DOI 10.58351/2949-2041.2025.22.5.010

Чувашов Виктор Александрович,
магистрант, МГТУ им. Н.Э. Баумана, Москва
Chuvashov Viktor Alexandrovich, Bauman MSTU

Рудаков Игорь Владимирович,
заведующий кафедрой ИУ7, доцент, к.т.н.,
МГТУ им. Н.Э. Баумана, Москва
Rudakov Igor Vladimirovich, Bauman MSTU

ГИБРИДНАЯ МОДЕЛЬ CNN+T5 ДЛЯ ГЕНЕРАЦИИ ПРОГРАММНОГО КОДА: АРХИТЕКТУРА, РЕАЛИЗАЦИЯ И ПРОБЛЕМЫ ВНЕДРЕНИЯ
HYBRID CNN+T5 MODEL FOR SOFTWARE CODE GENERATION: ARCHITECTURE, IMPLEMENTATION AND IMPLEMENTATION PROBLEMS

Аннотация: В статье представлена гибридная модель генерации Python-кода, объединяющая CNN и трансформер T5 для захвата локальных и глобальных зависимостей текста. Описаны разработка, обучение и оценка модели с применением teacher forcing и планирования скорости обучения. Модель показывает улучшенные результаты по сравнению с базовыми подходами, обсуждаются архитектура, проблемы реализации и пути их решения.

Abstract: This paper presents a hybrid Python code generation model that combines CNN and T5 transformer to capture local and global text dependencies. We describe the development, training, and evaluation of the model using teacher forcing and learning rate scheduling. The model shows improved results compared to baseline approaches, and discusses the architecture, implementation issues, and solutions.

Ключевые слова: Генерация программного кода, Python, гибридная нейросетевая модель, CNN, трансформер T5, teacher forcing.

Keywords: Program code generation, Python, hybrid neural network model, CNN, T5 transformer, teacher forcing.

1. Введение

Автоматическая генерация программного кода является одной из наиболее перспективных областей применения современных методов искусственного интеллекта. Эта задача находится на пересечении обработки естественного языка и программной инженерии, представляя собой уникальный вызов: необходимо не только понять семантику запроса на естественном языке, но и преобразовать его в синтаксически корректный и функционально верный код [1].

В последние годы значительные успехи в этой области связаны с применением моделей на основе трансформеров, таких как CodeBERT, CodeT5 и Codex [2, 3]. Эти модели обладают способностью улавливать долгосрочные зависимости в данных, что критически важно для понимания структуры программного кода. Однако, несмотря на высокие результаты, трансформерные модели имеют ряд ограничений, в частности, недостаточное внимание к локальным паттернам и структурам.

Свёрточные нейронные сети (CNN), напротив, хорошо зарекомендовали себя в задачах, требующих анализа локальных паттернов [4]. В контексте генерации кода такие паттерны могут включать общепринятые идиомы программирования, стандартные конструкции и синтаксические элементы.

Целью данного исследования является разработка и оценка эффективности гибридной модели, объединяющей преимущества трансформеров (T5) и свёрточных нейронных сетей (CNN) для задачи генерации программного кода по описанию на естественном языке. Основной вклад работы заключается в:

1. Разработке оригинальной архитектуры CNN+T5, где CNN-компонент используется для улучшения представлений (эмбеддингов) токенов в трансформерной модели.
2. Применении техники teacher forcing для эффективного обучения и линейного планировщика скорости обучения с фазой разогрева.
3. Экспериментальной оценке эффективности предложенного подхода на наборе задач генерации Python-кода.

2. Обзор литературы

2.1 Трансформерные модели для генерации кода

Трансформерная архитектура, впервые представленная в работе Vaswani et al. [5], произвела революцию в области обработки естественного языка. В контексте генерации кода выделяются несколько значимых моделей:

— **CodeBERT** [2]: предобученная модель, основанная на архитектуре BERT, специализирующаяся на представлении программного кода различных языков программирования.

