгоритма DINO[6, 4, 15] для извлечения признаковых представлений исходного кода и сравнительный анализ его эффективности с готовыми моделями (CodeBERT[1], UnixCoder[14]) на downstream-задачах, таких как классификация кода.

Для достижения поставленной цели необходимо выполнить следующие задачи.

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.

1 Описание предметной области

1.1 Машинное обучение

Машинное обучение (Machine Learning, 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 отображение(функция), которое преобразует объекты из пространства входных признаков в пространство предсказаний.

В ходе обучения модель автоматически выявляет статистические закономерности между входными и выходными данными, анализируя множество примеров пар и оптимизируя функцию потерь. Такой подход не требует глубоких знаний предметной области или ручного конструирования признаков, поскольку модель самостоятельно выделяет релевантные паттерны непосредственно из данных. Благодаря архитектурным особенностям и регуляризации, обученная модель способна обобщать выявленные зависимости и применять их для предсказаний на новых данных.

Алгоритмы ML делятся на три основные категории:

– Обучение с учителем
– Обучение без учителя
– 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.

Основные преимущества обучения с учителем:

– *Интерпретируемость* – в задачах обучения с учителем ответы обычно интерпретируемы для человека, поскольку разметка данных выполняется вручную. Это позволяет моделям обучаться на четко определенных примерах, где каждому входу соответствует однозначный, понятный целевой признак.

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

– *Простота* – обучение с учителем представляет собой задачу аппроксимации таблично заданной функции, что делает его наиболее простым и формализуемым видом обучения.

Основные недостатки обучения с учителем:

– *Наличие разметки* – создание размеченных данных требует значительных временных затрат, а для узкоспециализированных областей может быть практически невозможна без привлечения экспертов.

– *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.

К моделям, обучаемым данным методом, можно отнести такие как:

– Линейная регрессия[20]

– Дерево решений[21]

– Метод опорных векторов[18]

1.1.2 Обучение без учителя

Обучение без учителя – это вид машинного обучения, при котором модель обучается на примерах, для которых известны только входные данные. В таких задачах, как правило, отсутствует верное решение, а качество результата оценивается исключительно с помощью специально разработанных метрик.

Задачи обучения без учителя включают в себя:

– *Кластеризацию* – задача разделения набора данных на однородные группы таким образом, чтобы объекты внутри одного кластера были максимально схожи между собой, а объекты из разных кластеров – максимально различны.

– *Снижение размерности* – задача уменьшения объема данных за счет выделения ключевой информации.

– *Поиск ассоциативных правил* – задача выявления устойчивых взаимосвязей между событиями в больших данных.

– *Генеративные модели* – задача создания новых объектов на основе некоторых входных данных, задающих параметры выходного объекта.

– *Извлечение признаков* – задача автоматического преобразования исходных данных в компактное и информативное векторое представление, пригодное для решения downstream задач.

Основные преимущества обучения без учителя:

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

– *Универсальность* – применимо к большему классу задач, поскольку для обучения требуется лишь мера качества предсказания, которая может быть основанна на чем угодно, а не только на ответах.

Основные недостатки обучения без учителя:

– *Сложность оценки качества* – отсутствие разметки требует построеия не менее эффективной и объективной функции оценки качества предсказания.

– *Проблема интерпретируемости* – во многих задачах обучения без

учителя предсказываемые моделью значения сложно интерпретируемы человеку за счет того, что оно опирается на некоторые функции, а не ответ человека.

К моделям, обучаемым данным методом, можно отнести такие как:

– KMeans[11]

– DBSCAN[8]

– GAN[16]

– LLM[3]

– Diffusion models[12]

– DINO

1.1.3 Обучение с подкреплением

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

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

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”.

– *Не требует дифференцируемости* - функция оценки не обязана быть дифференцируемой.

Основные недостатки обучения с подкреплением:

– *Сложная оценка* – необходимо получать внешнюю оценку, которая может требовать значительного времени вычисления(например, симуляции физических процессов) или человеческого вмешательства.

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

лучших действий. Для получения качественной модели алгоритм обучения должен позволять модели исследовать новые способы решения задачи и эксплуатировать уже изученные.

Пример моделей, которые обучаются с помощью обучения с подкреплением:

– LLM[3]

– Алгоритм автоматического управления транспортных средства.

– Игровой искусственный интеллект.

1.2 Глубокое обучение

Глубокое обучение – это подраздел машинного обучения, основанный на использовании искусственных нейронных сетей (NN). Свое название NN получили из-за попытки их построения на основе структуры нейронных сетей млекопитающих и состоят из нейронных слоев, каждый из которых поочередно активируется.

Нейронные сети состоят из взаимосвязанных слоев искусственных нейронов, которые обрабатывают входные данные через последовательность линейных преобразований и нелинейных функций активации. Каждый нейрон имеет настраиваемые параметры: веса и смещения, которые подбираются в процессе обучения для минимизации ошибки предсказания.

Такая архитектура позволяет моделям находить более сложные закономерности. Например, в задаче классификации изображений начальные слои определяют базовые признаки, такие как перепады света, простые геометрические фигуры. Второй слой, используя уже обработанные данные, определяет типы объектов на изображении. Таким образом модель может построить цепь признаков, получаемых друг из друга, которые помогают решить задачу с высокой точностью.

