ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)**

Кафедра искусственного интеллекта

Факультета информационных технологий и анализа больших данных

***Дисциплина: «*Машинное обучение в семантическом и сетевом анализе»**

*Направление подготовки: «Прикладная математика и информатика»*

*Профиль: «Прикладное машинное обучение»*

*Факультет информационных технологий и анализа больших данных*

*Форма обучения очная*

*Учебный 2024/2025 год, 6 семестр*

**Курсовая работа на тему:**

«Применение методов глубокого обучения для классификации текстов на основе семантического анализа.»

*Выполнил:*

студент группы ПМ22-1

Гусев Я. Г.

*Научный руководитель:*

Пасканов В. Д.

**Москва 2025**

**Содержание**

[Введение 3](#_Toc199118480)

[Глава 1. Обзор современных подходов к анализу и классификации программного кода 6](#_Toc199118481)

[1.1. Постановка задачи классификации текстов 6](#_Toc199118482)

[1.2. Классические методы представления текстов 7](#_Toc199118483)

[1.3. Модели на основе эмбеддингов и трансформеров 9](#_Toc199118484)

[1.4. Графовые представления текстов и кода 10](#_Toc199118485)

[Глава 2. Теоретическое описание реализованных подходов 13](#_Toc199118486)

[2.1 Классификация кода с использованием Word2Vec и полносвязной нейронной сети 13](#_Toc199118487)

[2.2 Классификация с использованием CodeBERT и графовой нейронной сети 14](#_Toc199118488)

[2.3 Классификация на основе CodeBERT и полносвязной нейросети. 16](#_Toc199118489)

[2.4 Классификация на основе TF-IDF по символьным n-граммам и графовой агрегации через Node2Vec 16](#_Toc199118490)

[Глава 3. Практическая реализация и эксперименты 20](#_Toc199118491)

[3.1 Датасет и подготовка данных 20](#_Toc199118492)

[3.2 Модель на основе Word2Vec и полносвязной сети 20](#_Toc199118493)

[3.3 Модель с CodeBERT-эмбеддингами и графовой нейросетью 22](#_Toc199118494)

[3.4 Модель CodeBERT + полносвязная сеть 24](#_Toc199118495)

[3.5 Авторский метод: TF-IDF + Node2Vec + FC 24](#_Toc199118496)

[3.6 Сравнительный анализ и выводы 25](#_Toc199118497)

[Заключение 28](#_Toc199118498)

[Список использованных источников 30](#_Toc199118499)

[Приложение…………………………………………………………………………31](#_Приложение)

# ****Введение****

Классификация текстов на основе семантического анализа представляет собой одну из наиболее актуальных задач в области обработки естественного языка (NLP). Современные приложения, начиная с автоматического анализа отзывов пользователей и категоризации документов и заканчивая поддержкой систем рекомендаций и интеллектуальных помощников, требуют высокой точности и надёжности при определении тематики, стиля или языка текста. Несмотря на широкое распространение эвристических и статистических методов, которые основываются на частотном анализе токенов или шаблонах синтаксиса, эти подходы демонстрируют недостаточную устойчивость в нетривиальных ситуациях. В частности, популярные платформы для обмена сообщениями иногда ошибочно определяют язык фрагмента кода из-за недостаточности синтаксических признаков.

В последние годы для решения подобных задач активно применяются методы глубокого обучения, включая трансформеры (например, BERT), однако такие архитектуры могут требовать больших вычислительных ресурсов. Кроме того, такие подходы часто игнорируют семантическую близость между токенами. В связи с этим возникает необходимость в более эффективных методах, сочетающих точность и скорость.

Одним из перспективных направлений является использование графовых структур для моделирования отношений между элементами текста: они позволяют как строить граф над готовыми эмбеддингами (связав сниппеты, близкие по смыслу), так и получать новые эмбеддинги на основе графовых структур (например, с помощью Node2Vec). В курсовой работе рассматривается подход, где графовые модели используются для повышения скорости и качества классификации текстов — в частности, для задачи определения языка программирования по фрагментам исходного кода.

**Актуальность.**

Большое число редакторов кода, IDE, платформ анализа кода и специализированных форумов используют синтаксические шаблоны, частотные словари или ручные эвристики для определения языка программирования. Такие методы недостаточно хорошо работают в сложных случаях, таких как короткие или неоднозначные фрагменты кода, а также в случае гибридных контекстов (например, встраивание SQL-строки в Python). Учитывая широкое распространение платформ, нуждающихся в быстром и точном распознавании языка программирования, растёт необходимость в системах, позволяющих удовлетворять эту потребность.

Методы глубокого обучения, такие как классификация частотных эмбеддингов на основе полносвязных сетей не позволяют добиться достаточной точности, а трансформеры работают слишком долго. Интеграция графовых нейронных сетей и создание эмбеддингов при помощи графов позволяет сочетать точность трансформеров при помощи учёта семантического сходства между токенами и скорость как обучения, так и применения на практике.

**Цель исследования -** разработать и экспериментально оценить различные методы классификации фрагментов программного кода, включающие в себя традиционные векторные представления, контекстные эмбеддинги трансформеров и различные графовые алгоритмы и нейронные сети.

**Задачи исследования.**

1. Провести обзор существующих методов классификации текстов и программных фрагментов, охватывающий традиционные статистические подходы, модели на основе трансформеров, алгоритмы построения графов и графовые нейронные сети.
2. Выбрать и подготовить датасет фрагментов кода с соответствующими метками языков программирования для обучения и тестирования моделей.
3. Реализовать базовый метод классификации на основе Word2Vec и полносвязной нейронной сети.
4. Построить графовую модель на основе трансформерных эмбеддингов и обучить графовую нейронную сеть.
5. Реализовать метод с использованием эмбеддингов CodeBERT и классификатора на их основе.
6. Построить граф токенов на основе TF-IDF, отбора по дисперсии и косинусного сходства, получить эмбеддинги с помощью Node2Vec и обучить классификатор.
7. Провести сравнительный эксперимент по метрикам качества, времени обучения и предсказания для всех моделей.
8. Выявить достоинства и ограничения различных подходов к классификации программного кода.

**Объект исследования** - датасет фрагментов исходного кода с метками языков программирования, используемый для обучения и оценки моделей классификации.

**Предмет исследования** - методы представления фрагментов кода в векторной форме, способы построения и использования графов для извлечения семантических признаков, а также алгоритмы их последующей классификации.

# Глава 1. Обзор современных подходов к анализу и классификации программного кода

**1.1. Постановка задачи классификации текстов**

Классификация текстов — одна из ключевых задач в области обработки естественного языка (NLP). Суть этой задачи заключается в том, чтобы отнести текст или его фрагмент к одной из заранее заданных категорий. Подобные методы применяются в самых разных сферах: от фильтрации спама и анализа отзывов до систем модерации и автоматической сортировки документов. В программной инженерии задачи классификации тоже встречаются часто — например, при определении языка программирования, распознавании назначения скриптов или тематическом разбиении проектов.