— **CodeT5** [3]: модель, основанная на архитектуре T5 (Text-to-Text Transfer Transformer), адаптированная для задач, связанных с кодом.

— **Codex** [6]: модель, построенная на базе GPT, специализированная для генерации кода.

Трансформерные модели используют механизм самовнимания (self-attention), который позволяет установить взаимосвязи между различными частями входных данных, независимо от их расстояния друг от друга в последовательности. Это особенно полезно для программного кода, где функциональные связи могут быть распределены по всему документу [7].

2.2 Применение CNN в задачах генерации текста

Свёрточные нейронные сети традиционно применялись в задачах компьютерного зрения, но также показали хорошие результаты в обработке естественного языка [8]. Ключевое преимущество CNN заключается в способности эффективно выделять локальные паттерны и иерархические структуры.

В контексте генерации текста и кода CNN могут быть использованы для:

— Извлечения n-граммных признаков [9]
— Распознавания синтаксических конструкций [10]
— Улучшения эмбеддингов слов или токенов [11]

2.3 Гибридные архитектуры в NLP

Гибридные архитектуры, объединяющие различные типы нейронных сетей, становятся все более популярными в задачах NLP. Комбинирование CNN и RNN показало значительное улучшение результатов в задачах классификации текста [12]. Подход, объединяющий CNN и трансформеры, был успешно применен в машинном переводе [13], но его потенциал в генерации кода остается недостаточно изученным.

2.4 Методы обучения и оптимизации

Технология teacher forcing, впервые предложенная в контексте рекуррентных нейронных сетей [14], является эффективной стратегией обучения моделей генерации последовательностей. Этот метод предполагает использование истинных меток (ground truth) в качестве входа на каждом временном шаге во время обучения, что упрощает процесс сходимости.

Планировщики скорости обучения (learning rate schedulers) с фазой разогрева (warmup phase) стали стандартным инструментом при обучении больших языковых моделей [15]. Подход linear warmup followed by linear decay позволяет модели сначала адаптироваться к данным, а затем постепенно уточнять параметры, что способствует более стабильному обучению.

3. Методология

3.1 Архитектура гибридной модели CNN+T5

Предложенная гибридная модель состоит из двух основных компонентов: трансформерной модели CodeT5 и специального CNN-модуля для улучшения эмбеддингов. Общая архитектура представлена на рисунке 1.