К сожалению, у нейронных сетей есть и недостатки:

– *Сложность интерпретации* – Признаки, извлекаемые моделью, представляют собой абстрактные числовые паттерны, работающие по принципу

”чёрного ящика” - их внутренняя логика не поддаётся содержательной интерпретации, что исключает возможность объяснения причин конкретных предсказаний и не гарантирует устойчивой работы на новых данных

– *Вычислительная сложность* – Некоторые методы машинного обучения, особенно глубокие нейронные сети с миллионами параметров и алгоритмы обработки больших данных, требуют значительных вычислительных ресурсов и продолжительное время обучения, что ограничивает их применение в условиях ограниченных аппаратных возможностей.

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, LLM.

– *Обучение с подкреплением* – Робототехника, Игровые AI, LLM.

Обучение нейронных сетей осуществляется методами градиентного спуска, которые итеративно корректируют параметры модели в направлении антиградента функции потерь. Ключевое требования для такой оптимизации – дифференцируемость всех компонентов сети, чтобы можно было вычислить градиенты с помощью алгоритма обратного распространения ошибки.

1.3 Обработка естественного языка

Обработка естественного языка – направление 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. Поэтому для эффективной обработки таких данных требуются архитектуры, способные учитывать произвольные зависимости между элементами последовательности.

Чтобы представить последовательность текста в виде последовательности чисел, используют токенизаторы. Токенизатор – инструмент, который разбивает текст на отдельные элементы. Этот процесс трансформирует человекопонятный текст в машинночитаемые токены. В современных токенизаторах токен может описывать как самые часто встречающиеся слова, отдельные части слов(корни, суффиксы, окончания), так и отдельные символы.

Для того чтобы легче менять значимость токена, используется слой Embedding. В результате работы данного слоя каждому токену сопоставляется вектор, при этом в процессе обучения эти векторы изменяются так, чтобы передавать смысловую нагрузку токенов.

В последнее время наиболее популярной архитектурой нейронных сетей является Transformer. В основе этой ахритектуры лежит механизм внимания(Attention), который позволяет модели анализировать взаимосвязи между всеми словами в тексте одновременно, независимо от их позиции. Attention вычисляет взвешанные зависимости между токенами, определяя, насколько каждое слово влияет на другие в последовательности, что обеспечивает контекстное понимание, превосходящее классические модели.

Так как Transformer не использует сверточные и рекурентные слои, он не имеет представления о порядке слов. Для учета позиционной информации применяют positional embedding – специальные векторы, добавляемые к embedding слов перед передачей на вход модели.

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[9] и RTD.

Задача извлечения признаков заключается в автоматическом преобразовании исходных данных в компактные числовые векторы, которые сохраняют ключевые характеристики данных и нужны для дальнейшего анализа.

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[14], 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.

Задача MLM обладает существенными ограничениями в формировании целостного понимания текста, поскольку фокусируется исключительно на

локальном восстановлении маскированных токенов в ограниченном контексте, игнорируя глобальную структуру и логические связи между предложениями.

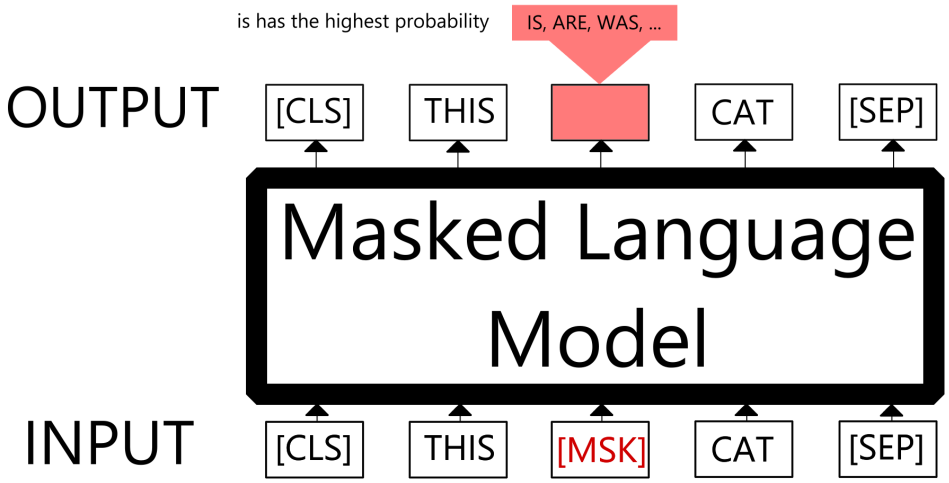


Рисунок 1 – Пример задачи MLM

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.

Основными преимуществами использования MLM являются:

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

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

– Подготовка к downstream-задачам – навык восстановления контекста полезен для автопродления 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.