В данной работе рассматривается задача классификации языка программирования по программному коду или его фрагменту. В отличие от человеческого языка, язык программирования, как правило, обладает более строгой структурой, но меньшим лексическим разнообразием. Кроме того, синтаксис разных языков может пересекаться — например, C, C++ и C# во многом похожи. Также возникают проблемы с гибридными конструкциями, где один язык вложен в другой, например, JavaScript внутри C++. Такие случаи особенно трудны для автоматической классификации, так как одних ключевых слов или символов может быть недостаточно.

Многие существующие подходы к этой задаче основаны на шаблонах — например, они используют регулярные выражения или просто считают, как часто встречаются те или иные токены. Но такие методы не всегда справляются, даже крупные сервисы, вроде Telegram или GitHub, иногда ошибаются при определении языка. Основная проблема в том, что подобные алгоритмы не учитывают смысловые связи и зависимость между частями текста.

Более современные подходы используют векторное представление текста — эмбеддинги, но и здесь часто теряется информация о внутренней структуре кода. Один из способов решить эту проблему — представить фрагмент кода в виде графа, где узлы — это, например, токены или целые конструкции, а рёбра показывают связи между ними. На основе таких графов можно генерировать эмбеддинги, на основе которых потом строится классификатор, либо же сразу применять графовые нейронные сети (GNN), которые учитывают структуру при обучении.

Применение гибридных архитектур, сочетающих графовые представления с трансформерными моделями, демонстрирует высокую эффективность при анализе программного кода. В частности, в работе «GraphCodeBERT: Pre-training Code Representations with Data Flow» (Guo и др., 2021) предложена модель, которая объединяет эмбеддинги исходного кода с графом потока данных (Data Flow Graph). Такой подход позволяет учитывать как поверхностную структуру текста, так и глубокие семантические зависимости между элементами программы, что существенно повышает качество решения задач классификации и генерации. Полученные результаты показывают, что использование информации, полученной из графов, в сочетании с контекстными представлениями даёт заметный прирост точности по сравнению с исключительно текстовыми методами.

Таким образом, задача классификации кода требует подходов, которые умеют не только «понимать» текст, но и учитывать его структуру и контекст. В этой работе рассматриваются и сравниваются различные методы — от классических векторных представлений до графовых моделей и современных трансформеров.

**1.2. Классические методы представления текстов**

До появления современных моделей глубокого обучения задача представления текста в числовой форме решалась с помощью сравнительно простых методов. Основная цель заключалась в преобразовании текста в вектор признаков, на основе которого можно было обучить классификатор. В этом разделе рассмотрим наиболее популярные классические подходы:

* **Мешок слов (Bag-of-Words, BoW)** — один из самых ранних и широко используемых методов. Текст представляется в виде вектора, где каждая позиция соответствует конкретному слову из словаря, а значение — количеству его появлений в тексте (или просто 0/1, если используется бинарная схема). Этот подход не учитывает порядок слов и контекст, но при больших объёмах данных может давать неплохие результаты.
* **TF-IDF (Term Frequency–Inverse Document Frequency)** — модификация BoW, при которой учитывается не только частота слова в конкретном тексте, но и то, насколько оно редкое или типичное для корпуса в целом. Это позволяет снизить влияние "шумовых" или часто встречающихся слов, делая признаки более информативными. Тем не менее, и TF-IDF не способен уловить смысл текста или отношения между словами.
* **Регулярные выражения (RegEx)** — хотя они не являются методом векторизации, их часто использовали для предварительной фильтрации и извлечения признаков, особенно при анализе кода. Например, можно выделить конструкции типа for (...), if (...) или шаблоны комментариев, которые встречаются только в определённых языках. Однако такой подход требует ручной настройки и плохо масштабируется.
* **Метод k-ближайших соседей (k-NN)** — один из самых простых алгоритмов классификации, который часто применялся в паре с вышеописанными представлениями. Он не требует обучения в обычном смысле: классификация происходит на основе сходства между векторами нового текста и примерами из обучающей выборки. На практике k-NN может быть весьма медленным при большом объёме данных и чувствителен к размерности признакового пространства, но его часто используют как базовую модель для сравнения.

Эти методы применялись не только в задачах анализа обычных текстов, но и при работе с программным кодом. В частности, элементы кода интерпретировались как "слова", и на основе их частот строились BoW или TF-IDF векторы. N-граммы, особенно символьные, использовались для моделирования локальных шаблонов. Однако у таких подходов есть серьёзные ограничения: они плохо работают на коротких или неполных сниппетах, не учитывают смысловую близость между элементами и не способны различать семантически схожие, но синтаксически разные фрагменты. В связи с этим на смену классическим методам пришли модели, способные учитывать контекст, структуру и семантику.

**1.3. Модели на основе эмбеддингов и трансформеров**

Одним из ключевых этапов в развитии методов обработки текста стало появление плотных векторных представлений – эмбеддингов. **В отличие от частотных признаков, эмбеддинги представляют слова, символы или другие единицы текста в виде векторов фиксированной размерности, обученных таким образом, что семантически близкие элементы оказываются ближе друг к другу в пространстве признаков, чем семантически далёкие.**

Ранние модели, такие как Word2Vec, GloVe и FastText, обучаются на больших корпусах текстов и используют статистику совместных появлений слов в контексте. Эти методы позволяют улавливать некоторые смысловые отношения между различными элементами текста, однако они присваивают каждому элементу одно фиксированное представление, независимо от конкретного контекста, в котором оно встречается. Такое фиксированное представление может быть серьёзным ограничением для анализа кода, поскольку идентификаторы, операторы и ключевые слова могут использоваться в разных языках программирования или в различных конструкциях с различным значением.

Качество представлений значительно улучшилось благодаря появлению контекстных моделей, таких как BERT (Bidirectional Encoder Representations from Transformers). Эти модели основаны на архитектуре трансформеров и обучаются так, чтобы эмбеддинг токена зависел от его окружения. Таким образом, представление одного и того же токена может отличаться в зависимости от контекста, что особенно важно для кода, где лексическая неоднозначность намного выше, чем в текстах на естественном языке.

Одной из первых успешных моделей для работы с программным кодом стала «CodeBERT: A Pre-Trained Model for Programming and Natural Languages» (Feng et и др., 2020), в которой была реализована идея обучения на параллельных корпусах программ и текстов на естественном языке. Авторы показали, что такая предобученная модель способна успешно решать задачи классификации, сопоставления и генерации кода, существенно превосходя традиционные архитектуры без учёта семантики.