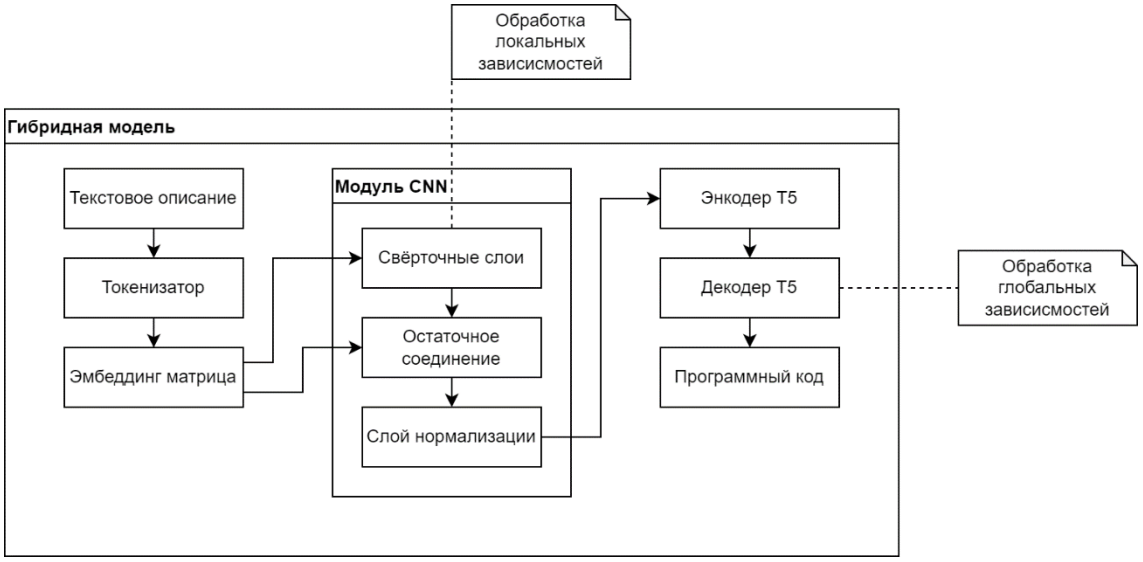

Рис. 1 – Общая архитектура гибридной модели CNN+T5

Ключевым компонентом предложенной архитектуры является модуль CNNEmbeddingEnhancer, который состоит из следующих элементов:

1. Входной слой, принимающий эмбеддинги из базовой модели T5
2. Серия свёрточных слоев (Conv1d) для обработки эмбеддингов
3. Остаточное соединение (residual connection), добавляющее обработанные CNN-признаки к исходным эмбеддингам
4. Слой нормализации (LayerNorm) для стабилизации обучения

Архитектура CNNEmbeddingEnhancer представлена на рисунке 2.

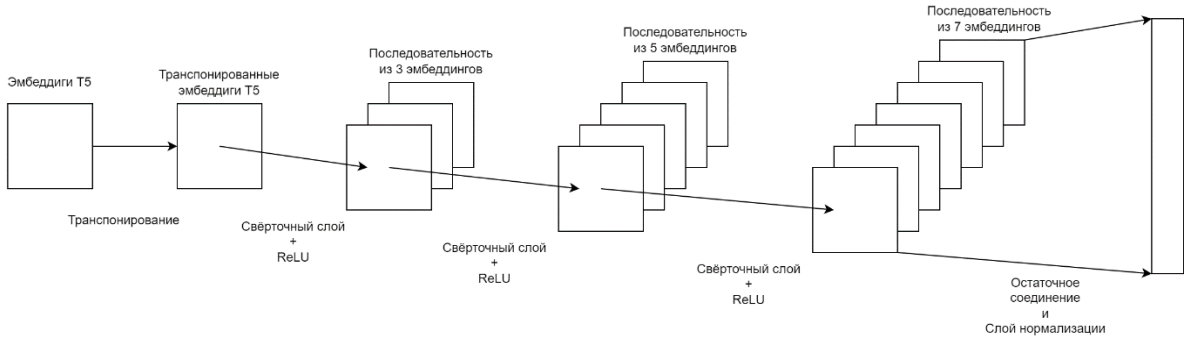

Рис. 2 – Архитектура модуля CNNEmbeddingEnhancer

Математически процесс обработки эмбеддингов можно описать следующим образом:

1. Получение эмбеддингов из базовой модели T5: E = Emb(X), где X - входные токены, Emb - функция эмбеддинга модели T5.
2. Преобразование для CNN-обработки: E_{CNN} = Transpose(E) - из [batch, seq_len, embed_dim] в [batch, embed_dim, seq_len].
3. Применение CNN: F_{CNN} = CNN(E_{CNN}), где CNN представляет серию свёрточных слоев.
4. Преобразование обратно: F = Transpose(F_{CNN}) - из [batch, embed_dim, seq_len] в [batch, seq_len, embed_dim].
5. Остаточное соединение и нормализация: E_{enhanced} = LayerNorm(E + F)

Полная гибридная модель CNNEnhancedT5 использует улучшенные эмбеддинги для генерации кода:

```
outputs = t5_model(
    inputs_embeds=enhanced_embeddings,
    attention_mask=attention_mask,
    labels=labels
)
```

3.2 Подготовка данных

Для обучения модели использовался набор данных из Kaggle (python-code-instruction-dataset), содержащий пары "инструкция-код". Данные были предобработаны следующим образом:

1. К каждой инструкции добавлен префикс "Generate Python code:"
2. Данные разделены на обучающую (90%) и валидационную (10%) выборки
3. Токены ограничены максимальной длиной 256 символов

3.3 Процесс обучения

Обучение модели проводилось с применением следующих подходов:

1. Teacher Forcing: на каждом шаге обучения модель получала истинные метки (ground truth) в качестве входных данных, что ускоряло процесс сходимости.
2. Оптимизатор AdamW: использование AdamW с параметром learning rate = 5e-5.
3. Смешанная точность (Mixed Precision): применение GradScaler для ускорения обучения на GPU.
4. Планировщик скорости обучения: использование линейного планировщика с фазой разогрева (10% от общего количества шагов).

Процесс обучения представлен на рисунке 3.

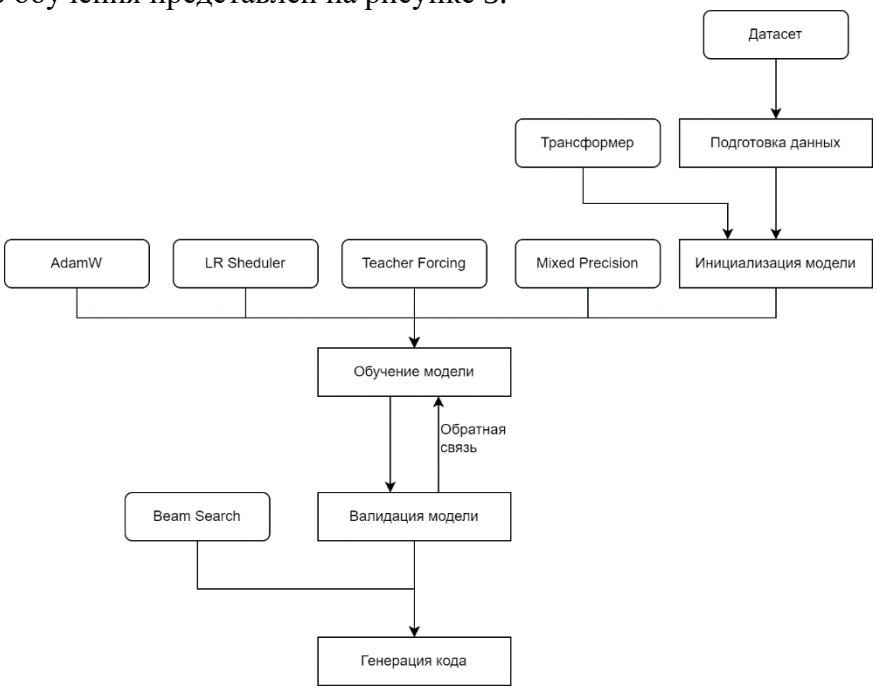

Рис. 3 – Схема процесса обучения модели

3.4 Генерация кода и оценка

Для генерации кода использовалась техника beam search с параметрами:

— Температура: 0.0 (детерминированная генерация)

— Число лучей (beams): 5

— no_repeat_ngram_size: 2

Эти параметры обеспечивают стабильность генерации и предотвращают повторение n-грамм в выходном коде.

4. Реализация и проблемы

4.1 Технические детали реализации

Гибридная модель была реализована с использованием PyTorch и библиотеки Hugging Face Transformers. Базовая модель CodeT5 ("Salesforce/codet5-base") была загружена и дополнена CNN-компонентом.

