МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Лебедев Андрей Алексеевич

Большие языковые модели для генерации кода

Курсовая работа

Научный руководитель:
к.ф.-м.н. Тихомиров М.М.

Москва, 2024

Оглавление

1 Введение

Одной из главных технологий последних лет в области обработки естественного языка стали большие языковые модели (БЯМ), такие как GPT-4 [2], основанные на трансформерной архитектуре [1]. Первоначально разработанные для обработки и генерации естественного языка эти модели постепенно находят применение в разнообразных областях, включая генерацию программного кода.

Генерация кода с помощью больших языковых моделей – это перспективное направление, способное значительно упростить и ускорить процесс разработки программного обеспечения. Благодаря способности анализировать и обобщать большие объемы данных, БЯМ могут как предлагать решения типовых задач, так и адаптироваться к специфическим требованиям пользователей.

В данной курсовой работе рассматриваются возможности актуальных моделей для генерации кода с открытым исходным кодом и способы их оценки, а также проводятся эксперименты, направленные на исследование способов получения лучшего качества ответов.

1.1 Актуальность

Большие языковые модели демонстрируют значительный прогресс в понимании контекста и синтеза кода. Они могут помочь разработчикам, предлагая фрагменты кода, исправляя ошибки, документируя код и создавая целые программы по запросу на естественном языке. Так, например, GitHub CoPilot [3], дополняющий код в реальном времени и генерирующий код по запросу на естественном языке, собрал более 1,3 миллиона платных подписчиков, при этом более 50 000 организаций выбрали корпоративную версию, которая, по оценкам, повысит производительность разработчиков на 56%.

Кроме того, БЯМ для генерации кода оказывают влияние на процесс обучения программированию. Они могут выступать в роли виртуального

репетитора, предоставляя обучающимся мгновенные ответы на вопросы, помощь в поиске ошибок в коде, объяснение сложных концепций и готовые примеры кода.

1.2 Постановка задачи

Задача данной курсовой работы заключается в исследовании актуальных больших языковых моделей для генерации кода, а именно:

- научиться развертывать, запускать и тестировать модели;
- изучить способы оценки качества работы больших языковых моделей;
- сравнить качество работы некоторых моделей на русском языке по сравнению с английским.

2 Способы оценки моделей генерации кода

Оценка качества больших языковых моделей (БЯМ) для генерации кода важна для их разработки и применения. Существуют различные метрики и бенчмарки, позволяющие измерить эффективность и точность этих моделей.

Бенчмарк – это стандартизированное испытание эффективности модели, используемое для оценки различных возможностей БЯМ. Бенчмарк обычно состоит из набора данных, набора вопросов или заданий и механизма оценки. В контексте моделей для генерации кода наиболее популярными являются бенчмарки Mostly Basic Python Problems (MBPP) [4], HumanEval [5] и MultiPL-E [6], о которых далее пойдет речь.

2.1 MBPP

Впервые бенчмарк был описан Google Research в статье «Program Synthesis with Large Language Models» [4]. Он представляет собой около 1000 задач по программированию на Python, разработанных для решения программистами начального уровня, охватывающих основы программирования и функционал стандартных библиотек. Каждая задача состоит из описания на естественном языке, решения в формате кода и трех тестовых примеров.

text	code	test_list
Write a function to find the longest chain which can be formed from the given set of pairs.	class Pair(object): def __init__(self, a, b): self.a = a self.b = b def max_chain_length(arr, n): max = 0 mcl = [1 for i in range(n)] for i in range(1, n): for j in range(0, i): if (arr[i].a > arr[j].b and mcl[i] < mcl[j] + 1): mcl[i] = mcl[j] + 1 for i in range(n): if (max < mcl[i]): max = mcl[i] return max	["assert max_chain_length([Pair(5, 24), Pair(15, 25),Pair(27, 40), Pair(50, 60)], 4) == 3", "assert max_chain_length([Pair(1, 2), Pair(3, 4),Pair(5, 6), Pair(7, 8)], 4) == 4", "assert max_chain_length([Pair(19, 10), Pair(11, 12),Pair(13, 14), Pair(15, 16), Pair(31, 54)], 5) == 5"]

Рис. 2.1: Пример строки из датасета MBPP, где столбец text – описание задачи на естественном языке, code – решение, test_list – набор из трех автоматических тестов.

2.2 HumanEval

Датасет HumanEval от OpenAI, описанный в статье «Evaluating Large Language Models Trained on Code» [5], включает в себя 164 задачи по программированию. Каждая задача представлена сигнатурой функции, которую нужно написать и постановку задачи, оформленную как комментарий. К задаче прилагается решение и набор тестов. Все задачи были написаны вручную, чтобы они не оказались в обучающих данных.

prompt	canonical_solution	test
from typing import List def has_close_elements(numbers: List[float], threshold: float) -> bool: """ Check if in given list of numbers, are any two numbers closer to each other than given threshold. >>> has_close_elements([1.0, 2.0, 3.0], 0.5) False >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3) True """	for idx, elem in enumerate(numbers): for idx2, elem2 in enumerate(numbers): if idx != idx2: distance = abs(elem - elem2) if distance < threshold: return True return False	METADATA = { 'author': 'jt', 'dataset': 'test' } def check(candidate): assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False

Рис. 2.2: Пример строки из датасета HunamEval, где столбец prompt – сигнатура функции для реализации и описание задачи на естественном языке в виде комментария, canonical_solution – решение, test – набор из нескольких автоматических тестов.

2.3 MultiPL-E

MultiPL-E – это система перевода задач генерации кода с модульными тестами на другие языки программирования. С ее помощью был создан первый крупный многоязычный бенчмарк на основе двух других популярных бенчмарков на Python: HumanEval и MBPP, переведенных на 18 языков программирования.

(a) Original Python assertion.

```
assert lsi([0]) == (None, None)
```

(b) Equivalent R.

```
if(!identical(lsi(c(0)), c(NULL, NULL))){
    quit('no', 1)}
```

(c) Equivalent JavaScript.

```
assert.deepEqual(lsi([0]), [void 0, void 0]);
```

Рис. 2.3: Пример перевода автоматических тестов.

(a) Original Python docstring from HumanEval #95.

```
Given a dictionary, return True if all keys are
strings in lower case or all keys are strings in
upper case, else return False. The function should
return False is the given dictionary is empty.
```

(b) Terminology translated to Perl.

```
Given a hash, return 1 if all keys are strings
in lower case or all keys are strings in upper case,
else return "". The function should return "" is
the given hash is empty.
```

Рис. 2.4: Формулировка задачи на Python и ее перевод на Perl.

2.4 Метрика pass@k

Для оценки модели генерации кода требуется определить, какое количество задач из бенчмарка ей удалось решить верно. Проверка правильности решения реализуется с помощью автоматических тестов: из сгенерированного ответа извлекается код функции, к нему приписываются тесты, а затем этот код запускается. Можно дать модели несколько шансов написать правильное решение. Тогда оценить ее качество позволяет метрика pass@k. Pass@k — это метрика, которая оценивает вероятность того, что хотя бы одно из k сгенерированных моделью решений является правильным, то есть проходит все автоматические тесты:

$$\text{pass@k} = \frac{1}{|D|} \sum_{i=1}^{|D|} \left(\max_{j=1}^{k} S_{ij} \right),$$

где $|D|$ – количество задач в бенчмарке, S_{ij} – результат проверки правильности j-го решения для i-й задачи (1, если решение правильное, и 0, если неправильное). Таким образом, $\left(\max_{j=1}^{k} S_{ij}\right)$ будет равно 1, если хотя бы одно из первых k решений для i-й задачи является правильным, и 0, если все k решений неправильны.