Однако, несмотря на высокую точность, трансформерные модели имеют ряд ограничений. Основное из них — значительные вычислительные затраты, особенно при необходимости обработки большого объёма данных или при дообучении модели на новой выборке. Эта проблема критична в условиях ограниченных ресурсов или при работе в реальном времени с большим потоком данных.

**1.4. Графовые представления текстов и кода**

Другое направление в моделировании текстов — применение графовых структур. В графах узлы соответствуют элементам текста: символам, словам, предложениям или другим смысловым единицам. Рёбра отражают связи между ними — семантические, синтаксические или статистические. Такой подход позволяет моделировать внутреннюю структуру текста более гибко, чем линейные или позиционные представления.

Для программного кода графовое представление может отражать как поверхностные связи между токенами, так и более сложные зависимости (например, связи между переменной и её использованием в другом месте кода). На основе эмбеддингов можно вычислить семантическое сходство между токенами и построить граф, где рёбра соединяют те элементы, которые находятся в определённой метрике близости. Также можно использовать статистические методы (такие как TF-IDF или выбор по дисперсии) для предварительной фильтрации признаков, после чего строится граф, отражающий их взаимосвязи.

Полученные графы могут быть использованы в сочетании с графовыми нейронными сетями (Graph Neural Networks, GNN), которые позволяют создавать модели, учитывающие как локальные, так и глобальные зависимости между узлами. В отличие от традиционных нейросетей, GNN агрегируют информацию от соседних узлов, постепенно обновляя представления каждого узла. Самыми популярными архитектурами являются **GraphSAGE**, в котором обновление признаков осуществляется за счёт агрегации информации от случайной подвыборки соседей, и **GAT (Graph Attention Network),** где используется механизм внимания для взвешивания вклада соседей при передаче информации. Эти методы позволяют учитывать неоднородность связей и более точно моделировать значимость различных элементов графа.

Концепция построения представлений кода на основе графов получила развитие в ряде исследований. В работе «Learning to Represent Programs with Graphs» (Allamanis и др., 2018) показано, что включение множества видов рёбер в графовую модель позволяет достичь высокой точности при анализе программ. В продолжение этой идеи, в работе «Global Relational Models of Source Code» (Hellendoorn и др., 2021) предложена архитектура, в которой используется глобальное внимание по различным типам связей в коде — как синтаксическим, так и семантическим. Авторы демонстрируют значительное улучшение по сравнению с базовыми моделями на ряде задач, включая классификацию.

Графовые структуры особенно эффективны для анализа кода, где важную роль играют не только лексические элементы, но и их взаимное расположение и контекст. В отличие от естественного языка, где семантика сильно зависит от линейного порядка слов, в коде связи определяются структурой (отступами, вложенностью) и неявными зависимостями (например, использованием переменной в другой части кода). Например, блоки кода, относящиеся к одной логической конструкции (условия, циклы, вызовы функций), могут быть разделены другими фрагментами, но при этом сохранять смысловую целостность. Графовая модель позволяет явно представить эти связи и использовать их при обучении. В задачах классификации программного кода графовое представление может повысить устойчивость модели к нестандартному форматированию, отсутствию контекста и другим источникам шума, с которыми плохо справляются классические модели.

# Глава 2. Теоретическое описание реализованных подходов

### 2.1 Классификация кода с использованием Word2Vec и полносвязной нейронной сети

В рассматриваемой задаче необходимо построить модель, способную определять язык программирования по входному фрагменту кода. Поскольку исходные данные представлены в виде текстовых строк, в первую очередь необходимо преобразовать их в числовое пространство, пригодное для последующей обработки алгоритмами машинного обучения. Это достигается путём векторизации: код преобразуется в последовательность токенов, для которых далее строятся плотные векторные представления. Финальная классификация осуществляется на основе агрегированных эмбеддингов с помощью нейронной сети, либо же при помощи графа, построенного на их основе.

На вход подаётся набор программных фрагментов , каждый из которых содержит строковое представление кода. Первым этапом обработки является предобработка и токенизация. Предобработка включает удаление всех комментариев (однострочных и многострочных), а также строковых литералов. Это позволяет исключить из рассмотрения неинформативные фрагменты текста, не связанные напрямую с синтаксисом языка.

Далее применяется токенизация, которая разбивает фрагмент на последовательность лексем — идентификаторов, ключевых слов, операторов, чисел и скобок. Обозначим полученную последовательность как:

где — словарь токенов, сформированный по всей обучающей выборке.

В первом подходе для представления токенов в числовом виде используется модель Word2Vec, обучаемая на объединённой выборке токенизированных фрагментов. Используется архитектура Skip-gram с окном контекста шириной k, задача которой — предсказывать вероятности появления соседних токенов по текущему:

где — отображение токенов в пространство эмбеддингов размерности d.

После обучения модели каждый токен t получает вектор , а весь фрагмент представляется как усреднение векторов своих токенов:

На следующем этапе к вектору применяется полносвязная нейросеть, состоящая из трёх линейных слоёв с промежуточной активацией ReLU и регуляризацией Dropout:

где — параметры модели. Результатом является вектор , в котором каждая компонента соответствует вероятности принадлежности фрагмента к одному из K языков программирования.

Для обучения используется функция потерь кросс-энтропии:

где — истинная метка класса, представленная в one-hot виде.

Данный подход реализует классическую схему преобразования текста в эмбеддинги и последующей классификации. Он служит базовой отправной точкой, на основе которой далее сравниваются более сложные архитектуры, использующие глубинные языковые модели и графовые структуры.

**2.2 Классификация с использованием CodeBERT и графовой нейронной сети**

Во втором подходе для извлечения векторных представлений программ используется предобученная трансформерная модель CodeBERT, способная обрабатывать естественный язык, но преимущественно предназначенная для программного языка. В отличие от Word2Vec, который строит локальные эмбеддинги на уровне токенов, CodeBERT возвращает контекстно-зависимые представления, учитывающие полную структуру входного текста.

Для каждого фрагмента кода изначально применяется токенизация согласно протоколу CodeBERT (на основе BPE), после чего извлекается скрытое представление [CLS]-токена:

где d = 768 — размерность эмбеддингов в модели CodeBERT-base.

Далее на полученном множестве векторов строится графовая структура. Используется k-ближайших соседей в евклидовом пространстве эмбеддингов для формирования рёбер между вершинами, каждая из которых соответствует отдельному фрагменту кода:

Пусть G = (V, E) — построенный ненаправленный граф. На этом графе обучается одна из двух моделей: GraphSAGE или GAT (Graph Attention Network).

Модель GraphSAGE агрегирует информацию от соседей вершины i по формуле:

где , а AGG — агрегирующая функция (например, среднее). В GAT вместо усреднения используется взвешенное суммирование с обучаемыми коэффициентами внимания :

Модель обучается на задачу многоклассовой классификации с кросс-энтропийной функцией потерь, аналогичной использованной в предыдущем методе.