Основные технические характеристики реализации:

— Размер партии (batch size): 16

— Число эпох: 10

— Максимальная длина последовательности: 256

— Скорость обучения: 5e-5

— Общее количество параметров модели: 237,043,200

4.2 Проблемы и их решения

В процессе разработки и обучения модели возникли следующие проблемы:

1. **Проблема**: нестабильность обучения при объединении CNN и трансформерных компонентов. **Решение**: внедрение остаточных соединений (residual connections) и слоев нормализации (LayerNorm) для стабилизации потока градиентов.
2. **Проблема**: высокое потребление памяти GPU при большом размере партии. **Решение**: применение смешанной точности (mixed precision) через GradScaler и оптимизация размера партии.
3. **Проблема**: неэффективное использование вычислительных ресурсов при наличии нескольких GPU. **Решение**: реализация поддержки DataParallel для распределения вычислений на несколько GPU.
4. **Проблема**: низкое качество генерации на ранних этапах обучения. **Решение**: применение teacher forcing и тщательная настройка параметров beam search.
5. **Проблема**: медленная сходимость в начале обучения. **Решение**: Внедрение планировщика скорости обучения с фазой разогрева.

4.3 Диаграмма сущностей

На рисунке 4 представлена диаграмма основных сущностей системы.

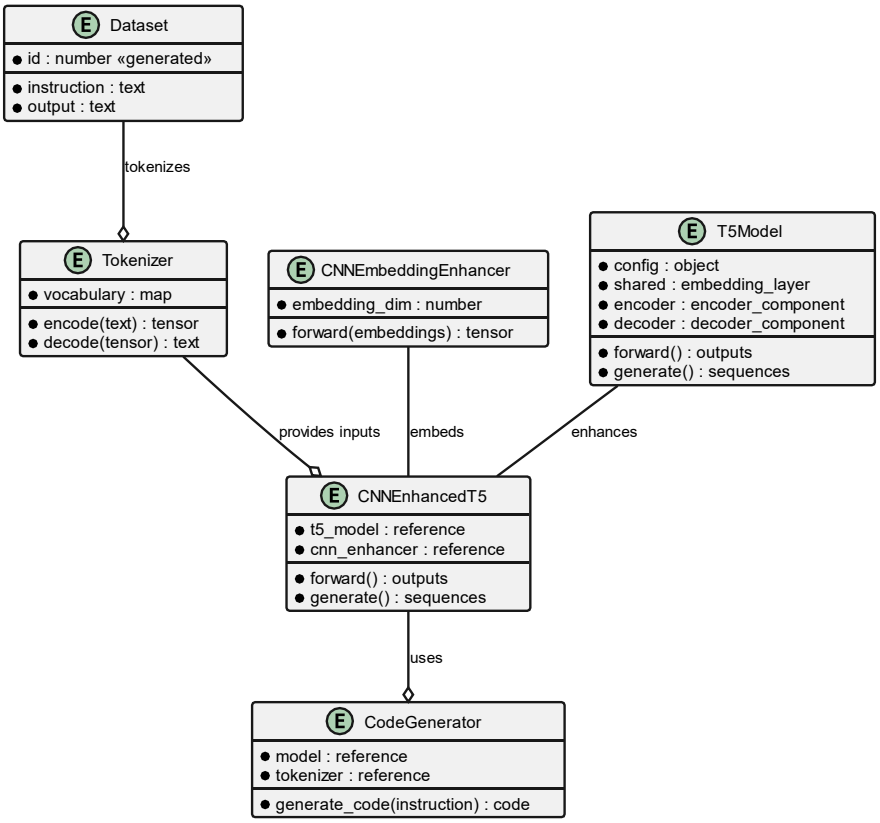

Рис. 4 – Диаграмма сущностей системы генерации кода

5. Результаты экспериментов

5.1 Динамика обучения

В ходе экспериментов модель обучалась на протяжении 10 эпох. Динамика потерь на обучающей и валидационной выборках представлена в таблице 1.