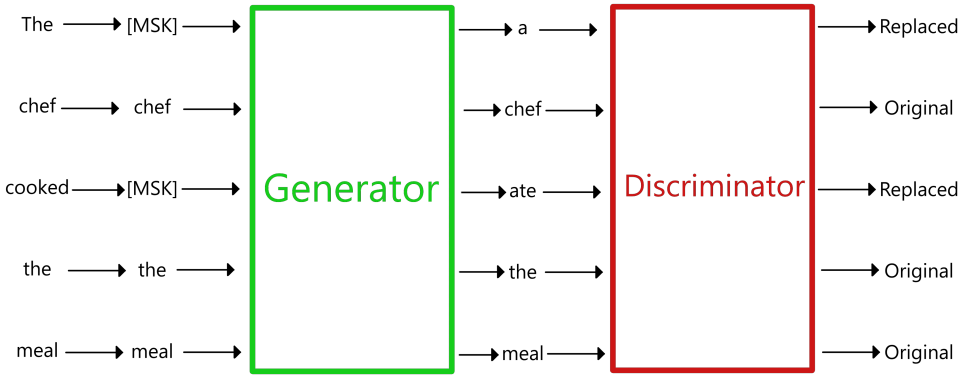


Рисунок 2 – Пример задачи RTD

На рисунке 2 изображен классический процесс обучения модели на задаче 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 текст и для каждого элемента делает предсказание, является ли он замененным.

3. Функция потерь – ошибка считается для замененных токенов и оригинальных.

Основными преимуществами использования RTD являются:

– Углубленное моделирование зависимостей текста – за счет детекции некорректных токенов подстановок, что усиливает контекстуальное понимание лингвистических паттернов.

– Качество представлений – превосходит MLM при равных ресурсах (например, ELECTRA[5] показывает результат схожий с моделью RoBERTa[10], однако используя в 4 раза меньше вычислений).

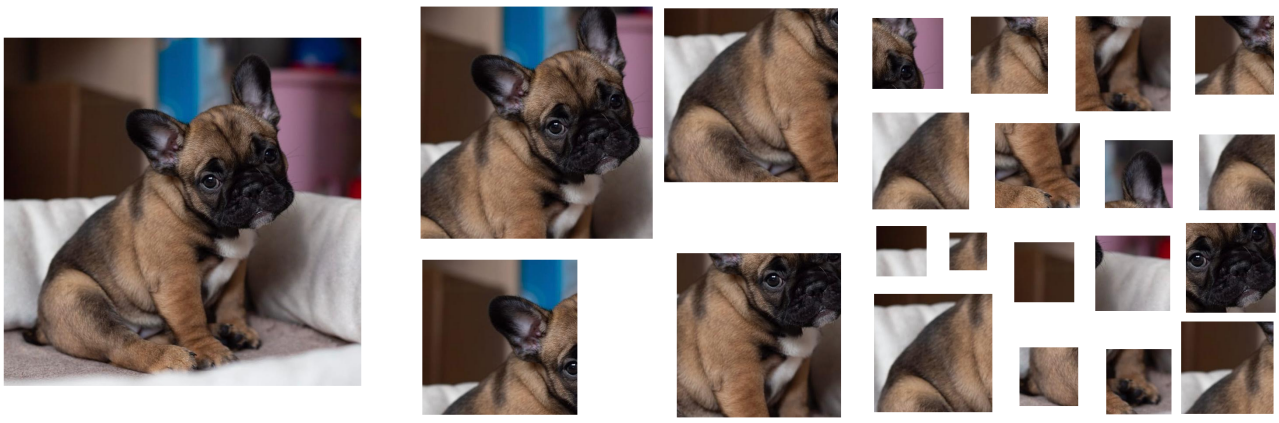
1.4.3 DINO

Задача MLM, несмотря на свою эффективность в предобучении языковых моделей, не обеспечивает понимания текста в силу локального характера оптимизации, где модель предсказывает токены, не требуя анализа глобальных связей. Так же модель может правильно угадывать слово не понимая смысла текста.

В последнее время большую популярность получил метод обучения DINO[6, 4, 15], разработанный FacebookResearch. Это современный алгоритм самообучения, который позволяет эффективно извлекать универсальные признаки из изображений без использования размеченных данных.

Ключевая идея DINO – принцип самодистеляции, где модель учится согласовывать представления одного изображения после различных аугментаций, формируя устойчивые и информативные представления.

Для обучения используется две модели – учитель и ученик. Учитель генерирует эталонные представления изображения, тогда как ученик пытается предсказать выход учителя. Во время обучения изменяются исключительно параметры ученика, тогда как веса модели учителя вычисляются как экспоненциально взвешенное среднее весов ученика на каждой итерации, обеспечивая стабильное самообучение на неразмеченных данных.



(a) Оригинальное изображение (b) Глобальные кропы (c) Локальные кропы

Для того, чтобы избежать предсказания константы в DINO применяют

центрирование. Из предсказания модели учителя вычитается скользящее среднее предыудщих предсказаний модели учителя. Центрирование смещает выход учителя так, чтобы среднее значение элементов вектора было близко к нулю, это раздвигает кластеры в пространстве признаков.