3 Методы декодирования

В последние годы растёт интерес к open-ended генерации текста благодаря развитию БЯМ с трансформерной архитектурой. Результаты open-ended генерации впечатляют из-за возможности обобщения на новые задачи, например, на работу с кодом или получение нетекстовых данных в качестве входа. Достичь этого позволила не только модернизация архитектуры трансформера, но и разработка новых методов декодирования, о которых далее пойдет речь.

Авторегрессионная генерация текста основана на предположении, что распределение вероятностей последовательности слов может быть разложено на произведение распределений следующих слов:

$$P(w_{1:T} \mid W_0) = \prod_{t=1}^{T} P(w_t \mid w_{1:t-1}, W_0), w_{1:0} = \emptyset,$$

где W_0 – начальная последовательность слов в контексте.

3.1 Жадный поиск

Жадный поиск – это самый простой метод декодирования. На каждом шаге он выбирает слово с наибольшей вероятностью в качестве следующего слова: $w_t = argmax_w P(w \mid w_{1:t-1})$.

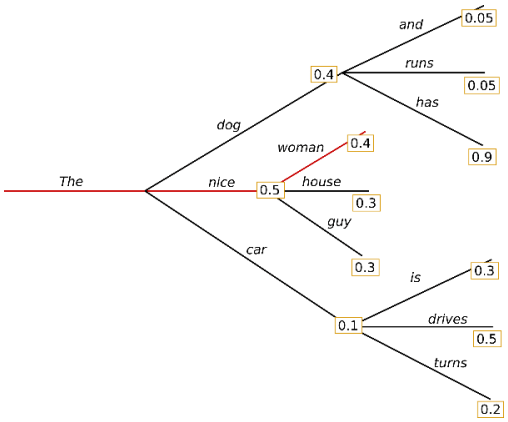

Рис. 3.1: Визуализация работы жадного декодирования.

3.2 Beam search

Beam search снижает риск потери неочевидных последовательностей слов с высокой суммарной вероятностью. При данном декодировании сохраняются наиболее вероятные num_beams последовательностей на каждом шаге, и в конечном итоге выбирается та, которая имеет самую высокую суммарную вероятность.

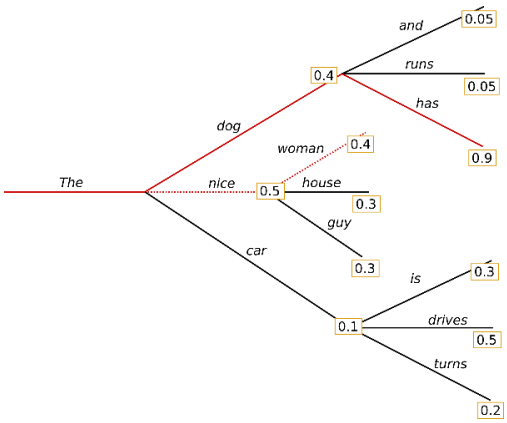

Рис. 3.2: Визуализация работы beam search с num_beams=2.

В open-ended генерации beam search может оказаться не лучшим вариантом по нескольким причинам:

1. Beam search хорошо работает в задачах, где длина желаемого ответа более или менее предсказуема, например, при машинном переводе. Но это не относится к open-ended генерации, где желаемая длина выходных данных может сильно варьироваться, например, при генерации историй или имитации диалога.

2. Beam search генерирует повторяющиеся последовательности. Это можно контролировать с помощью n-грамм, однако для этого требуется длительная настройка (например, чтобы биграмма «Санкт-Петербург» не была ограничена в повторении).

3. Как утверждается в работе Ari Holtzman et al. (2019) «The Curious Case of Neural Text Degeneration» [7], грамотная человеческая речь не строится согласно выбору из распределения слов с высокой вероятностью. Другими словами, сгенерированный текст должен получаться «живым», а не предсказуемым.

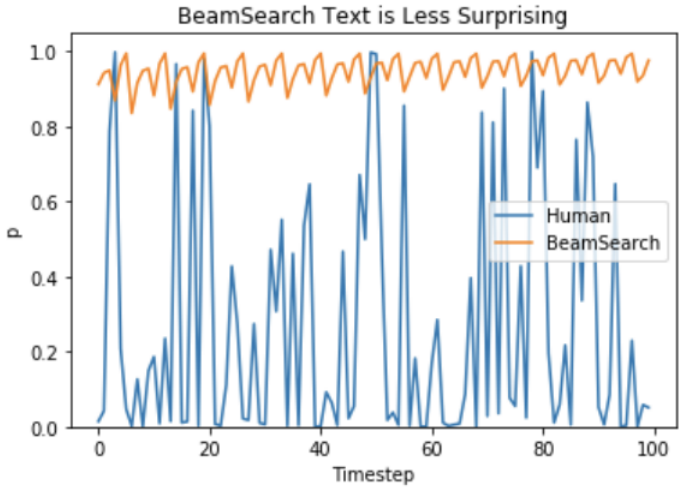

Рис. 3.3: Сравнение распределения вероятностей при выборе слов в человеческой речи и при beam search.

3.3 Top-p и Top-K Семплирование

Семлирование означает случайный выбор следующего слова w_t из распределения вероятностей.

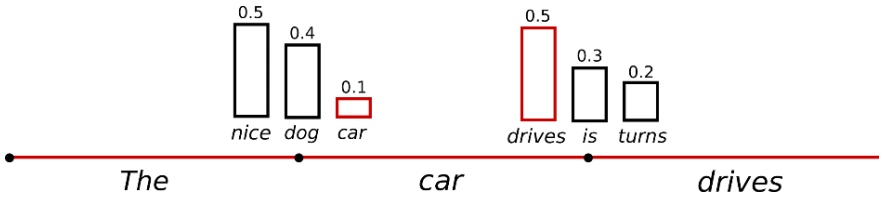

Рис. 3.4: Визуализация работы семплирования.

В статье Fan et. al. (2018) «Hierarchical Neural Story Generation» [8] представили идею Top-K семплирования. В Top-K семплировании отбираются K самых вероятных следующих слов, и вероятности перераспределяются только между ними. В GPT-2 [9] был использован такой метод декодирования, что повлекло успех модели в open-ended генерации.

Чтобы модель перестала генерировать бессмыслицу, можно снизить дисперсию распределения (увеличить правдоподобие для слов с высокой вероятностью и уменьшить его для остальных) за счет снижения температуры softmax.