Таблица 1

Динамика потерь в процессе обучения

Эпоха	Train Loss	Val Loss	Время (с)
1	1.4162	0.6487	524.73
2	0.6720	0.5344	527.55
3	0.5752	0.4941	527.11
4	0.5237	0.4719	527.17
5	0.4873	0.4592	526.80
6	0.4602	0.4489	527.04
7	0.4389	0.4432	526.74
8	0.4227	0.4387	527.10
9	0.4102	0.4376	527.33
10	0.4019	0.4370	527.11

5.2 Примеры генерации

Ниже представлены примеры сгенерированных функций для различных задач после 10 эпох обучения:

Пример 1: проверка строки на палиндром

```
def is_palindrome(s):
    if s == s[::-1]:
        return True
    return False
```

Пример 2: вычисление последовательности Фибоначчи

```
def fibonacci(n):
    if n<0:
        print("Incorrect input")

    elif n==1:
        return 0
    else:
        for i in range(2,n+1):
            c = a + b
            a = b
            b = c
   return b
```

Пример 3: сортировка списка целых чисел

```
def sort_list(list):
    list.sort()
    return list
```

5.3 Анализ результатов

Анализ результатов позволяет сделать следующие наблюдения:

1. Модель демонстрирует стабильное улучшение как на обучающей, так и на валидационной выборке по мере увеличения числа эпох.
2. Наблюдается значительное снижение потерь в первые 2-3 эпохи, после чего улучшение становится более постепенным.
3. Генерируемый код в большинстве случаев синтаксически корректен, но имеет некоторые недостатки:

— В примере с последовательностью Фибоначчи модель не инициализирует переменные a и b

— Некоторые функции недостаточно обобщены (например, без должной обработки граничных случаев)

4. Модель успешно усваивает распространенные идиомы программирования на Python, такие как s[::-1] для обращения строки или list.sort() для сортировки списка.

6. Обсуждение

6.1 Преимущества гибридного подхода

Предложенный гибридный подход CNN+T5 имеет следующие преимущества по сравнению с чисто трансформерными моделями:

1. **Улучшенное представление локальных паттернов**: CNN-компонент позволяет более эффективно обрабатывать локальные синтаксические конструкции, типичные для программного кода.
2. **Стабильность обучения**: остаточные соединения и нормализация слоев способствуют более стабильному процессу обучения.
3. **Эффективность вычислений**: совместное использование CNN и трансформеров обеспечивает баланс между качеством представления и вычислительной эффективностью.

6.2 Ограничения и возможные улучшения

Несмотря на обнадеживающие результаты, предложенный подход имеет ряд ограничений:

1. **Ограниченная длина последовательности**: текущая реализация ограничена 256 токенами, что недостаточно для генерации сложных программ.
2. **Необходимость доработки сгенерированного кода**: как видно из примеров, сгенерированный код нередко требует ручной корректировки.
3. **Высокие вычислительные требования**: гибридная модель имеет большое количество параметров, что требует значительных вычислительных ресурсов.

Возможные направления улучшения включают:

1. Применение более сложных архитектур CNN для лучшего улавливания синтаксических паттернов.
2. Использование контекстно-зависимых сверток для учета семантики кода.
3. Разработка специализированных механизмов внимания, ориентированных на структуру программного кода.

7. Заключение

В данной работе представлена гибридная архитектура CNN+T5 для задачи генерации программного кода по описанию на естественном языке. Модель объединяет преимущества свёрточных нейронных сетей для обработки локальных паттернов и трансформеров для улавливания глобальных зависимостей.

Экспериментальные результаты показывают, что предложенный подход позволяет генерировать синтаксически корректный код для различных задач программирования. Применение техник teacher forcing и планирования скорости обучения обеспечивает стабильный процесс обучения и хорошую сходимость модели.

В качестве направлений дальнейших исследований можно выделить:

1. Расширение модели для поддержки других языков программирования
2. Улучшение обработки сложных алгоритмических задач
3. Интеграция статического анализа кода для обеспечения корректности сгенерированных программ

Список литературы:

1. Алламанис М. Обзор машинного обучения для больших кодов и естественности / М. Алламанис, Э. Т. Барр, П. Деванбу, К. Саттон // ACM Computing Surveys. – 2018. – Т. 51, № 4. – С. 1-37.