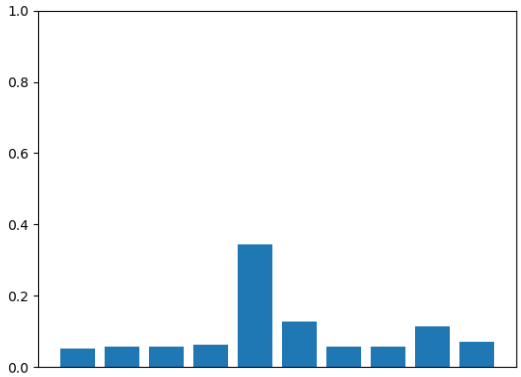
Также в loss функции используется техника заострения(Sharpening), которая делает выходное вероятностное распределение более пиковым, уменьшая энтропию и подчеркивая наиболее вероятные классы

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

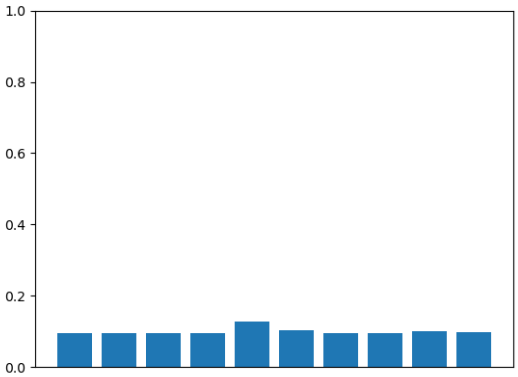
Это помогает избежать тривиальных решений, когда модель вырождается и предсказывает равномерное распределение для всех входов.

В алгоритме DINO функцией потерь является кросс энтропия – функция потерь, измеряющая расстояние между двумя распределениями вероятностей.

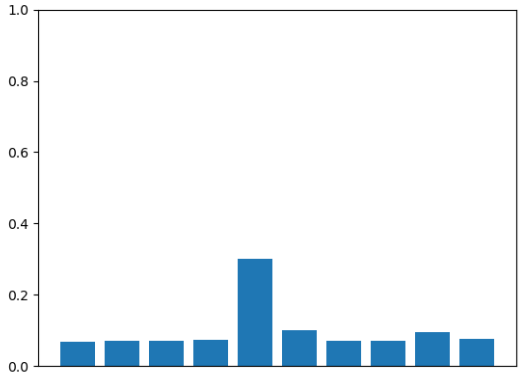
$$Loss = -P_t(x) \log P_s(x) . \quad (2)$$



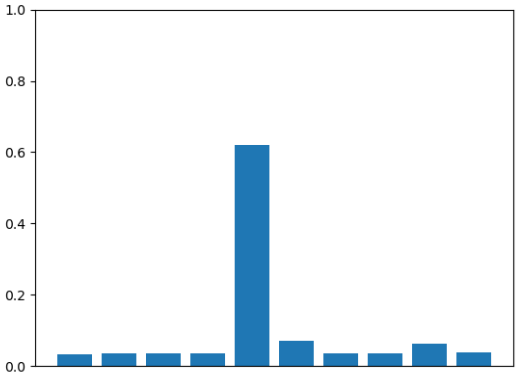
(a) Оригинальные данные



(b) Данные после sharpening, $\tau = 1$



(c) Данные после sharpening, $\tau = 0.2$



(d) Данные после sharpening, $\tau = 0.1$

Рисунок 4 – Распределение данных

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 графический процессор 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.

Большая скорость обучения модели может быть достигнута за счет использования большего количества графических процессоров.

2.2 Требование к программному обеспечению

Для обучения нейронной сети требуется:

1. Операционная система на базе Linux или любая другая, поддерживающая все используемые библиотеки.

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

Первое исследование было решено провести на примерах кода на одном языке программирования, чтобы исключить из исследования возможные зависимости от разных языков.

В качестве языка для экспериментов был выбран C в виду его невероятной

популярности и представленности в разработке кода, так же синтаксис многих современных языков был основан на синтаксисе языка C.

Тренировочный набор данных был собран с помощью репозитория CodeforcesYoink[17, 19]. 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 собранны данные из 104 задач. Количество тренировочных данных: 36489 примеров кода.

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).

```
#include <stdio.h>
int main(void){
   int i;
   scanf("%d",&i);
   printf("%d\n", i*i*i);
   return 0;
}
```

Рисунок 5 – Пример объекта из набора данных

Из всего набора данных было отобранно 48 задач. Для каждой задачи были выбраны решения выполненные на языке программирования C с наибольшим числом решений.

2.4 Алгоритм обучения

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

Для этого было необходимо сделать:

– Модифицировать репозиторий DINO для работы с оригинальной архитектурой трансформера

– Преобразовать multicrop augmentation

– Выбрать подходящие аугментации

2.4.1 Описание архитектуры нейронной сети

Изначально в методе обучения DINO была использована архитектура ViT – адаптация трансформерной архитектуры для обработки изображений, где входное изображение разбивается на последовательность неперекрывающихся патчей, которые линейно преобразуются в вектора embedding. Для составления единого вектора представления в начало последовательности добавляют токен CLS, финальное отображение которого используют для downstream задач.