Вместо семплирования из конкретного количества слов можно поступить следующим образом: выбирать из наименьшего возможного набора слов, суммарная вероятность которых превышает p, а затем перераспре-

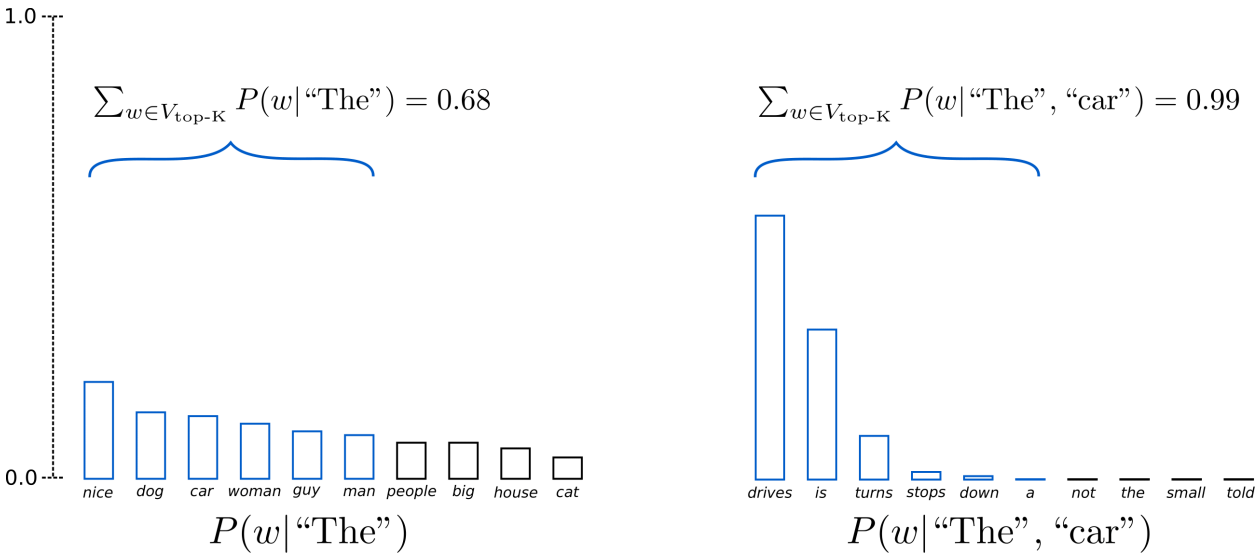

Рис. 3.5: Визуализация работы Top-K семплирования до и после снижения температуры при $K = 6$.

делять вероятности между этим набором слов. Данный подход называется Top-p семплированием. Таким образом, размер набора слов может динамически увеличиваться и уменьшаться в зависимости от распределения вероятностей для следующего слова.

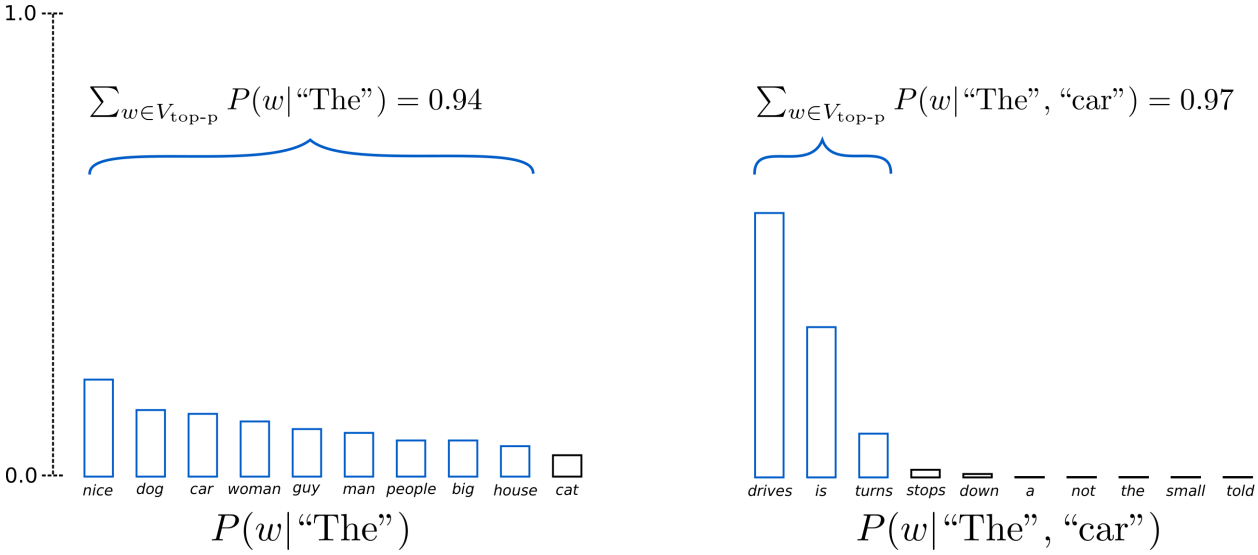

Рис. 3.6: Визуализация работы Top-p семплирования до и после снижения температуры при $p = 0.92$.

Top-p можно использовать в сочетании с Top-K для избежания очень низковероятных слов, добавляя при этом некоторое динамическое выделение.

4 Большие языковые модели для генерации кода

В этой главе будут рассмотрены три актуальные модели с открытым исходным кодом: DeepSeek Coder [10], Code Llama [11] и StarCoder2 [12]. Они являются лидерами в рейтинге прохождения бенчмарков HunamEval и MBPP, уступая лишь закрытым моделям, например, GPT-4. Для каждой модели будут описаны:

- данные для обучения;
- подходы к обучению;
- архитектура;
- прохождение бенчмарков.

4.1 DeepSeek Coder

4.1.1 Данные для обучения

Обучающие данные DeepSeek Coder содержат 2 триллиона токенов и состоят на 87% из программного кода (87 языков программирования, данные собраны до февраля 2023 года), на 10% из корпуса на английском языке, относящегося к коду и на 3% из корпуса на китайском языке. Корпус на английском языке состоит из материалов Markdown и StackExchange от GitHub, которые используются для улучшения понимания моделью концепций, связанных с кодом и улучшения ее способности справляться с использованием функций библиотек и исправлением ошибок. Корпус китайского языка состоит из статей, направленных на повышение уровня владения моделью китайским языком.

Интересно, что большие языковые модели для генерации кода в основном обучаются на уровне кода внутри конкретного файла, из-за чего игно-

рируются зависимости между различными модулями в проекте. В практических задачах такие модели с трудом поддаются эффективному масштабированию для обработки сценариев кода на уровне проекта. Поэтому для обучения DeepSeek сначала анализируются зависимости между файлами, а затем эти файлы упорядочиваются так, что обеспечить последовательное размещение контекста.

Недавние исследования (Lee et al. (2022) «Deduplicating training data makes language models better» [13]) продемонстрировали значительное повышение качества моделей, которое может быть достигнуто путем дедупликации обучающих данных. Обучающие корпуса часто содержат множество «почти-дубликатов», и производительность БЯМ может быть повышена за счет удаления длинных повторяющихся подстрок. В статье Kocetkov et al. (2022) «The stack: 3 tb of permissively licensed source code» [14] применили метод near-deduplication, что привело к серьезным улучшениям. Исследователи заключили, что данный метод является важнейшим этапом предварительной обработки обучающих данных для достижения конкурентоспособной производительности на бенчмарках. Таким образом, для обучения DeepSeek Coder также использовали near-deduplication, однако сделали это на уровне репозитория кода, а не на уровне конкретных файлов, поскольку описанный подход может отфильтровать определенные файлы в репозитории, потенциально нарушая его структуру.

DeepSeek-Coder-Instruct разработана на основе DeepSeek-Coder-Base посредством точной настройки на естественные инструкции в формате Alpaca Instruction [24].

4.1.2 Подходы к обучению

Модель DeepSeek Coder обучалась на двух задачах:

1. Предсказание следующего токена. Процесс обучения на эту задачу включает конкатенацию различных файлов для формирования входов фиксированной длины. Затем эти входы используются для обучения модели, позволяя ей предсказывать следующий токен на основе контекста.

