DOI: 10.15514/ISPRAS-2026-38(2)-8

Оценка корректности сгенерированного нейросетями кода: вероятностный подход

Д.А. Авагян, ORCID: 0009-0009-2409-7357 <david_avagyan@list.ru>

Московский государственный университет имени М.В. Ломоносова,
Россия, 119991, Москва, Ленинские горы, д. 1.

Аннотация. Большие языковые модели находят всё более широкое применение в разработке программного обеспечения. Однако исследование корректности генерируемого ими кода осложняется недостаточной формализацией понятия корректности программ. В данной работе описан вероятностный подход к оценке корректности кода, генерируемого нейросетями. Предложена метрика корректности TSA (Test Suite Accuracy), естественным образом выводимая в данной формализации, а также проводится сравнение с метрикой Pass@1. Проведённые эксперименты с 5 языковыми моделями Phi-1, Phi-2, Phi-3-mini-4k, Phi-4-mini и Qwen2.5-Coder подтверждают описанные теоретические свойства метрик. Практическим результатом проведённого исследования являются набор задач HumanEval++, расширяющий набор данных HumanEval+, и построенная на его основе реализация метрики TSA.

Ключевые слова: большие языковые модели; программная инженерия; генерация кода; качество кода; корректность кода; метрики.

Для цитирования: Авагян Д.А. Оценка корректности сгенерированного нейросетями кода: вероятностный подход. Труды ИСП РАН, том 38, вып. 2, 2026 г., стр. 111–128. DOI: 10.15514/ISPRAS–2026–38(2)–8.

Correctness evaluation of LLM-generated code: probabilistic approach

D.A. Avagian, ORCID: 0009-0009-2409-7357 <david_avagyan@list.ru>

Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.

Abstract. Large language models (LLMs) are increasingly being used in software development. However, the lack of a formal definition for code correctness complicates the study of the correctness of generated code. This paper describes a probabilistic approach to define the correctness of LLM-generated code. We introduce the TSA (Test Suite Accuracy) correctness metric, naturally derived within the presented approach, and compare it with Pass@1. Our evaluation of 5 LLMs (Phi-1, Phi-2, Phi-3-mini-4k, Phi-4-mini, and Qwen2.5-Coder) confirms the described properties of both metrics. The key contributions of the conducted research include the HumanEval++ dataset, which extends HumanEval+, and the TSA metric implementation built upon it.

Keywords: large language models; software engineering; code generation; code quality; code correctness; metrics.

For citation: Avagian D.A. Correctness evaluation of LLM-generated code: probabilistic approach. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 2, 2026, pp. 111-128 (in Russian). DOI: 10.15514/ISPRAS-2026-38(2)-8.

1. Введение

Любое программное обеспечение следует определённому жизненному циклу [1], который принято разбивать на несколько стадий:

- анализ и выработка требований;
- планирование;
- проектирование и дизайн;
- разработка;
- тестирование;
- эксплуатация.

Каждая из стадий характеризуется набором процессов и задач, для решения которых могут применяться большие языковые модели. Некоторыми примерами таких задач служат генерация кода и спецификации, проектирование программных интерфейсов, поиск уязвимостей, дедупликация кода, оценка трудоёмкости задач [2-11].

В данной статье рассматривается задача генерации программного кода по условию на естественном языке. Одной из основных проблем при решении этой задачи является оценка качества полученного решения. В общем случае ставится задача оценки качества программы независимо от модели, с помощью которой эта программа сгенерирована.

Качество программы – составное понятие, не имеющее на данный момент единого формального определения. Термин «качество программы» в научных работах может быть неразрывно связан с такими свойствами, как корректность, эффективность, безопасность и уязвимость, поддерживаемость, читаемость, сложность, соответствие стилистическим требованиям, стабильность выполнения, модульность, надёжность и масштабируемость [12-29]. Некоторые из перечисленных свойств кода, как правило, оцениваются с помощью фиксированного набора тестов [30-35], тогда как другие требуют привлечения асессоров [19, 36-40]. В ряде работ качество программ оценивается на основе текстового расстояния до эталонной программы [41-44]. Тем не менее, ключевым свойством качественной программы может считаться корректность, поскольку анализ и оценка остальных свойств некорректной программы не имеет смысла.