Использованная архитектура ViT в DINO разбивала исходное изображение на кусочки размером 16×16 пикселей. Затем к ним прибавлялись Positional Embeddig и эти данные передавались дальше в блоки модели.

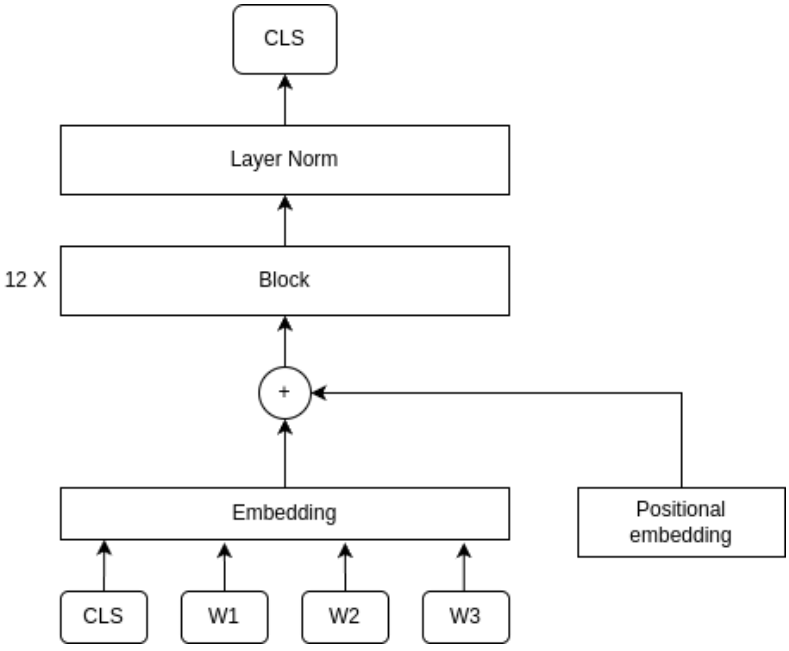


Рисунок 6 – Архитектура нейросети

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

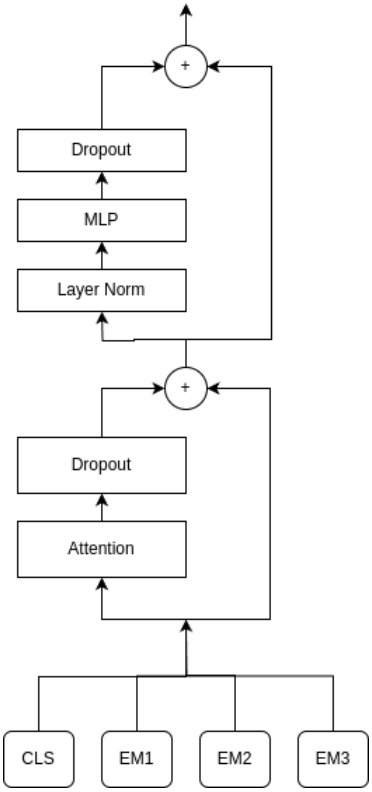


Рисунок 7 – Архитектура блока нейросети

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.

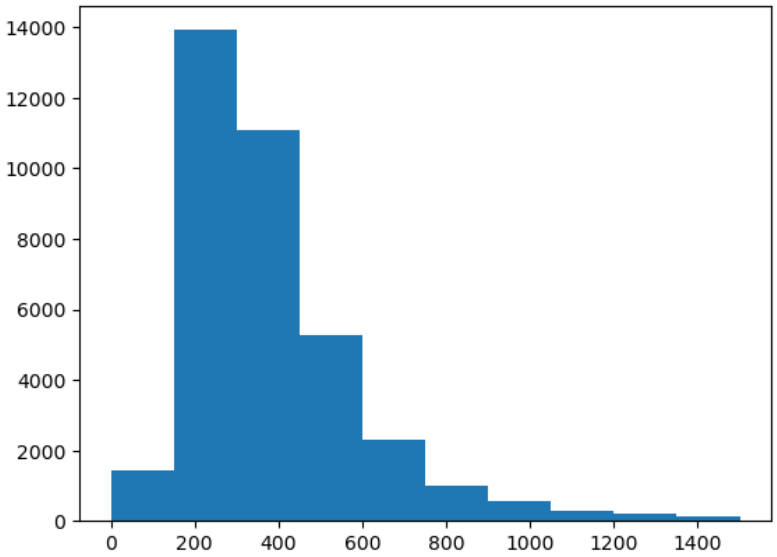


Рисунок 8 – Гистограмма распределения количества токенов в объектах тренировочного набора

Из рисунка 8 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[7]. Таким образом удалось существенно увеличить скорость работы и уменьшить потребление памяти.

После этих изменений модель смогла работать с большими последовательностями токенов.

2.4.2 Multcirop augmentation

Для обработки текста методом, аналогичным разбиению изображений на кропы, исходный текст сначала преобразуется в последовательность токенов. Затем эта последовательность делится на блоки случайного размера.

Для улучшения качества семантической значимости блоков можно делить код, основываясь на символе переноса строки, так как оконченная строка программного кода, по сути своей, уже является логическим выражением.