2. Fill-in-the-Middle. В сценарии предобучения необходима задача гене-

рации контента для заполнения на основании левого и правого контекстов. Из-за особых зависимостей в языках программирования полагаться исключительно на предсказание следующего токена недостаточно для обучения возможности заполнения по контексту. Подход предполагает случайное разделение текста на три части, затем перестановку этих частей и соединение с помощью специальных символов. Целью этого метода является включение в процесс обучения задачи заполнения по контексту.

Для токенизации использована библиотека transformers [17] для обучения Byte Pair Encoding (BPE) [16] токенизатора.

4.1.3 Архитектура

DeepSeek Coder представлен в трех размерах: 1.3B, 6.7B, и 33B. Каждая модель представляет собой декодер трансформера с Rotary Position Embedding (RoPE) [15]. Модель 33B включает Grouped-Query-Attention (GQA) [18] с размером группы 8, что повышает эффективность как обучения, так и использования.

Hyperparameter	DeepSeek-Coder 1.3B	DeepSeek-Coder 6.7B	DeepSeek-Coder 33B
Hidden Activation	SwiGLU	SwiGLU	SwiGLU
Hidden size	2048	4096	7168
Intermediate size	5504	11008	19200
Hidden layers number	24	32	62
Attention heads number	16	32	56
Attention	Multi-head	Multi-head	Grouped-query (8)
Batch Size	1024	2304	3840
Max Learning Rate	5.3e-4	4.2e-4	3.5e-4

Рис. 4.1: Детали архитектуры моделей DeepSeek.

4.1.4 Прохождение бенчмарков

Чтобы оценить возможности модели на разных языках, был расширен бенчмарк Humaneval до 8 популярных языков программирования: Python, C++, Java, PHP, TypeScript (TS), C#, Bash и JavaScript (JS). Для тестирования на MBPP и HumanEval был выбран жадный поиск, few-shot промптинг, и каждая задача подавалась вместе с автоматическими тестами.

Model	Size	Python	C++	Java	PHP	TS	C#	Bash	JS	Avg
Multilingual Base Models										
code-cushman-001	12B	33.5%	31.9%	30.6%	28.9%	31.3%	22.1%	11.7%	-	-
CodeShell	7B	35.4%	32.9%	34.2%	31.7%	30.2%	38.0%	7.0%	33.5%	30.4%
CodeGeeX2	6B	36.0%	29.2%	25.9%	23.6%	20.8%	29.7%	6.3%	24.8%	24.5%
StarCoderBase	16B	31.7%	31.1%	28.5%	25.4%	34.0%	34.8%	8.9%	29.8%	28.0%
CodeLlama	7B	31.7%	29.8%	34.2%	23.6%	36.5%	36.7%	12.0%	29.2%	29.2%
CodeLlama	13B	36.0%	37.9%	38.0%	34.2%	45.2%	43.0%	16.5%	32.3%	35.4%
CodeLlama	34B	48.2%	44.7%	44.9%	41.0%	42.1%	48.7%	15.8%	42.2%	41.0%
DeepSeek-Coder-Base	1.3B	34.8%	31.1%	32.3%	24.2%	28.9%	36.7%	10.1%	28.6%	28.3%
DeepSeek-Coder-MQA-Base	5.7B	48.7%	45.3%	41.1%	39.7%	44.7%	41.1%	27.8%	42.2%	41.3%
DeepSeek-Coder-Base	6.7B	49.4%	50.3%	43.0%	38.5%	49.7%	50.0%	28.5%	48.4%	44.7%
DeepSeek-Coder-Base	33B	**56.1%**	**58.4%**	**51.9%**	**44.1%**	**52.8%**	**51.3%**	**32.3%**	**55.3%**	**50.3%**
Instruction-Tuned Models										
GPT-3.5-Turbo	-	76.2%	63.4%	69.2%	60.9%	69.1%	70.8%	42.4%	67.1%	64.9%
GPT-4	-	**84.1%**	**76.4%**	**81.6%**	**77.2%**	**77.4%**	**79.1%**	**58.2%**	**78.0%**	**76.5%**
DeepSeek-Coder-Instruct	1.3B	65.2%	45.3%	51.9%	45.3%	59.7%	55.1%	12.7%	52.2%	48.4%
DeepSeek-Coder-Instruct	6.7B	78.6%	63.4%	68.4%	68.9%	67.2%	72.8%	36.7%	72.7%	66.1%
DeepSeek-Coder-Instruct	33B	**79.3%**	**68.9%**	**73.4%**	**72.7%**	**67.9%**	**74.1%**	**43.0%**	**73.9%**	**69.2%**

Рис. 4.2: Прохождение многоязычного бенчмарка HumanEval различными моделями.

Model	Size	pass@1
CodeShell	7B	38.6%
CodeGeeX2	6B	36.2%
StarCoder	16B	42.8%
CodeLlama-Base	7B	38.6%
CodeLlama-Base	13B	48.4%
CodeLlama-Base	34B	55.2%
DeepSeek-Coder-Base	1.3B	46.2 %
DeepSeek-Coder-MQA-Base	5.7B	57.2 %
DeepSeek-Coder-Base	6.7B	60.6 %
DeepSeek-Coder-Base	33B	66.0 %
GPT-3.5-Turbo	-	70.8%
GPT-4	-	**80.0%**
DeepSeek-Coder-Instruct	1.3B	49.4%
DeepSeek-Coder-Instruct	6.7B	65.4%
DeepSeek-Coder-Instruct	33B	**70.0%**

Рис. 4.3: Прохождение бенчмарка MBPP различными моделями с метрикой pass@1.

4.2 Code Llama

4.2.1 Данные для обучения

Code Llama построена на базе Llama 2 [19] и дообучена на 500 миллиардах дополнительных токенов, состоящих в основном из кода (85%). Llama 2 в свою очередь обучена на 2 триллионах текстовых токенах, включая лишь 80 миллиардов токенов кода. В оригинальной статье Roziere B. et al. «Code

llama: Open foundation models for code» [11] сравнивается Code Llama 7B с идентичной моделью, обученной с нуля на том же наборе данных (рис. 4.4). По окончании обучения лосс модели, обученной с нуля, равен лоссу Code Llama 7B примерно на половине времени его обучения. И этот разрыв становится больше с течением времени.

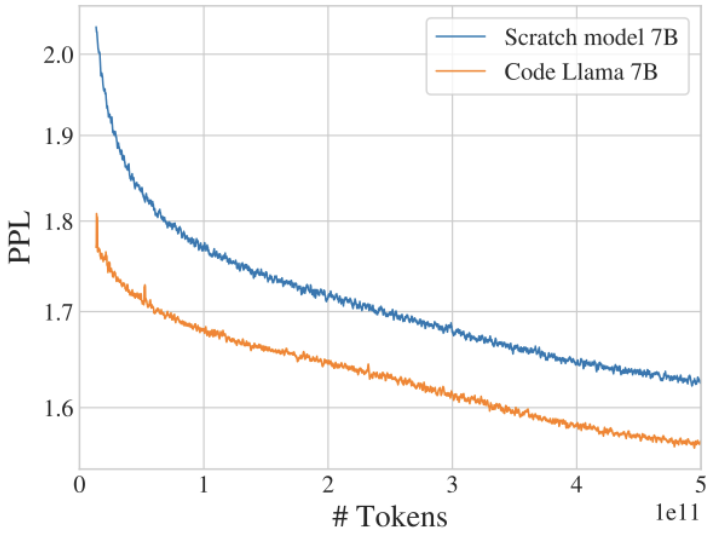

Рис. 4.4: Лосс при обучении Code Llama 7B и идентичной модели, обученной с нуля.