Такой подход позволяет использовать мощные контекстуальные эмбеддинги и дополнительно учитывать структуру распределения данных за счёт графовых связей между объектами.

**2.3 Классификация на основе CodeBERT и полносвязной нейросети**

В третьем методе используется тот же механизм извлечения признаков, что и в предыдущем пункте — контекстные эмбеддинги с помощью модели CodeBERT, но без построения графа. Вместо графовой архитектуры на полученных эмбеддингах обучается простая полносвязная нейронная сеть.

Для каждого фрагмента кода извлекается эмбеддинг [CLS]-токена:

Далее применяется архитектура нейронной сети с тремя слоями, аналогично использованной в пункте 2.1. Такая модель реализует классический пайплайн "эмбеддер + классификатор":

Обучение проводится с использованием функции потерь:

Такой подход проще в реализации по сравнению с графовой моделью, но сохраняет высокую точность благодаря мощному языковому представлению, предоставляемому CodeBERT.

### 2.4 Классификация на основе TF-IDF по символьным n-граммам и графовой агрегации через Node2Vec

Разработанный метод призван объединить преимущества классических подходов и современных моделей представления кода. Он быстрее трансформеров, поскольку не использует BERT-подобные модели на этапе кодирования, но сохраняет высокую выразительность признаков за счёт графового представления лексических элементов и обучения узловых эмбеддингов. Метод особенно хорошо подходит для задач, где важна скорость, но при этом требуется учитывать семантическую и структурную связанность элементов кода.

Метод включает несколько этапов: символьный TF-IDF, фильтрацию признаков по дисперсии, построение графа сходства между признаками на основе косинусной близости, обучение узловых эмбеддингов с помощью **Node2Vec**, а также классификацию итоговых представлений при помощи глубокой полносвязной нейронной сети с остаточными связями.

Каждому фрагменту кода сопоставляется вектор признаков TF-IDF, основанный на символьных n-граммах (анализ по символам, без приведения к нижнему регистру). Компоненты TF-IDF рассчитываются по классической формуле:

Здесь — частота появления n-граммы j в документе i, — количество документов, содержащих n-грамму j,

После этого из TF-IDF-матрицы исключаются признаки с низкой дисперсией:

Из оставшихся признаков строится неориентированный взвешенный граф G = (V, E), где каждая вершина v ∈ V — это отдельная n-грамма, а рёбра соответствуют высокому косинусному сходству между признаками:

В граф добавляются рёбра между признаками и , если сходство превышает определённый порог , заданный как:

также реализованы **дропаут рёбер** (для регуляризации), **добавление случайных рёбер** (для повышения связности графа), **аугментация** графа дополнительными лексическими связями (при включении соответствующей опции).

На полученном графе применяется **Node2Vec** — стохастический метод обучения представлений узлов, который использует случайные блуждания по графу и модель Skip-gram. Основная идея заключается в том, что вершины, часто встречающиеся рядом в случайных путях, получают похожие векторы:

где:

— искомое эмбеддинг-представление вершины v,

— множество узлов, встречающихся в контексте вершины v во время случайных блужданий,

— вероятность появления узла u в окружении v, моделируемая через softmax:

Для сглаживания распределения эмбеддингов дополнительно применяется преобразование:

Таким образом, каждая n-грамма получает числовое представление, в котором зашифрованы как её частотные характеристики, так и структура её взаимосвязей с другими терминами.

Полученные эмбеддинги далее агрегируются по документу и поступают в классификатор, реализованный в виде полносвязной нейросети с остаточными связями и Batch Normalization. Архитектура сети включает три скрытых слоя с функцией активации GELU и Dropout:

В данной главе были рассмотрены четыре метода классификации программного кода. В качестве базового подхода использовалось векторное представление Word2Vec, после чего был предложен оригинальный метод на основе TF-IDF, графов сходства и алгоритма Node2Vec. Каждый последующий метод представляет собой усложнение модели, направленное на повышение точности распознавания языков программирования при сохранении вычислительной эффективности. В следующей главе будет представлена реализация рассмотренных подходов, а также проведен сравнительный анализ их результатов.

# Глава 3. Практическая реализация и эксперименты

### 3.1 Датасет и подготовка данных

В качестве исходных данных использовался датасет, содержащий **45 628** фрагментов кода, размеченных по **15 языкам программирования**. Примеры включают популярные языки: Python, Java, C++, JavaScript и др. Формат каждого примера — пара вида:

{"code": "...", "language": "Python"}

Датасет был разбит на обучающую, валидационную и тестовую выборки в пропорции 75/20/5. Ниже приведён график распределения классов (см. Рис. 1).

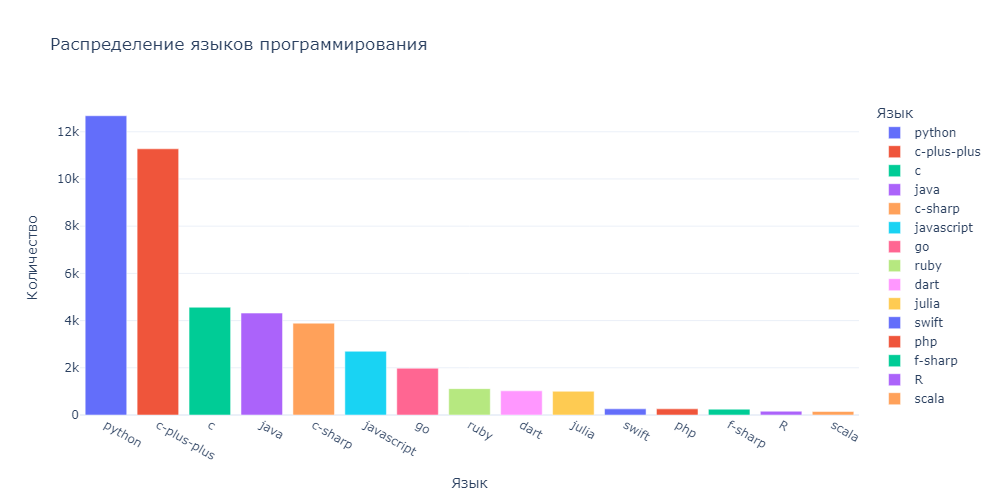


Рис. 1 - Распределение примеров по языкам программирования

Поскольку распределение несбалансированное, для оценки моделей использовалась F1-мера (гармоническое среднее точности и полноты)

### 3.2 Модель на основе Word2Vec и полносвязной нейросети

Первый реализованный подход основан на обучении модели Word2Vec на предобработанных при помощи регулярных выражений и токенизированных фрагментах кода. Модель Word2Vec обучалась с помощью библиотеки genism.