Развивая прошлую идею можно использовать инструменты для синтаксического анализа кода, чтобы разбивать текст на блоки, благодаря ветвям синтаксического дерева. Такой подход позволяет получать завершенные

логические блоки.

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

2.4.3 Аугментация данных

Аугментация текстовых данных является нетривиальной задачей, к которой есть несколько 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 предобученые 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, 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 отметаются спелчекермо из современных IDE.

После определения необходимых аугментаций все остальные были удалены.

3 Реализация и тестирование

3.1 Обучение модели

В данном разделе будет рассмотрен процесс обучения модели, включая выбор архитектуры и метода оптимизации.

3.1.1 Архитектура проекта

Проект состоит из следующих файлов:

– main.py – основной файл, который запускает процесс обучения

– eval_batch.ipynb – необходим для запуска обученной модели

– transformer.py – файл с описанием структуры архитектуры модели

– dataset.py – файл с реализацией класса датасета

– utils.py – файл с вспомогательными функциями

– augentations.py – файл с аугментациями данных

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 22.04.5 LTS, 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[1] и UnixCoder[14], так как они считаются лучшими на данный момент.

```
#include <stdio.h>\n
int main(void){\n
\t   int i;\n
\t   scanf("%d",&i);\n
\t   printf("%d\n", i*i*i);\n
\t   return 0;\n
}
```

Рисунок 9 – Код, разделенный токенизатором

На рисунке 9 изображен текст, разбитый на токены, где токены отличаются цветом.

Изначально весь выбраный датасет был пропущен через модели DINO, CodeBert и UnixCoder, а полученные вектора сохранены в numpy файлы.

Так как для сравнения будет решаться задача классификации, в качестве базового решения (baseline) был выбран наивный баесовский классификатор(Naive Bayes Classifier). Наивный байесовский классификатор

был выбран поскольку он является классическим методом машинного обучения, широко применяемым для задач обработки текстовых данных. Во время предобработки исходный текст был токенизирован и в дальнейшем токены были трансформированы с помощью TF-IDF.

3.2.1 Устойчивость представлений

В рамках исследования проведено сравнение устойчивости векторных представлений, сгенерированных различными моделями, к синтаксическим вариациям исходного кода. Для этого были созданы модифицированные версии исходного кода, включающие добавление комментариев, добавление функци и намеренное допущение ошибок. Каждая версия кода была преобразована в векторное представление, после чего выполнено сравнение полученных векторов с исходным представлением при помощи метрики косинусного расстояния.

Этот эксперимент позволяет определить, насколько устойчивы векторные представления к незначительным изменениям в коде: хорошая модель должна выдавать близкие векторы для семантически эквивалентных программ, даже если их поверхностный синтаксис отличается. Например, если косинусное расстояние между представлениями остаётся малым при переименовании переменных, значит, модель захватывает смысл кода, а не его поверхностные особенности.

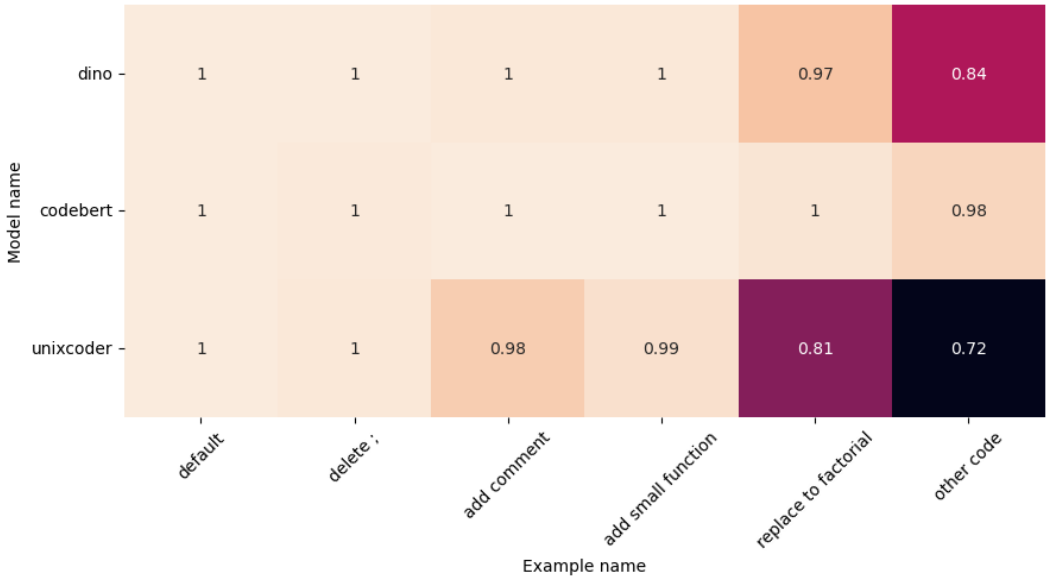


Рисунок 10 – Косинусное расстояние между default и другими версиями

Описание версий:

– default - программа для вычисления i числа последовательности Фибоначчи

– delete ; - код, из которого были удалены все символы завершения

– add comment - добавлен однострочный комментарий

– add small function - добавлена функция, суммирующая 2 переменные типа int

– replace to factorial - тело функции вычисления i числа последовательности заменено на вычисление факториала

– other code - изначальное решение полностью изменено, а новое решение выводит в консоль пирамидку из символов ”*”

Проведенный анализ(10) выявил существенные различия в реакции исследуемых моделей на синтаксические изменения исходного кода:

– Модель UniXcoder продемонстрировала наибольшую чувствительность к модификациям кода, реагируя на все значимые изменения, включая добавление комментариев и небольших функций. Высокая чувствительность свидетельствует о способности модели улавливать тонкие семантические различия.