Code Llama – семейство моделей Llama 2, специализированных на работу с кодом, с тремя версиями, каждая в 4 размерах (7B, 13B, 34B и 70B параметров):

- Code Llama: базовая модель для задачи генерации кода;
- Code Llama Python: специально для работы с Python;
- Code Llama Instruct: дообучена на человеческих инструкциях и на синтетических данных с кодом.

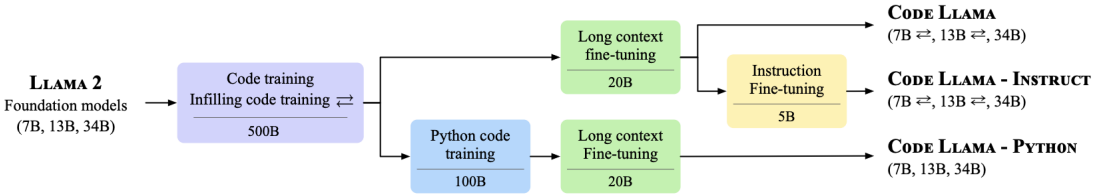

Рис. 4.5: Семейство моделей Code Llama.

Code Llama 7B, 13B и 34B дообучены на 500 миллиардах токенов, а Code Llama 70B на 1 триллионе. Code Llama обучались преимущественно на почти-дедуплицированном наборе данных открытого кода. 8% данных представляют собой описания на естественном языке, связанные с кодом, и содержат множество обсуждений кода или его фрагментов. Чтобы сохранить навыки понимания естественного языка, включена небольшая часть датасета для обучения Llama 2.

Dataset	Sampling prop.	Epochs	Disk size
Code Llama (500B tokens)			
Code	85%	2.03	859 GB
Natural language related to code	8%	1.39	78 GB
Natural language	7%	0.01	3.5 TB
Code Llama - Python (additional 100B tokens)			
Python	75%	3.69	79 GB
Code	10%	0.05	859 GB
Natural language related to code	10%	0.35	78 GB
Natural language	5%	0.00	3.5 TB

Рис. 4.6: Обучающий датасет Code Llama и Code Llama Python.

4.2.2 Подходы к обучению

Обучение на задачу предсказания следующего токена и тонкая настройка подходят для завершения кода, однако не дают возможности заполнить его недостающую часть на основе контекста. Поэтому обучение Code Llama включает как авторегрессию, так и завершение по контексту, что позволяет применять модель, например, для завершения кода в реальном времени в редакторах кода. Обучение на задачу Fill-in-the-Middle реализовано аналогично DeepSeek Coder. В качестве токенизатора выбран алгоритм BPE, как и в Llama 2.

4.2.3 Архитектура

Архитектура модели заимствована из Llama [20] и представляет собой стандартную архитектуру декодера трансформера со следующими модификациями: нормализация RMSNorm [21], функция активации SwiGLU [22] и Rotary Positional Embeddings (RoPE).

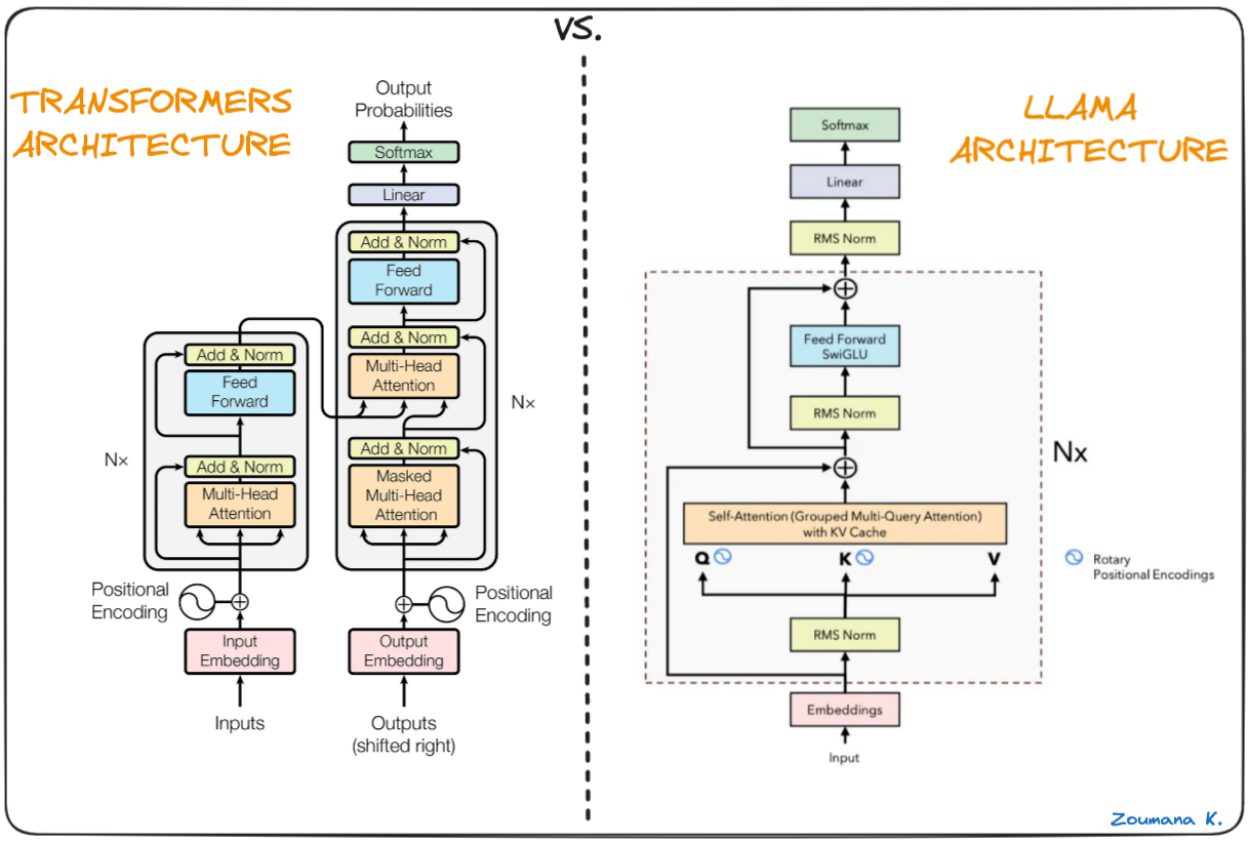

Рис. 4.7: Стандартная архитектура трансформера и ее модифицированная версия в Llama 1.

4.2.4 Прохождение бенчмарков

Оценка pass@1 моделей вычисляется с использованием жадного декодирования, а pass@10 и pass@100 с использованием Top-p семплирования с $p = 0,95$ и температурой 0,8. Для тестирования выбраны zero-shot на HumanEval и 3-shot на MBPP. Результаты отображены на рис. 4.8.