**def** preprocess\_code(code: str) **->** str:

    code **=** re.sub(r'(".\*?")|(//.\*?$|#.\*?$|--.\*?$)', **lambda** m: m.group(1) **or** "", code, flags**=**re.MULTILINE)

    code **=** re.sub(r'(""".\*?"""|\'\'\'.\*?\'\'\'|".\*?"|\'.\*?\')|/\\*.\*?\\*/', **lambda** m: m.group(1) **or** "", code, flags**=**re.DOTALL)

**return** code

**def** tokenize\_code(code: str) **->** list[str]:

    processed **=** preprocess\_code(code)

    tokens **=** re.findall(r'\"\"\".\*?\"\"\"|\'\'\'.\*?\'\'\'|"(?:\\.|[^\\"])\*"|\'(?:\\.|[^\\\'])\*\'|\b[\d]+(?:\.[\d]\*)?\b|\b[a-zA-Z\_][\w]\*\b|::|==|!=|<=|>=|[+/\*\-=%&|^<>!~]|[(),.;:{}\[\]`]', processed)

**return** [t.strip() **for** t **in** tokens **if** t.strip()]

model **=** Word2Vec(sentences**=**train\_tokens **+** val\_tokens **+** test\_tokens, vector\_size**=**100, window**=**5, min\_count**=**1, workers**=**12)

После обучения для каждого фрагмента кода строился усреднённый вектор на основе токенов, присутствующих в словаре модели.

Для классификации использовалась простая полносвязная сеть с двумя скрытыми слоями:

**class** FCembed(nn.Module):

**def** \_\_init\_\_(self, input\_dim, hidden\_dim\_0, hidden\_dim\_1, output\_dim, dropout\_rate=0.13):

        super().\_\_init\_\_()

        self.net **=** nn.Sequential(

            nn.Linear(input\_dim, hidden\_dim\_0),

            nn.ReLU(),

            nn.Dropout(dropout\_rate),

            nn.Linear(hidden\_dim\_0, hidden\_dim\_1),

            nn.ReLU(),

            nn.Dropout(dropout\_rate),

            nn.Linear(hidden\_dim\_1, output\_dim)

        )

**def** forward(self, x):

**return** self.net(x)

Обучение велось с использованием CrossEntropyLoss и оптимизатора AdamW. В процессе обучения фиксировалась F1-мера на тренировочной и валидационной выборке. На тестовой выборке F1-мера достигла 0.75, время обучения – 1:21 мин., время выполнения предсказания 0.001 сек. График сходимости приведен на Рис. 2.

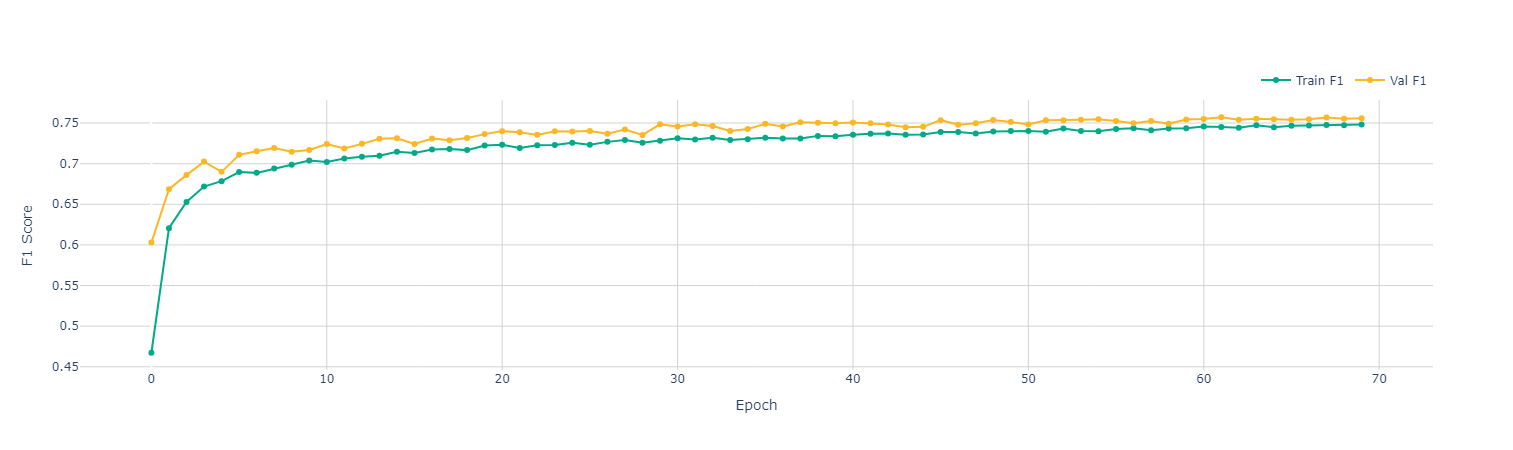


Рис. 2 - График F1-меры модели (Word2Vec + FC)

### 3.3 Модель с CodeBERT-эмбеддингами и графовой нейросетью

Во втором подходе для векторизации кода использовались эмбеддинги модели **CodeBERT**. Токенизация и извлечение эмбеддингов выполнялись с помощью библиотеки transformers:

tokenizer **=** AutoTokenizer.from\_pretrained("microsoft/codebert-base")

emb\_model **=** AutoModel.from\_pretrained("microsoft/codebert-base")

Для построения графа на основе эмбеддингов использовался алгоритм **k-ближайших соседей** (k-NN), где каждая вершина — это фрагмент кода, а рёбра соединяют k ближайших фрагментов по косинусной близости.

knn **=** NearestNeighbors(n\_neighbors**=**k**+**1).fit(all\_embeddings)

\_, indices **=** knn.kneighbors(all\_embeddings)

edge\_index **=** torch.tensor(

    [[i, j] **for** i **in** range(len(indices)) **for** j **in** indices[i] **if** i **!=** j],

    dtype**=**torch.long).t().contiguous()

Были реализованы две графовые архитектуры: **GraphSAGE** и **GAT**. GraphSAGE:

**class** GraphSAGEClassifier(nn.Module):

**def** \_\_init\_\_(self, in\_dim, hidden\_dim, out\_dim, num\_layers=3):

        super().\_\_init\_\_()

        self.convs **=** nn.ModuleList([

            SAGEConv(in\_dim **if** i **==** 0 **else** hidden\_dim, hidden\_dim)

**for** i **in** range(num\_layers)

        ])

        self.jump **=** JumpingKnowledge(mode**=**'cat')

        self.fc **=** nn.Linear(hidden\_dim **\*** num\_layers, out\_dim)