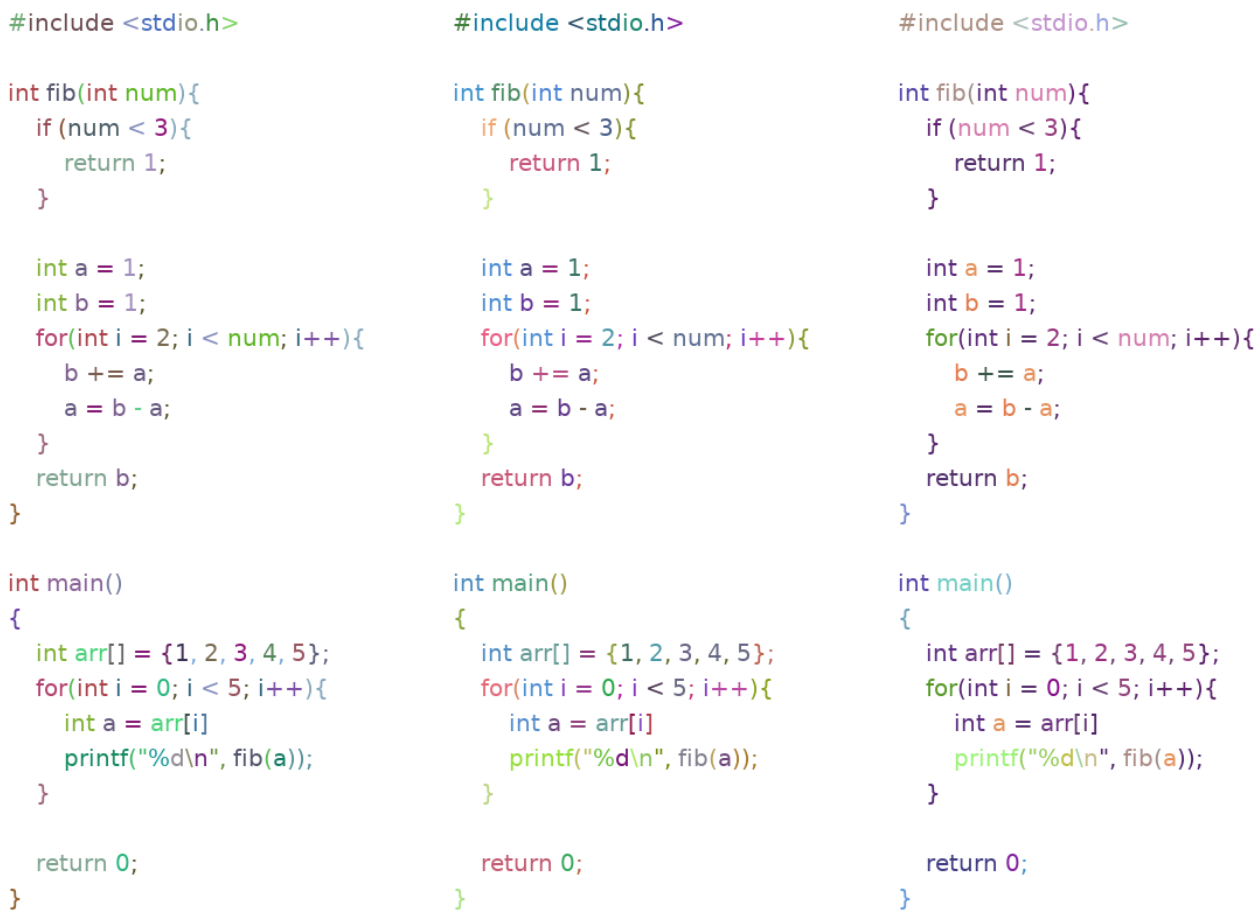
– Модель CodeBERT показала наименьшую реакцию на внесенные изменения, сохраняя близкие векторные представления даже при существенных модификациях кода. Это может указывать либо на чрезмерную обобщающую способность модели, либо на недостаточную чувствительность к локальным синтаксическим изменениям.

– Модель DINO заняла промежуточное положение, продемонстрировав сбалансированную реакцию: она надежно фиксировала значительные структурные изменения исходного кода, но при этом не проявляла избыточной чувствительности к незначительным синтаксическим вариациям.

Полученные результаты позволяют сделать вывод о том, что модель UniXcoder наиболее чуствительна, тогда как CodeBERT почти не реагирует на изменения. Модель DINO представляет собой компромиссный вариант, сочетающий разумную чувствительность к значимым изменениям с устойчивостью к несущественным модификациям.