Корректность кода, в свою очередь, также может рассматриваться как составное свойство программы. Синтаксическая корректность кода обеспечивается успешной сборкой, запуском и завершением программы, тогда как семантическая корректность – это свойство кода правильно решать поставленную задачу. Анализ корректности кода требует рассмотрения как наиболее вероятных входных данных, так и краевых случаев, соответствующих условию исходной задачи.

Основным методом проверки корректности программ является модульное и функциональное тестирование. Применимость подхода в каждом конкретном случае зависит от наличия полного тестового множества и окружения для запуска программ. Стоит отметить, что полнота тестового покрытия зависит от модели вычислений, используемой для описания программы. Как следствие, понятие корректности и её оценка также зависят от модели вычислений.

Наиболее распространённой метрикой, используемой для оценки корректности программ, сгенерированных нейросетями, является Pass@k – это оценка вероятности того, что хотя бы одна из k программ, случайно выбранных из n сгенерированных моделью, проходит все тесты [30]. Значения метрики для всего набора задач усредняются для оценки общих способностей нейросети к генерации корректного кода. Так, в метрике учитывается итеративность процесса генерации кода нейросетями, однако номер первой итерации, на которой нейросеть сгенерировала корректный код, не влияет на значение метрики. Таким образом, при $k > 1$ оптимизация метрики Pass@k положительно подкрепляет недетерминированность модели, то есть решения, генерируемые нейросетью после оптимизации метрики, могут регулярно оказываться неверными. В связи с этим свойством из семейства Pass@k в данном исследовании рассматривается только метрика Pass@1, равная доле задач в тестовой выборке, для которых нейросеть с первого раза сгенерировала код, проходящий все тесты.

В статье [45] в качестве оценки качества работы языковой модели используется мера её скалиброванности, то есть свойства предсказывать вероятность корректности собственного ответа. Для хорошо скалиброванных моделей в качестве оценки корректности сгенерированного кода предлагается использовать непосредственно вероятность этого кода как последовательности токенов с точки зрения самой языковой модели. Для оценки скалиброванности авторы [45] применяют метрики ECE (Expected Calibration Error), Brier Score и Skill Score. При этом корректность самих программ оценивается как по тестам (Pass@k), так и по эталонному решению (Exact-Match). Стоит отметить, что настоящая работа нацелена на исследование метрик корректности кода напрямую, тогда как идея оценки скалиброванности модели ортогональна понятию корректности программ, а конкретная метрика качества кода, на которой базируется метод оценки скалиброванности, при этом не фиксируется в статье [45].

В условиях отсутствия тестов и канонических решений оценивать корректность генерируемых программ необходимо иными способами. В работе [46] вводится понятие некогерентности – вероятности того, что две случайные сгенерированные моделью программы дают разные ответы (или демонстрируют принципиально разное поведение) на случайных входных данных. Ввиду теоремы Райса [47] прямое вычисление корректности по введённому определению в общем случае невозможно, поэтому авторы [46] оценивают корректность через оценку некогерентности с помощью конечного множества сгенерированных программ и входных данных. При этом вероятностное распределение программ задаётся исследуемой языковой моделью, а распределение входов – непосредственно авторами. Определение корректности кода, на котором построено исследование [46], также легло в основу настоящей работы, а идея оценки корректности генерируемого кода на базе конечного множества программ и тестов реализована иным образом, описанным далее в разделе 2.

Метрика Pass@k описана в статье [30] совместно с набором данных HumanEval, содержащим 164 задачи на языке Python, собранные и размеченные вручную. Каждая задача сопровождается фрагментом кода на Python, содержащим сигнатуру целевой функции, которую требуется реализовать, и документирующую строку с описанием функционала и примерами тестов. В среднем каждая задача покрыта 7-8 тестами, чего оказывается недостаточно для полноценного тестирования программ даже на простых задачах из набора HumanEval. Будучи одним из наиболее популярных наборов данных в области кодогенерации, HumanEval также переведён на десяток других языков, включая Java, C#, Golang, Kotlin и TypeScript [48]. В рамках фреймворка EvalPlus [49], основанного на EvalPerf [29], построены наборы данных HumanEval+ и MBPP+, содержащие задачи из оригинальных наборов HumanEval [30] и MBPP [50] со значительно расширенным тестовым покрытием. В настоящей работе для исследования корректности программ на языке Python, решающих простые алгоритмические задачи, используется адаптированная версия набора данных HumanEval+. Отметим, что выводы, полученные в результате экспериментов, могут не обобщаться на иные языки программирования или типы задач.