**def** forward(self, data):

        x, edge\_index **=** data.x, data.edge\_index

        xs **=** []

**for** conv **in** self.convs:

            x **=** F.relu(conv(x, edge\_index))

            x **=** F.dropout(x, p**=**0.13, training**=**self.training)

            xs.append(x)

        x **=** self.jump(xs)

**return** F.log\_softmax(self.fc(x), dim**=**1)

И аналогичная реализация GAT-модели:

**class** GATClassifier(nn.Module):

**def** \_\_init\_\_(self, in\_dim, hidden\_dim, out\_dim, heads=8):

        super().\_\_init\_\_()

        self.conv1 **=** GATConv(in\_dim, hidden\_dim, heads**=**heads)

        self.conv2 **=** GATConv(hidden\_dim **\*** heads, out\_dim, heads**=**1)

**def** forward(self, data):

        x, edge\_index **=** data.x, data.edge\_index

        x **=** F.elu(self.conv1(x, edge\_index))

        x **=** F.dropout(x, p**=**0.15, training**=**self.training)

        x **=** self.conv2(x, edge\_index)

**return** F.log\_softmax(x, dim**=**1)

**CodeBERT + kNN + GraphSAGE:** F1 = 0.79, время обучения – 4:37 мин., время предсказания – 0.014 сек.

**CodeBERT + kNN + GAT:** F1 = 0.70, время обучения – 6:50 мин., время предсказания – 0.025 сек.

Графики сходимости приведены на Рис. 3 и Рис. 4 соответственно.

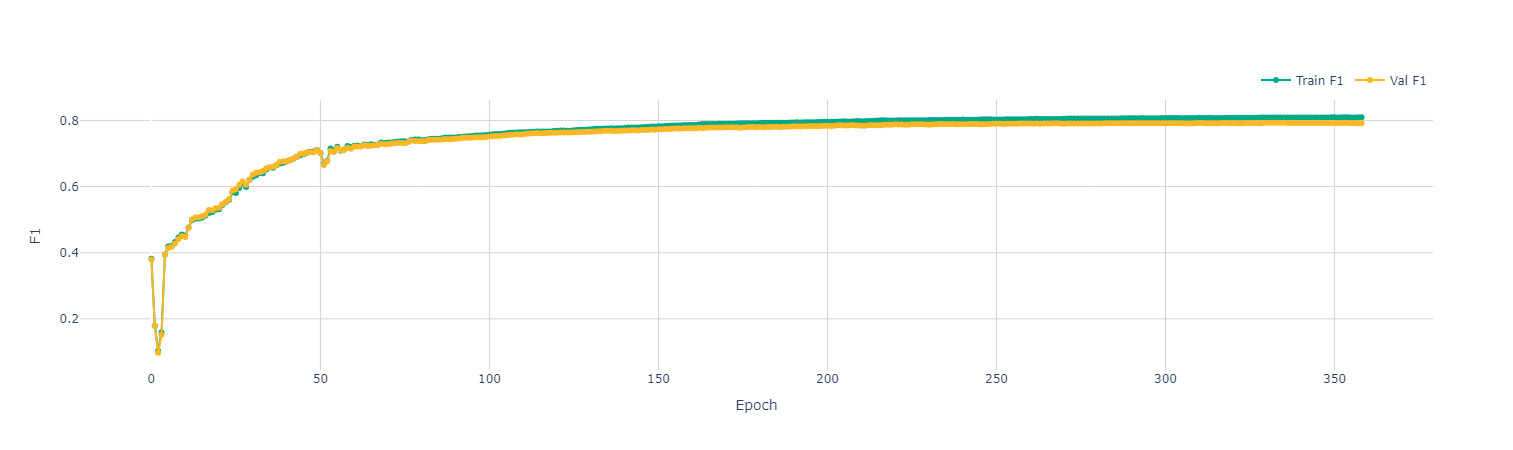


Рис. 3 – График F1-меры модели (SAGEConv)

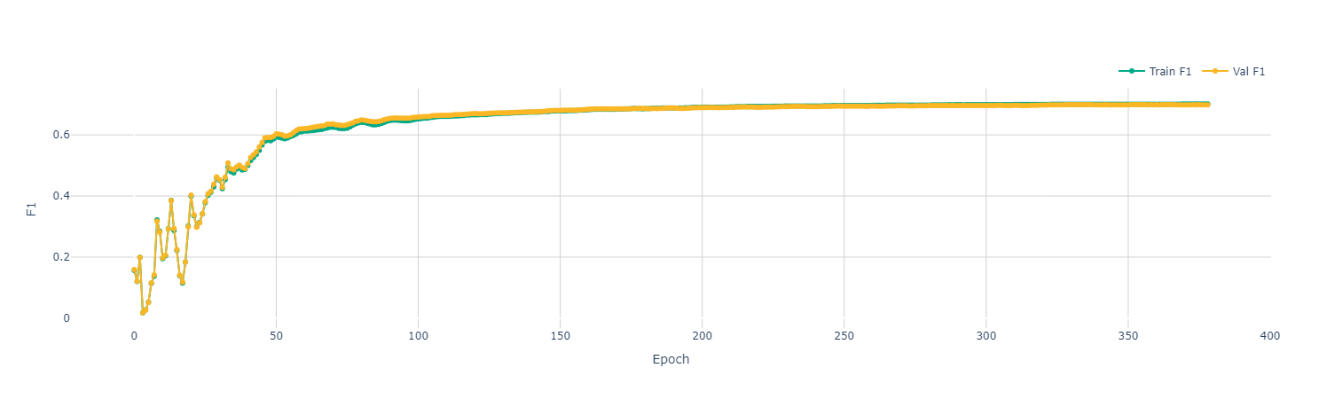


Рис. 4 – График F1-меры модели (GAT)

### 3.4 Модель CodeBERT + полносвязная сеть

В третьем эксперименте классификация CodeBERT-эмбеддингов проводилась напрямую с использованием полносвязной нейронной сети, в отличие от предыдущего подхода, без построения дополнительных графов (см. п. 3.2).

Входом в модель служили эмбеддинги длины 768, сеть состояла из двух скрытых слоёв и dropout-слоя.