4.3 StarCoder2

4.3.1 Данные для обучения

Модель StarCoder2 разработана в рамках проекта BigCode. Ранее сообщество выпустило The Stack v1 [23] – набор данных с кодом объемом 6,4 ТБ на 384 языках программирования. Stack v1 включает в себя инструмент под названием «Am I in The Stack», предназначенный для разработчиков и позволяющий проверить, включен ли их код в набор данных. 4 мая 2023 года BigCode выпустили StarCoder, обученную на The Stack v1, и на момент сво-

Model	Size	HumanEval			MBPP		
		pass@1	pass@10	pass@100	pass@1	pass@10	pass@100
code-cushman-001	12B	33.5%	-	-	45.9%	-	-
GPT-3.5 (ChatGPT)	-	48.1%	-	-	52.2%	-	-
GPT-4	-	67.0%	-	-	-	-	-
PaLM	540B	26.2%	-	-	36.8%	-	-
PaLM-Coder	540B	35.9%	-	88.4%	47.0%	-	-
PaLM 2-S	-	37.6%	-	88.4%	50.0%	-	-
StarCoder Base	15.5B	30.4%	-	-	49.0%	-	-
StarCoder Python	15.5B	33.6%	-	-	52.7%	-	-
StarCoder Prompted	15.5B	40.8%	-	-	49.5%	-	-
Llama 2	7B	12.2%	25.2%	44.4%	20.8%	41.8%	65.5%
	13B	20.1%	34.8%	61.2%	27.6%	48.1%	69.5%
	34B	22.6%	47.0%	79.5%	33.8%	56.9%	77.6%
	70B	30.5%	59.4%	87.0%	45.4%	66.2%	83.1%
Code Llama	7B	33.5%	59.6%	85.9%	41.4%	66.7%	82.5%
	13B	36.0%	69.4%	89.8%	47.0%	71.7%	87.1%
	34B	48.8%	76.8%	93.0%	55.0%	76.2%	86.6%
	70B	53.0%	84.6%	96.2%	62.4%	81.1%	91.9%
Code Llama - Instruct	7B	34.8%	64.3%	88.1%	44.4%	65.4%	76.8%
	13B	42.7%	71.6%	91.6%	49.4%	71.2%	84.1%
	34B	41.5%	77.2%	93.5%	57.0%	74.6%	85.4%
	70B	67.8%	90.3%	97.3%	62.2%	79.6%	89.2%
Unnatural Code Llama	34B	62.2%	85.2%	95.4%	61.2%	76.6%	86.7%
Code Llama - Python	7B	38.4%	70.3%	90.6%	47.6%	70.3%	84.8%
	13B	43.3%	77.4%	94.1%	49.0%	74.0%	87.6%
	34B	53.7%	82.8%	94.7%	56.2%	76.4%	88.2%
	70B	57.3%	89.3%	98.4%	65.6%	81.5%	91.9%

Рис. 4.8: Оценка Code Llama и других моделей с pass@k на HumanEval и MBPP.

его выпуска модель 15B была лучшей открытой БЯМ для генерации кода. The Stack v2 [12] основан на крупном архиве с кодом Software Heritage, который включает более 600 языков программирования. Помимо репозиториев использовались Github issues, Kaggle и Jupyter ноутбуки, документация к коду и другие датасеты на естественном языке, связанные с математикой и программированием. Чтобы подготовить данные для обучения, была выполнена дедупликация, фильтрация для устранения некачественного кода, маскировка личной информации и удаление вредоносного кода. С помощью этого нового датасета, состоящего из более чем 900 миллиардов токенов, что в 4 раза больше, чем в The Stack v1, была разработана StarCoder2.

	Dataset	Tokens (B)	3B	7B	15B
	the-stack-v2-train-smol	525.5	✓	✓	✗
	the-stack-v2-train-full	775.48	✗	✗	✓
the-stack-v2-train-extras	Pull requests	19.54	✓	✓	✓
	Issues	11.06	✓	✓	✓
	Jupyter structured	14.74	✓	✓	✓
	Jupyter scripts	16.29	✓	✓	✓
	Kaggle scripts	1.68	✓	✓	✓
	Documentation	1.6	✓	✓	✓
	OpenWebMath	14.42	✗	✓	✓
	Wikipedia	6.12	✗	✓	✓
	StackOverflow	10.26	✓	✓	✓
	Arxiv	30.26	✗	✓	✓
	LHQ	5.78	✓	✓	✓
	Intermediate Repr.	6	✓	✓	✓
	Unique tokens (B)		622.09	658.58	913.23

Рис. 4.9: Состав обучающих данных моделей StarCoder2.

4.3.2 Подходы к обучению

В отличие от описанных ранее моделей StarCoder2 обучался лишь на задачу генерации следующего токена. Токенизатор унаследован от StarCoderBase и является BPE-токенизатором, обученным на небольшом срезе The Stack v1.24. В ходе экспериментов авторы заключили, что увеличение размера словаря до 100.000 токенов не улучшает производительность, поэтому оставили 49.152 токена.

4.3.3 Архитектура

Архитектура StarCoder2 похожа на CodeLlama, то есть стандартная архитектура декодера трансформера модифицируется посредством нормализации RMSNorm, функции активации SwiGLU, Rotary Positional Embeddings (RoPE) и заменой Multi-Query Attention на Grouped Query Attention.

4.3.4 Прохождение бенчмарков

Авторы оригинальной статьи заключили, что бенчмарки MBPP и HumanEval, во-первых, содержат ошибки, а во-вторых автоматические тесты в них не

являются достаточными для проверки сгенерированных функций. Поэтому было принято решение использовать для оценки в том числе HumanEval+ and MBPP+, которые содержат в 80 и 35 больше тестов. Оценка производилась по метрике pass@1 с использованием жадного поиска.

Model	HumanEval	HumanEval+	MBPP	MBPP+
StarCoderBase-3B	21.3	17.1	42.6	35.8
DeepSeekCoder-1.3B	28.7	23.8	55.4	46.9
StableCode-3B	28.7	24.4	53.1	43.1
StarCoder2-3B	**31.7**	**27.4**	**57.4**	**47.4**
StarCoderBase-7B	30.5	25.0	47.4	39.6
CodeLlama-7B	33.5	25.6	52.1	41.6
DeepSeekCoder-6.7B	**47.6**	**39.6**	**70.2**	**56.6**
StarCoder2-7B	35.4	29.9	54.4	45.6
StarCoderBase-15B	29.3	25.6	50.6	43.6
CodeLlama-13B	37.8	32.3	62.4	52.4
StarCoder2-15B	**46.3**	**37.8**	**66.2**	**53.1**
CodeLlama-34B	48.2	44.3	65.4	52.4
DeepSeekCoder-33B	**54.3**	**46.3**	**73.2**	**59.1**

Рис. 4.10: Результаты StarCoder2 и других моделей на бенчмарках HumanEval, HumanEval+, MBPP и MBPP+ по метрике pass@1.

5 Эксперименты

Помимо исследования принципов работы больших языковых моделей для генерации кода в рамках курсовой работы стояла практическая задача задача, а именно изучить способы оценки качества работы БЯМ, научиться развертывать, запускать и тестировать модели, научиться работать с инструктивными и обычными моделями, а также оценить качество работы некоторых моделей на русском языке по сравнению с английским. В связи с этим была проведена серия экспериментов, о которых далее пойдет речь.

5.1 DeepSeek-Coder-Instruct-7B, MBPP, zero-shot, no tests

Задачей эксперимента было протестировать инструктивную версию DeepSeek Coder 7B на бенчмарке MBPP с собственным системным промптом, но без подсказок, то есть в формате zero-shot. Более того, сами задачи подавались без автоматических тестов.

Цели:

- научиться запускать и оценивать модель;
- изучить влияние добавления системного промпта;
- подобрать лучшие параметры и изучить поведение модели при их изменении;
- посмотреть на качество модели на задачах на естественном языке.

Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования было выбрано Top-p и Top-K семплирование с $K = 40$ и $p = 0.95$. Результат: 34,6% (173/500 тестов).

Выводы:

- в процессе эксперимента удалось научиться развертывать, запускать и оценивать модель на бенчмарке;

- была проведена серия дополнительных экспериментов над параметрами p и K, в результате которых оказалось, что модель DeepSeek-Coder-Instruct 7B устойчива даже к серьезным отклонениям от оптимальных параметров;

- не удалось воспроизвести результаты оригинальной статьи, где оценка той же модели на том же бенчмарке составила 65.4%. К возможным причинам можно отнести zero-shot, отсутствие автоматических тестов в промпте, изменение оригинального системного промпта.

5.2 DeepSeek-Coder-Instruct-7B, MBPP, few-shot, no tests

Задачей данного эксперимента было протестировать инструктивную модель DeepSeek 7B на бенчмарке MBPP с собственным системным промптом и в формате few-shot. Сами задачи снова подавались без автоматических тестов.

Цели:

- научиться работать с chat_template для инструктивных моделей и корректно составить few-shot промпт;

- изучить, как изменение промпта путем добавления примеров диалогов между пользователем и моделью может повлиять на качество генерации

Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования было выбрано Top-p и Top-K семплирование с $K = 40$ и $p = 0.95$. Результат: 38,2% (191/500 тестов).

Выводы:

- в процессе эксперимента удалось корректно составить few-shot промпт для модели DeepSeek Coder;

- добавление примеров взаимодействия между моделью и пользователем дало неплохое увеличение качества (на 3.6%), однако такой результат все еще далек от того, что в оригинальной статье.

< | begin_of_sentence | > You are a smart assistant in writing code that helps the user solve his tasks. Below is an instruction describing the task. Write an answer that exactly fulfills the user's request.

Instruction:
Write a function to find the minimum cost path to reach (m, n) from (0, 0) for the given cost matrix cost[][] and a position (m, n) in cost[][]. The function should have the following name: min_cost.
Response:
expected model's answer
<|EOT|>

Instruction:
Write a function to find the similar elements from the given two tuple lists. The function should have the following name: similar_elements.
Response:
expected model's answer
<|EOT|>

Instruction:
Write a python function to remove first and last occurrence of a given character from the string. The function should have the following name: remove_Occ.
Response:

Рис. 5.1: Пример few-shot для DeepSeek Coder с собственным системным промптом и 2 примерами диалога между моделью и пользователем из датасета MBPP.

5.3 DeepSeek-Coder-Instruct-7B, MBPP, with tests, original prompt

Задачей данного эксперимента являлось воспроизвести результаты из оригинальной статьи DeepSeek Coder на бенчмарке MBPP. Для этого промпт был составлен аналогично тому, как это делали авторы, то есть был использован их системный промпт, их способ организации few-shot и были добавлены автоматические тесты к каждой задаче.

Цель: воссоздать условия тестирования модели из оригинальной статьи и попытаться воспроизвести то же качество.

Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования был выбран жадный поиск. Результат: 56% (280/500 тестов).

Выводы:

- в процессе эксперимента удалось воссоздать условия оценки модели в оригинальной статье;
- результат практически повторяет заявленный авторами (65.4%), из чего можно заключить, что добавление автоматических тестов в каждую задачу значительно увеличивает качество ответа модели. Однако такие условия тестирования можно расценить как не совсем естественные, так как в бенчмарке были заготовлены автоматические те-

сты, однако пользователь далеко не всегда сможет вместе с задачей на естественном языке подать аналогичные тесты.

You are an AI programming assistant, utilizing the Deepseek Coder model, developed by Deepseek Company, and you only answer questions related to computer science. For politically sensitive questions, security and privacy issues, and other non-computer science questions, you will refuse to answer

Please refer the given examples and generate a python function for my problem.
Examples are listed as follows:

- Example 1:
>>> Problem:
Write a function to find the similar elements from the given two tuple lists.
>>> Test Cases:
assert similar_elements((3, 4, 5, 6),(5, 7, 4, 10)) == (4, 5)
assert similar_elements((1, 2, 3, 4),(5, 4, 3, 7)) == (3, 4)
assert similar_elements((11, 12, 14, 13),(17, 15, 14, 13)) == (13, 14)

>>> Code:
```python
def similar_elements(test_tup1, test_tup2):
  res = tuple(set(test_tup1) & set(test_tup2))
  return (res)
```
The same for example 2 and 3.

Here is my problem:
>>> Problem:
Write a function to concatenate all elements of the given list into a string.
>>> Test Cases:
assert concatenate_elements(['hello','there','have','a','rocky','day']) == ' hello there have a rocky day'
assert concatenate_elements(['Hi', 'there', 'How','are', 'you']) == ' Hi there How are you'
assert concatenate_elements(['Part', 'of', 'the','journey', 'is', 'end']) == ' Part of the journey is end'

Рис. 5.2: Пример оригинального промпта DeepSeek Coder Instruct.

5.4 DeepSeek-Coder-Instruct-7B, MBPP, with tests, custom prompt

Задачей данного эксперимента являлось снова воспроизвести результаты из оригинальной статьи DeepSeek Coder на бенчмарке MBPP, однако в этот раз с собственным системным промптом, но с оригинальным способом организации few-shot и автоматическими тестами.

Цель: узнать, как изменение системного промпта может повлиять на качество модели.

Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования был выбран жадный поиск. Результат: 58.4% (292/500 тестов).

Выводы:

- качество оказалось еще лучше, чем с оригинальным промптом, однако разница между двумя последними экспериментами составляет

всего 2.4%, что говорит о том, что системный промпт не сильно влияет на поведение модели.

5.5 DeepSeek-Coder-Instruct-7B quantized, MBPP, with tests, original prompt

Задачей данного эксперимента являлось оценить изменения в поведении квантизированной модели DeepSeek Coder 7B Instruct по сравнению с обычной.

Цель: выяснить, как квантизация влияет на качество генерации.

Модель квантизировалась с помощью встроенных методов библиотек transformers и bitsandbytes, а именно технологии LLM.int8() [25], где используется absmax квантизация и отдельная обработка выбросов. Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования было выбрано жадное декодирование. Результат: 58.4% (292/500 тестов).

Выводы:

- несмотря на то, что обычно качество модели незначительно снижается при квантизации, на результат прохождения бенчмарка MBPP моделью DeepSeek Coder 7B это не повлияло.

5.6 DeepSeek-Coder-Instruct-7B, HumanEval

Задача данного эксперимента заключалась в том, чтобы протестировать инструктивную модель DeepSeek Coder 7B на бенчмарке HumanEval в режиме zero-shot.

Цели:

- научиться работать бенчмарком HumanEval и попытаться воспроизвести результаты оригинальной статьи (78.6%);
- посмотреть, как изменится качество генерации, если решается задача продолжения кода вместо задачи на естественном языке (как это было в MBPP).

Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования было выбрано Top-p и Top-K семплирование с $K = 40$ и

$p = 0.95$, режим zero-shot. Результат: 68.3% (112/164 тестов).

Выводы:

- в процессе эксперимента удалось получить достаточно высокий результат инструктивной модели DeepSeek Coder 7B, однако он все равно ниже того, что опубликован в оригинальной статье;
- к возможным причинам результата ниже заявленного можно отнести изменение системного промпта модели;
- модели удается значительно лучше решать задачу продолжения кода вместо генерации по запросу на естественном языке.