Широкое применение наборов данных HumanEval [30] и MBPP [50], а также их расширенных версий HumanEval+ и MBPP+ [49] в современных исследованиях сопряжено с риском утечки тестовых данных в тренировочные выборки языковых моделей, поскольку задачи из этих наборов данных могут быть частью открытых наборов, используемых при обучении. Несмотря на это, оригинальные работы, в которых представлены языковые модели Phi-1 [8], Phi-2 [51-52], Phi-3-mini-4k [53], Phi-4-mini [50-51] и Qwen2.5-Coder [52-53], рассматриваемые в настоящем исследовании, содержат значения метрики Pass@k (как правило, Pass@1), вычисленной авторами на этих данных. Согласно результатам из перечисленных работ, лишь при оценке моделей Phi-1 [8] и Qwen2.5-Coder [52-53] были проведены дополнительные исследования качества работы модели в условиях возможной утечки данных HumanEval в обучающую выборку. Однако часть этих исследований проведена с использованием другой большой языковой модели для оценки качества работы предложенной модели, а также небольшого набора новых задач, составленных и размеченных авторами вручную. В настоящей статье соответствующие риски дополнительно не рассматриваются, а условия задач оставлены в оригинальном виде для возможности сравнения результатов экспериментов с другими научными работами. Выбор моделей для проведения экспериментов обоснован, с одной стороны, имеющимися вычислительными ресурсами, а с другой, – целью исследования поведения метрик корректности генерируемого кода в зависимости от качества работы модели.

В данной работе описано формальное теоретико-вероятностное определение корректности кода, основанное на подходе из работы [46]. Исходя из этого определения введена метрика корректности сгенерированного кода TSA (Test Suite Accuracy), позволяющая сравнивать языковые модели и ранжировать их по качеству более точно, чем метрика Pass@1.

Вклад работы состоит в следующем:

- введена новая метрика корректности кода TSA на основе теоретико-вероятностного определения корректности программы, предложенного в работе [46];
- собран набор данных HumanEval++, основанный на наборе HumanEval+ и содержащий разбиение тестов на группы, что позволяет вычислять метрику TSA для оценки качества работы моделей на этом наборе данных;
- проведены эксперименты с пятью языковыми моделями Phi-1 [8], Phi-2 [51-52], Phi-3-mini-4k (далее Phi-3) [53], Phi-4-mini (далее Phi-4) [50-51] и Qwen2.5-Coder [52-53], подтверждающие теоретические свойства TSA и Pass@1 в рассмотренной формализации, а также проведён анализ полученных результатов.

Дальнейшее изложение имеет следующую структуру. Раздел 2 содержит описание метрики TSA и теоретического базиса, лежащего в основе рассматриваемого вероятностного определения корректности кода. В этом же разделе сформулирована метрика Pass@1 в рамках описанной формализации понятия корректности кода, а также приведены результаты анализа свойств метрик TSA и Pass@1. В разделе 3 рассматриваются детали подготовки и результаты экспериментов, подтверждающих описанные свойства метрик.

2. Теоретические основы исследования

В данном разделе приводится определение корректности кода, аналогичное определению из работы [46] и лежащее в основе исследования, а также описание метрики TSA и её статистических свойств.

2.1 Корректность кода

Пусть $\mathcal{I}$ – множество входов программы, а $\mathcal{O}$ – множество выходов. *Программой* будем называть функцию $\pi: \mathcal{I} \mapsto \mathcal{O}$, осуществляющую отображение входов в выходы. В действительности программа π может зацикливаться или завершаться неуспешно на некотором подмножестве входов. Корректность кода определяется с учётом этого факта, но допустимо считать такие результаты работы программы элементами множества $\mathcal{O}$: это не повлияет на определение корректности.