3.2.2 Визуализация токенных представлений через цветовое пространство

Метод визуализации токенных представлений через цветовое представление позволяет интерпретировать семантические свойства текста через цветовую проекцию векторных представлений. Модель выдает похожие цвета семантически схожим токенам. Каждая из моделей обработала по 48 примеров решений, полученные представления для каждого из токенов в дальнейшем с помощью процесса снижения размерности t-SNE[13] представления были ужаты до 3х мерных. Полученные вектора были нормализованы и использованы как цвет для отображения токенов.



```c
#include <stdio.h>

int fib(int num){
   if (num < 3){
      return 1;
   }

   int a = 1;
   int b = 1;
   for(int i = 2; i < num; i++){
      b += a;
      a = b - a;
   }
   return b;
}

int main()
{
   int arr[] = {1, 2, 3, 4, 5};
   for(int i = 0; i < 5; i++){
      int a = arr[i]
      printf("%d\n", fib(a));
   }

   return 0;
}
```

(a) DINO

```c
#include <stdio.h>

int fib(int num){
   if (num < 3){
      return 1;
   }

   int a = 1;
   int b = 1;
   for(int i = 2; i < num; i++){
      b += a;
      a = b - a;
   }
   return b;
}

int main()
{
   int arr[] = {1, 2, 3, 4, 5};
   for(int i = 0; i < 5; i++){
      int a = arr[i]
      printf("%d\n", fib(a));
   }

   return 0;
}
```

(b) CodeBERT

```c
#include <stdio.h>

int fib(int num){
   if (num < 3){
      return 1;
   }

   int a = 1;
   int b = 1;
   for(int i = 2; i < num; i++){
      b += a;
      a = b - a;
   }
   return b;
}

int main()
{
   int arr[] = {1, 2, 3, 4, 5};
   for(int i = 0; i < 5; i++){
      int a = arr[i]
      printf("%d\n", fib(a));
   }

   return 0;
}
```

(c) UnixCoder

На основании представленных изображений можно сделать вывод, что модель DINO выделяет ключевые слова (такие как int, for, printf), при этом каждому ключевому слову назначается цвет, который не является заранее фиксированным (например, цвет может варьироваться в зависимости от того, определяет ли int локальную или глобальную переменную). Кроме того, в отличие от моделей CodeBert и UnixCoder, модель DINO присваивает различным переменным и константам разные цвета, что является важным отличием.

3.2.3 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		DINO	CodeBert	UnixCoder
Acc,%	76	KNN	80	80	89
		MLP	85	81	95

Таблица 1 – Точность в процентах решения задачи определения номера задания

Точность решения при помощи DINO значительно выше, чем у baseline. Не уступает решению модели, использующей представления модели CodeBert. Но в тоже время хуже, чем более сложная модель UnixCoder.

3.2.4 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, которое отображает является ли задача принятой.

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

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

Таблица 2 – Точность в процентах решения задачи определения статуса решения

Точность решений при помощи DINO значительно выше, чем у baseline. В некоторых случаях уступает решению модели, использующей представления модели CodeBert. И в тоже время хуже, чем более сложная модель 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 DINO для обработки программного кода, что привело к созданию модели DINO.

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

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

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

Модель DINO 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 // arXiv. — 2020.
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 // arXiv. — 2021.
3. D. C. Human Language Understanding Reasoning. — 2022.
4. 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 // arXiv. — 2024.
5. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators / K. Clark, M.-T. Luong, Q. V. Le, C. D. Manning // arXiv. — 2020.
6. Emerging Properties in Self-Supervised Vision Transformers / M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, A. Joulin. — 2021.
7. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness / T. Dao, D. Y. Fu, S. Ermon, A. Rudra, C. Ré // arXiv. — 2022.
8. Martin Ester H.-P. K. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. — 1998.
9. 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 // arXiv. — 2021.

10. RoBERTa: A Robustly Optimized BERT Pretraining Approach / Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov // arXiv. — 2019.

11. S. L. Least square quantization in PCM's. Bell Telephone Laboratories Paper. — 1957.

12. Score-Based Generative Modeling through Stochastic Differential Equations / Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, B. Poole // arXiv. — 2021.

13. Talk G. T. Visualizing Data Using t-SNE.

14. 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.

15. Vision Transformers Need Registers / T. Darcet, M. Oquab, J. Mairal, P. Bojanowski // arXiv. — 2024.

16. What to expect of artificial intelligence in 2017 //. MIT Technology Review. — 2017.

17. yolomachine. CodeforcesYoink. — 2021. — URL: https : / / github . com / yolomachine/CodeforcesYoink (дата обращения: 21.02.2025).

18. Вьюгин. В. Математические основы теории машинного обучения и прогнозирования. — 2013.

19. Гоменюк А. А. АНАЛИЗ ИСХОДНОГО КОДА ПРИ ПОМОЩИ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ. — 2021.

20. Демиденко Е. З. Линейная и нелинейная регрессия. // Финансы и статистика. — 1981.

21. Левитин А. В. Ограничения мощи алгоритмов: Деревья принятия решения // Алгоритмы. Введение в разработку и анализ. — 2006.