5.7 DeepSeek-Coder-Instruct-7B, RU-HumanEval

Задача данного эксперимента заключалась в том, чтобы протестировать инструктивную модель DeepSeek Coder 7B на русской версии бенчмарка HumanEval в режиме zero-shot. Русская версия HumanEval представляет собой качественно переведенный оригинальный датасет с сохранением его структуры.

Цель: оценить, как изменится качество генерации на тех же заданиях, что в оригинальном HumanEval, но сформулированных на русском языке.

Оценка вычислялась с помощью метрики pass@1, в качестве метода декодирования было выбрано Top-p и Top-K семплирование с $K = 40$ и $p = 0.95$, режим zero-shot. Результат: 67.6% (111/164 тестов).

Выводы:

- качество DeepSeek Instruct 7B не ухудшается при работе на русском языке по сравнению с английским.

№	Модель	Бенчмарк	Формат	Метод декодирования	pass@1
1	DeepSeek-Coder-Instruct-7B	MBPP	zero-shot, no tests, custom prompt	Top-p и Top-K семплирование	34.6%
2	DeepSeek-Coder-Instruct-7B	MBPP	few-shot, no tests, custom prompt	Top-p и Top-K семплирование	38.2%
3	DeepSeek-Coder-Instruct-7B	MBPP	few-shot, with tests, original prompt	Жадное декодирование	56%
4	DeepSeek-Coder-Instruct-7B	MBPP	few-shot, with tests, custom prompt	Жадное декодирование	58.4%
5	DeepSeek-Coder-Instruct-7B quantized	MBPP	few-shot, with tests, original prompt	Жадное декодирование	58.4%
6	DeepSeek-Coder-Instruct-7B	HumanEval	zero-shot, no tests, custom prompt	Top-p и Top-K семплирование	68.3%
7	DeepSeek-Coder-Instruct-7B	RU-HumanEval	zero-shot, no tests, custom prompt	Top-p и Top-K семплирование	67.6%

Таблица 5.1: Результаты экспериментов

6 Заключение

В первую очередь в рамках данной курсовой работы были рассмотрены различные методики оценки больших языковых моделей для генерации кода, а именно бенчмарки MBPP, HumanEval и MultiPL-E и метрика pass@k

Отдельное внимание было уделено методам декодирования, а именно жадному поиску, beam search и top-p/top-k семплированию, каждый из которых имеет свои преимущества и недостатки в контексте генерации кода. Более того, для данного домена до сих пор нет явного решения, какой из методов лучше использовать.

Далее были подробно изучены три модели для генерации кода: DeepSeek Coder, Code Llama и StarCoder2. Для каждой модели были описаны данные для обучения, детали предобучения и тонкой настройки, архитектура и результаты на бенчмарках.

Проведенные эксперименты позволили на практике рассмотреть возможности БЯМ для генерации кода. DeepSeek-Coder-Instruct-7B стала ключевой моделью для изучения и продемонстрировала высокую производительность в различных сценариях. Были воспроизведены результаты модели на различных бенчмарках, изучен промптинг, влияние zero-shot и few-shot подходов, квантизации. А также стало ясно, что БЯМ не ухудшают свое качество при работе с русским языком в сравнении с английским. Стоит добавить, что на данный момент модели лучше справляются с задачей продолжения кода, нежели генерацией по запросу на естественном языке.

В заключение можно отметить, что большие языковые модели имеют огромный потенциал для автоматизации процесса написания кода. Хотя сейчас открытые модели находятся на начальных этапах своего развития, в перспективе они могут существенно повысить эффективность разработки программного обеспечения, предоставляя разработчикам мощные инструменты для генерации кода на различных языках программирования. Дальнейшие исследования в этой области могут привести к еще большему совершенствованию этих моделей и расширению их возможностей, что сделает их незаменимыми ассистентами в ежедневных задачах.

Литература

[1] Vaswani A. et al., *Attention is all you need* //Advances in neural information processing systems. – 2017. – Т. 30.

[2] OpenAI, *Gpt-4 technical report* //arXiv preprint arXiv:2303.08774. – 2023.

[3] Dakhel A. M. et al, *Github copilot ai pair programmer: Asset or liability?* //Journal of Systems and Software. – 2023. – Т. 203. – С. 111734.

[4] Austin J. et al., *Program synthesis with large language models* //arXiv preprint arXiv:2108.07732. – 2021.

[5] Chen M. et al., *Evaluating large language models trained on code* //arXiv preprint arXiv:2107.03374. – 2021.

[6] Cassano F. et al., *Multipl-e: A scalable and extensible approach to benchmarking neural code generation* //arXiv preprint arXiv:2208.08227. – 2022.

[7] Holtzman A. et al., *The curious case of neural text degeneration* //arXiv preprint arXiv:1904.09751. – 2019.

[8] Fan A., Lewis M., Dauphin Y., *Hierarchical neural story generation* //arXiv preprint arXiv:1805.04833. – 2018.

[9] Radford A. et al., *Language models are unsupervised multitask learners* //OpenAI blog. – 2019. – Т. 1. – №. 8. – С. 9.

[10] Guo D. et al., *DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence* //arXiv preprint arXiv:2401.14196. – 2024.

[11] Roziere B. et al. *Code llama: Open foundation models for code* //arXiv preprint arXiv:2308.12950. – 2023.

[12] Lozhkov A. et al., *StarCoder 2 and The Stack v2: The Next Generation* //arXiv preprint arXiv:2402.19173. – 2024.

[13] Lee K. et al., *Deduplicating training data makes language models better* //arXiv preprint arXiv:2107.06499. – 2021.

[14] Kocetkov D. et al. *The stack: 3 tb of permissively licensed source code* //arXiv preprint arXiv:2211.15533. – 2022.

[15] Su J. et al., *Roformer: Enhanced transformer with rotary position embedding* //Neurocomputing. – 2024. – Т. 568. – С. 127063.

[16] Sennrich R., Haddow B., Birch A., *Neural machine translation of rare words with subword units* //arXiv preprint arXiv:1508.07909. – 2015.

[17] Wolf T. et al., *Huggingface's transformers: State-of-the-art natural language processing* //arXiv preprint arXiv:1910.03771. – 2019.

[18] Ainslie J. et al., *Gqa: Training generalized multi-query transformer models from multi-head checkpoints* //arXiv preprint arXiv:2305.13245. – 2023.

[19] Touvron H. et al., *Llama 2: Open foundation and fine-tuned chat models* //arXiv preprint arXiv:2307.09288. – 2023.

[20] Touvron H. et al., *Llama: Open and efficient foundation language models* //arXiv preprint arXiv:2302.13971. – 2023.

[21] Zhang B., Sennrich R. *Root mean square layer normalization* //Advances in Neural Information Processing Systems. – 2019. – Т. 32.

[22] Shazeer N., *Glu variants improve transformer* //arXiv preprint arXiv:2002.05202. – 2020.

[23] Kocetkov D. et al., *The stack: 3 tb of permissively licensed source code* //arXiv preprint arXiv:2211.15533. – 2022.

[24] Taori R. et al., *Alpaca: A strong, replicable instruction-following model* //Stanford Center for Research on Foundation Models. – 2023. – Т. 3. – №. 6. – С. 7.

[25] Dettmers T. et al., *LLM.int8(): 8-bit matrix multiplication for transformers at scale.* - 2022 //CoRR abs/2208.07339.