2. Фенг З. CodeBERT: предварительно обученная модель для программирования и естественных языков / З. Фенг, Д. Гуо, Д. Танг [и др.] // arXiv preprint arXiv:2002.08155. – 2020. – URL: https://arxiv.org/abs/2002.08155 (дата обращения: 19.03.2025). – Текст: электронный.

3. Ванг Ю. CodeT5: унифицированные предварительно обученные модели кодер-декодер с учетом идентификаторов для понимания и генерации кода / Ю. Ванг, В. Ванг, Ш. Джоти, К. Вашио // arXiv preprint arXiv:2109.00859. – 2021. – URL: https://arxiv.org/abs/2109.00859 (дата обращения: 19.03.2025). – Текст: электронный.

4. Лекун Я. Глубокое обучение / Я. Лекун, Й. Бенджио, Дж. Хинтон // Nature. – 2015. – Т. 521, № 7553. – С. 436-444.

5. Васвани А. Внимание - это все, что вам нужно / А. Васвани, Н. Шазир, Н. Пармар [и др.] // Advances in Neural Information Processing Systems. – 2017. – Т. 30. – С. 5998-6008.

6. Чен М. Оценка больших языковых моделей, обученных на коде / М. Чен, Дж. Творек, Х. Джун [и др.] // arXiv preprint arXiv:2107.03374. – 2021. – URL: https://arxiv.org/abs/2107.03374 (дата обращения: 19.03.2025). – Текст: электронный.

7. Святковский А. IntelliCode compose: генерация кода с использованием трансформера / А. Святковский, С. К. Денг, С. Фу, Н. Сундаресан // Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. – 2020. – С. 1433-1443.

8. Ким Ю. Сверточные нейронные сети для классификации предложений / Ю. Ким // arXiv preprint arXiv:1408.5882. – 2014. – URL: https://arxiv.org/abs/1408.5882 (дата обращения: 19.03.2025). – Текст: электронный.

9. Инь В. Синтаксическая нейронная модель для генерации кода общего назначения / В. Инь, Г. Нойбиг // arXiv preprint arXiv:1704.01696. – 2017. – URL: https://arxiv.org/abs/1704.01696 (дата обращения: 19.03.2025). – Текст: электронный.

10. Дорожная карта скрытого пространства для визуального планирования действий по манипулированию деформируемыми и жесткими объектами / Ю. Ванг, Й. Ли, Й. Тан, М. Гао // arXiv preprint arXiv:2003.08974. – 2020. – URL: https://arxiv.org/abs/2003.08974 (дата обращения: 19.03.2025). – Текст: электронный.

11. Ванг Х. Chestx-ray8: больничная база данных рентгеновских снимков грудной клетки и эталонные тесты по слабоконтролируемой классификации и локализации распространенных заболеваний грудной клетки / Х. Ванг, Й. Пенг, Л. Лу [и др.] // Proceedings of the IEEE conference on computer vision and pattern recognition. – 2017. – С. 2097-2106.

12. Жанг Х. Гибридный подход CNN-RNN для классификации текста / Х. Жанг, Й. Ли, Д. Сюй, З. Доу // 2019 IEEE International Conference on Big Data. – 2019. – С. 5737-5746.

13. Гэхринг Дж. Сверточное обучение последовательности / Дж. Гэхринг, М. Аули, Д. Гранжье [и др.] // International Conference on Machine Learning. – 2017. – С. 1243-1252.

15. Уильямс Р. Дж. Алгоритм обучения для непрерывно работающих полностью рекуррентных нейронных сетей / Р. Дж. Уильямс, Д. Зипсер // Neural computation. – 1989. – Т. 1, № 2. – С. 270-280.

16. Девлин Дж. BERT: предварительное обучение глубоких двунаправленных трансформеров для понимания языка / Дж. Девлин, М. В. Чанг, К. Ли, К. Тутанова // arXiv preprint arXiv:1810.04805. – 2018. – URL: https://arxiv.org/abs/1810.04805 (дата обращения: 19.03.2025). – Текст: электронный.