Поскольку программы π, рассматриваемые в данном исследовании, являются результатом генерации большими языковыми моделями, то множество возможных программ $\mathcal{P} \ni \pi$ является вероятностным пространством. Введём также вероятностное распределение на множестве входов $\mathcal{I}$: оно может отражать практические знания автора задачи или программы об ожидаемых и невозможных входах. Определим *степень надёжности* $r(\pi)$ программы π как вероятность $r(\pi) = \mathbb{P}\big(\pi(i) = o(i)\big)$ того, что она возвращает правильный ответ $o(i)$ на случайном входе i. Если задача допускает множество $O(i)$ различных верных ответов для каждого входа i, то определение степени надёжности корректируется соответственно: $r(\pi) = \mathbb{P}\big(\pi(i) \in O(i)\big)$. В данной работе именно степень надёжности программы используется в качестве оценки *корректности* программы. Абсолютно корректной является программа со степенью надёжности равной 1.

Отметим, что для заданной задачи множества $\mathcal{I}$, $\mathcal{O}$ и вероятностное распределение на множестве $\mathcal{I}$ постоянны, а значит для заданной программы π её степень надёжности $r(\pi)$ также постоянна. Однако для её вычисления требуется определить результат программы π на всех входах из множества $\mathcal{I}$ с ненулевой вероятностью, что почти никогда не реализуемо на практике, поскольку множество $\mathcal{I}$ может иметь бесконечную мощность. Для решения этой проблемы рассмотрим разбиение множества $\mathcal{I}$ на конечное число множеств I_k, $k \in \overline{1,s}$ – *группы входов*, для которых $\mathbb{P}(I_k) = w_k > 0$. Суть разбиения состоит в выделении независимых групп входов, отвечающих различным требованиям к программе в соответствии с условием задачи. Для оценки корректности программы π строится конечное множество *тестов* $C \subset \mathcal{I}$, состоящее из *групп тестов* $C_k = C \cap \mathcal{I} \neq \emptyset$, $k \in \overline{1,s}$. Каждая группа тестов C_k должна содержать не менее одного теста $n_k = |C_k| \geq 1$, то есть C должно содержать некоторую трансверсаль множеств I_k, $k \in \overline{1,s}$. Это свойство множества тестов C будем называть *полнотой тестового покрытия*: оно необходимо для верной оценки корректности программы.

2.2 Метрика TSA

Для оценки корректности программы π необходимо вычислить результат её работы лишь на конечном множестве тестов C. Пусть $i_{k_j} \in \mathcal{I}$ – j-ый тест из множества C_k, а $\mathbb{P}\left(i_{k_j}\right) = \hat{p}_{k_j}$.

Вероятность группы тестов обозначим через $\widehat{w}_k = \mathbb{P}(C_k) = \sum_{j=1}^{n_k} \hat{p}_{k_j} > 0$. Пусть для сгенерированной программы $\pi \in \mathcal{P}$ вероятность успешного прохождения теста i_{k_j} равна $p_{k_j} = \mathbb{P}\left(\pi\left(i_{k_j}\right) = o\left(i_{k_j}\right)\right)$. Для фиксированной программы π это значение всегда равно 1 или 0, но в условиях генерации программы π из распределения на множестве $\mathcal{P}$ индикатор события успешного прохождения теста становится случайной величиной X_{k_j}, имеющей распределение Бернулли $\mathrm{Be}\left(p_{k_j}\right)$.

Рассмотрим случайную величину

$$\mathrm{TSA}_\pi = \sum_{k=1}^{s} \left[\frac{w_k}{\widehat{w}_k} \cdot \sum_{j=1}^{n_k} X_{k_j} \hat{p}_{k_j} \right].$$

Для заданной программы π функция TSA оценивает степень её надёжности с помощью полного конечного тестового покрытия C. Для практического вычисления этой функции требуется также задать вероятности отдельных тестов $\hat{p}_{k_j}$ и групп входов w_k. Построенная функция является оценкой корректности π и называется **TSA** (англ. *Test Suite Accuracy* – точность на группах тестов).