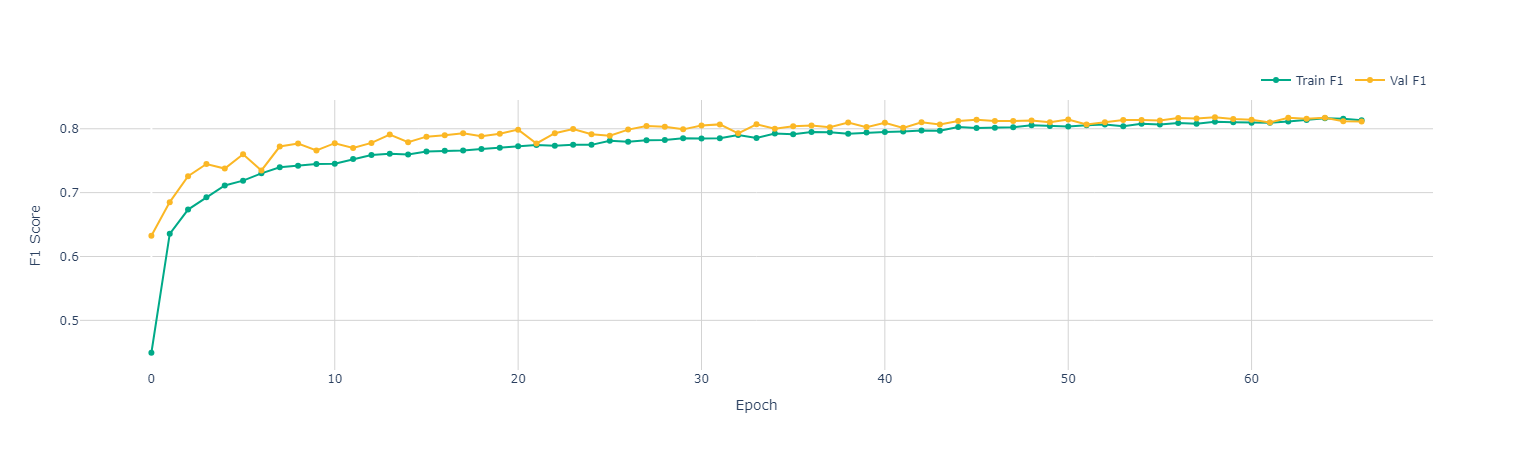
F1-мера достигла 0.82, время обучения - 4:36 мин., время предсказания – 0.021 сек. График обучения приведён на Рис. 5

Рис. 5 – График F1-меры модели (CodeBERT + FC)

### 3.5 Авторский метод: TF-IDF + Node2Vec + FC

В четвёртом эксперименте был реализован собственный подход к векторному представлению кода и классификации. Алгоритм включает несколько этапов:

1. **Векторизация кода** с помощью TF-IDF по char-n-граммам, отбор признаков по дисперсии:

self.vectorizer **=** TfidfVectorizer(analyzer**=**'char', lowercase**=False**)

tfidf **=** self.vectorizer.fit\_transform(texts)

tfidf\_selected **=** VarianceThreshold(...).fit\_transform(tfidf)

2. Построение графа по косинусной близости между признаками:

cossim **=** cosine\_similarity(tfidf\_selected.T)

**if** cossim[i, j] **>** threshold:

    G.add\_edge(feature\_i, feature\_j, weight**=**cossim[i, j])

3. Регуляризация графа: дропаут рёбер, добавление случайных связей, аугментация весов.

4. Обучение Node2Vec на построенном графе:

model **=** Node2Vec(G, dimensions**=**128, ...).fit()

embeddings **=** {node: model.wv[node] **for** node **in** G.nodes}

5. Получение эмбеддингов документов: взвешенное усреднение эмбеддингов по TF-IDF:

tfidf\_selected.dot(embedding\_matrix.T)

Для классификации использовалась глубокая полносвязная сеть с остаточными связями:

**class** CodeLanguageClassifier(nn.Module):

**def** forward(self, x):

        x **=** self.fc1(x) **+** self.fc1\_residual(x)

        x **=** self.fc2(x) **+** self.fc2\_residual(x)

        x **=** self.fc3(x) **+** self.fc3\_residual(x)

**return** self.fc4(x)

Архитектура включала три скрытых слоя (1024, 2048, 1024 нейронов), GELU-активацию, BatchNorm и Dropout.

**Все ключевые гиперпараметры** метода — размерность эмбеддингов, порог косинусной близости, глубина сети и др. подбирались автоматически с использованием библиотеки **Optuna.**

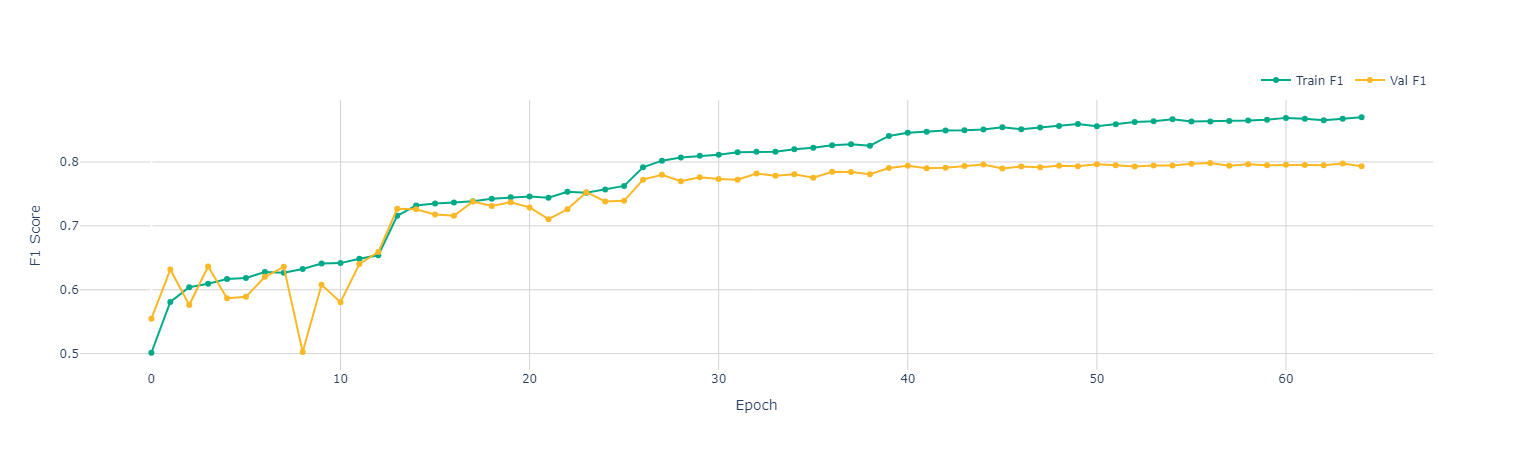
F1-мера: **0.81**, время обучения – **2:00** мин., время предсказания – **0.004** сек. График обучения представлен на Рис. 6.