При рассмотрении конкретной задачи и фиксировании множества тестов значение TSA постоянно для каждой программы π, однако при генерации случайных программ $\pi \in \mathcal{P}$ функция TSA становится случайной величиной, принимающей значения из отрезка $[0;\ 1]$, с дискретным распределением. Для оценки средней надёжности программ, генерируемых конкретной языковой моделью, может использоваться математическое ожидание TSA, оцениваемое средним значением TSA для нескольких программ π_l, $l \in \overline{1,t}$. Это усреднение равносильно оценке вероятностей p_{k_j} путём усреднения индикаторов успешности прохождения конкретного теста сгенерированными программами и последующему вычислению математического ожидания TSA с оцененными значениями $\mathbb{E}\, X_{k_j} = p_{k_j}$, поскольку функция TSA линейна относительно X_{k_j}. Если же рассматриваются различные задачи (а значит и множества $\mathcal{I}$, $\mathcal{O}$, C и соответствующие вероятностные распределения), то соответствующие функции TSA являются случайными величинами на разных вероятностных пространствах, поэтому полученные значения TSA для каждой задачи можно также усреднить по всему набору задач для оценки общей способности нейросети к генерации корректного кода.

2.3 Анализ TSA

Исследуем статистические свойства метрики TSA. Поскольку функция TSA линейна относительно X_{k_j}, то математическое ожидание

$$\mathbb{E}\ \mathrm{TSA}_\pi = \sum_{k=1}^{s} \left[\frac{w_k}{\widehat{w}_k} \cdot \sum_{j=1}^{n_k} p_{k_j} \hat{p}_{k_j} \right] = \sum_{k=1}^{s} \left[\frac{w_k}{\widehat{w}_k} \cdot s_k \right].$$

Величина $\frac{s_k}{\widehat{w}_k}$, где $s_k = \sum_{j=1}^{n_k} p_{k_j} \hat{p}_{k_j}$, равна оценке вероятности успешного прохождения случайного теста из группы C_k случайной программой $\pi \in \mathcal{P}$. Так, метрика TSA основана на оценках степени надёжности программы на группах входов I_k, полученных на конечных группах тестов $C_k \subset I_k$.

Для вычисления дисперсии TSA сделаем несколько предположений. Первое предположение заключается в независимости случайных величин X_{k_j} при различных k. На практике это означает, что соответствие случайной программы требованиям из условия задачи, заложенным в тестах из разных групп входов, независимо. Иными словами, соответствие

программы одному требованию из условия задачи не влияет на её соответствие другим требованиям. Второе предположение состоит в том, что тесты в каждой группе C_k упорядочены по возрастанию сложности. Формально это ограничение выражается неравенствами

$$p_{k_{j_1}} \geq p_{k_{j_2}}, \qquad X_{k_{j_1}} \geq X_{k_{j_1}} \ \forall\, 1 \leq j_1 < j_2 \leq n_k.$$

Это требование можно интерпретировать следующим образом: если программа π не прошла тест $i_{k_{j_1}}$, то на тесте $i_{k_{j_2}}$ её результат также окажется неверным с вероятностью 1. При этом случаи генерации программ, проходящих более сложные тесты, но не проходящих более простые, исключаются из рассмотрения как маловероятные. Отметим, что эти требования лишь упрощают вычисление дисперсии TSA, однако не являются частью определения метрики, поэтому могут не учитываться в её практических реализациях. Например, для многих задач тесты в действительности имеют одинаковую сложность в смысле рассматриваемого ограничения.

С учётом описанных предположений, значение дисперсии TSA оценивается на основе ковариации случайных величин X_{k_j} следующим образом.

$$\operatorname{cov}\left(X_{k_{j_1}}, X_{k_{j_2}}\right) = \mathbb{E}\, X_{k_{j_1}} X_{k_{j_2}} - \mathbb{E}\, X_{k_{j_1}} \cdot \mathbb{E}\, X_{k_{j_2}} = p_{k_{j_2}} - p_{k_{j_1}} p_{k_{j_2}}, \qquad j_1 < j_2$$

$$\mathbb{D}\left[\sum_{j=1}^{n_k} X_{k_j} \hat{p}_{k_j}\right] = \sum_{j_1=1}^{n_k}\sum_{j_2=1}^{n_k} \hat{p}_{k_{j_1}} \hat{p}_{k_{j_2}} \operatorname{cov}\left(X_{k_{j_1}}, X_{k_{j_2}}\right) =$$

$$= \sum_{j_1=1}^{n_k}\sum_{j_2=1}^{n_k} \hat{p}_{k_{j_1}} \hat{p}_{k_{j_2}} \left[\min\left\{p_{k_{j_1}}, p_{k_{j_2}}\right\} - p_{k_{j_1}} p_{k_{j_2}}\right] \leq$$