Рис. 6 – График F1-меры модели (TF-IDF + Node2Vec + FC)

### 3.6 Сравнительный анализ и выводы

В ходе экспериментов были протестированы пять различных подходов к классификации программного кода, различающихся как по архитектуре, так и по методам представления входных данных. Сравнение проведено по трем основным метрикам: F1-мера, время обучения модели и время выполнения одного предсказания. Результаты сведены в таблицу (Таблица 1):

| **Модель** | **F1-мера** | **Время обучения** | **Время предсказания (сек)** |
| --- | --- | --- | --- |
| Word2Vec + FC | 0.75 | 1:21 | 0.001 |
| BERT + KNN + SAGEConv | 0.79 | 4:37 | 0.014 |
| BERT + KNN + GAT | 0.70 | 6:50 | 0.025 |
| BERT + FC | 0.82 | 4:36 | 0.021 |
| TF-IDF + Node2Vec + FC (авторский) | 0.81 | 2:00 | 0.004 |

Таблица 1

Базовая модель Word2Vec + FC продемонстрировала F1-меру = 0.75 при минимальном времени инференса (0.001 сек), что подтверждает её пригодность в качестве лёгкого базового подхода, но малую эффективность при работе с неоднозначными фрагментами кода.

Модели, сочетающие трансформерные эмбеддинги CodeBERT с графовыми нейронными сетями, показали смешанные результаты. Подход с использованием GraphSAGE (F1 = 0.79) оказался более устойчивым и качественным по сравнению с GAT (F1 = 0.70), что, вероятно, связано с переусложнённой архитектурой последнего и высокой чувствительностью к шуму. **Хотя оба графовых подхода уступают по эффективности CodeBERT + полносвязному классификатору, их применение даёт стабильный прирост качества по сравнению с базовыми методами. Это подтверждает перспективность использования графовых структур в задачах, где важны семантические связи между фрагментами текста, однако текущая реализация требует оптимизации для снижения вычислительных затрат и качества модели.**

Наилучшее значение F1-меры (0.82) показала модель, основанная на эмбеддингах CodeBERT и полносвязной нейросети, что подтверждает высокую эффективность трансформерных моделей даже без использования графов. Однако при этом она значительно уступает по скорости предсказания и требует значительного времени обучения.

Наиболее сбалансированный результат продемонстрировал авторский метод, сочетающий TF-IDF, отбор признаков по дисперсии, создание графа на основе косинусной близости, агрегацию через Node2Vec и глубокую нейросетевую классификацию. При значении F1-меры 0.81 он почти не уступает лучшему результату, но при этом **существенно превосходит альтернативы по скорости обработки**. Благодаря этому он может быть использован в системах с **высокой нагрузкой**, где требуется **минимизация времени отклика** при сохранении максимального качества предсказаний.

Таким образом, можно сделать вывод, что применение трансформеров в сочетании с графовыми структурами либо продвинутыми архитектурами может обеспечить высокую точность, но зачастую требует значительных ресурсов. В то же время предложенная комбинация методов (TF-IDF + Node2Vec) демонстрирует почти такие же показатели качества при существенно меньших требованиях к ресурсам. Такой подход может служить практичной альтернативой для задач, где важны как качество предсказаний, так и вычислительная эффективность.

# Заключение

В рамках данной курсовой работы была рассмотрена задача классификации программного кода по его фрагменту с применением методов глубокого и графового обучения. Целью исследования являлось сравнение различных подходов к представлению кода и построению классификаторов, сочетающих эффективность, точность и вычислительную целесообразность.

Для достижения поставленной цели были решены следующие задачи: проведён обзор существующих методов представления кода, реализованы и протестированы четыре различных подхода, включая классическую схему с использованием Word2Vec, контекстные эмбеддинги CodeBERT с двумя вариантами классификации (с графовой нейросетью и без неё), а также авторский метод на основе символьного TF-IDF, графа косинусного сходства и Node2Vec. Каждый из методов был экспериментально оценён по метрикам качества (F1-мера), времени обучения и скорости предсказания.

Результаты экспериментов подтвердили, что использование контекстных эмбеддингов CodeBERT в сочетании с простой полносвязной сетью позволяет достичь наилучшей точности (F1 = 0.82), но требует слишком больших затрат времени. Графовые модели, особенно при использовании GraphSAGE, показали себя как эффективное дополнение, особенно в случае ограниченного контекста и высокой семантической неоднозначности. Авторский метод с графами n-грамм и Node2Vec продемонстрировал хороший баланс между скоростью и качеством, что позволяет рекомендовать его для промышленного внедрения в условиях высоких нагрузок и ограниченных ресурсов.

Таким образом, работа продемонстрировала целесообразность гибридных подходов, учитывающих как семантические признаки текста, так и его структурные особенности. Это особенно важно при работе с реальными проектами, где классические методы часто не справляются с разнообразием стилей и шаблонов кода.

В дальнейшем возможна интеграция более сложных архитектур (например, моделей семейства CodeT5), а также расширение графовой информации за счёт синтаксических деревьев и анализа потока данных. Отдельного внимания заслуживает задача интерпретируемости моделей, особенно в контексте графов, где объяснение классификационного решения может быть полезным для анализа кода человеком.

Работа демонстрирует, что сочетание графовых представлений и современных языковых моделей позволяет достичь более точных и устойчивых результатов в классификации программного кода по сравнению с традиционными подходами.

# Список использованных источников

1. Алламанис М., Брокшайд Дж., Чен Ч., Саттон Ч. Learning to Represent Programs with Graphs // arXiv preprint arXiv:1711.00740, 2018. URL: <https://arxiv.org/abs/1711.00740>.
2. Гуо Д., Рен Ш., Лу С., Фэн З., Чэнь Д., Чжоу Л. GraphCodeBERT: Pre-training Code Representations with Data Flow // arXiv preprint arXiv:2009.08366, 2020. URL: <https://arxiv.org/abs/2009.08366>.
3. Фэн З. и др. CodeBERT: A Pre-Trained Model for Programming and Natural Languages // arXiv preprint arXiv:2002.08155, 2020. URL: <https://arxiv.org/abs/2002.08155>.
4. Hellendoorn V.J., Wang C., Tarlow D., Bird C. Global Relational Models of Source Code // Proceedings of the International Conference on Learning Representations (ICLR), 2021. URL: <https://arxiv.org/abs/2005.08926>.
5. Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. Distributed Representations of Words and Phrases and their Compositionality // Advances in Neural Information Processing Systems (NeurIPS), 2013. URL: <https://papers.nips.cc/paper_files/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html>.
6. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // arXiv preprint arXiv:1810.04805, 2018. URL: <https://arxiv.org/abs/1810.04805>.
7. Grover A., Leskovec J. Node2Vec: Scalable Feature Learning for Networks // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016. DOI: 10.1145/2939672.2939754.

# Приложение

### Ссылки на исходный код и материалы работы

* 1. Google Colab с Jupyter Notebook: <https://colab.research.google.com/drive/1iFukrhscz8QPg7XfpcAV94C6JrGKHwGp?usp=sharing>
  2. Github-репозиторий: <https://github.com/yaromirgusev/course_work_gml>