$$\leq \sum_{j_1=1}^{n_k}\sum_{j_2=1}^{n_k} \hat{p}_{k_{j_1}} \hat{p}_{k_{j_2}} \left[p_{k_{j_2}} - p_{k_{j_1}} p_{k_{j_2}}\right] = \sum_{j=1}^{n_k} \hat{p}_{k_j} p_{k_j} \cdot \sum_{j=1}^{n_k} \hat{p}_{k_j} \left(1 - p_{k_j}\right) = (\widehat{w}_k - s_k) \cdot s_k$$

$$\mathbb{D}\ \mathrm{TSA}_\pi = \sum_{k=1}^{s} \mathbb{D}\left[\frac{w_k}{\widehat{w}_k} \cdot \sum_{j=1}^{n_k} X_{k_j} \hat{p}_{k_j}\right] \leq \sum_{k=1}^{s} \frac{w_k^2}{\widehat{w}_k^2} \cdot (\widehat{w}_k - s_k) \cdot s_k$$

Рассмотрим несколько частных случаев.

Следствие 1. Пусть все тесты в каждой группе тестов равновероятны, то есть $\hat{p}_{k_j} = \hat{p}_k\ \forall\, j \in \overline{1, n_k}$. Тогда

$$\mathrm{TSA}_\pi = \sum_{k=1}^{s}\left[w_k \cdot \frac{\sum_{j=1}^{n_k} X_{k_j}}{n_k}\right], \qquad \mathbb{E}\ \mathrm{TSA}_\pi = \sum_{k=1}^{s}\left[w_k \cdot \frac{\sum_{j=1}^{n_k} p_{k_j}}{n_k}\right],$$

$$\mathbb{D}\ \mathrm{TSA}_\pi \leq \sum_{k=1}^{s}\left[w_k^2 \cdot \frac{\sum_{j=1}^{n_k} p_{k_j} \cdot \left(n_k - \sum_{j=1}^{n_k} p_{k_j}\right)}{n_k^2}\right],$$

то есть метрика TSA равна взвешенной сумме среднего числа пройденных тестов в каждой группе.

Следствие 2. Пусть в каждой группе содержится всего один тест, то есть $n_k = 1\ \forall\, k \in \overline{1, s}$. Тогда

$$\mathrm{TSA}_\pi = \sum_{k=1}^{s} w_k X_k\,, \qquad \mathbb{E}\ \mathrm{TSA}_\pi = \sum_{k=1}^{s} w_k p_k\,, \qquad \mathbb{D}\ \mathrm{TSA}_\pi \leq \sum_{k=1}^{s} w_k^2 p_k (1 - p_k),$$

то есть метрика TSA равна взвешенному среднему числу пройденных тестов, при этом все тесты считаются независимыми.

Следствие 3. Если в следствии 1 положить все группы входов равновероятными, то есть $w_k = \frac{1}{s}\ \forall\, k \in \overline{1, s}$, то получим

$$\mathrm{TSA}_\pi = \frac{1}{s}\sum_{k=1}^{s} \frac{X_k}{n_k}, \qquad \mathbb{E}\ \mathrm{TSA}_\pi = \frac{1}{s}\sum_{k=1}^{s} \frac{p_k}{n_k}, \qquad \mathbb{D}\ \mathrm{TSA}_\pi \leq \frac{1}{s^2}\sum_{k=1}^{s}\left[\frac{\sum_{j=1}^{n_k} p_{k_j} \cdot \left(n_k - \sum_{j=1}^{n_k} p_{k_j}\right)}{n_k^2}\right],$$

то есть метрика TSA равна среднему числу пройденных тестов в каждой группе.

Следствие 4. Если в следствии 3 предположить, что все группы тестов содержат одинаковое число тестов, то есть $n_k = n\ \forall\, k \in \overline{1, s}$, то получим

$$\mathrm{TSA}_\pi = \frac{1}{ns}\sum_{k=1}^{s} X_k\,, \qquad \mathbb{E}\ \mathrm{TSA}_\pi = \frac{1}{ns}\sum_{k=1}^{s} p_k\,, \qquad \mathbb{D}\ \mathrm{TSA}_\pi \leq \frac{1}{n^2 s^2}\sum_{k=1}^{s}\left[\sum_{j=1}^{n} p_{k_j} \cdot \left(n - \sum_{j=1}^{n} p_{k_j}\right)\right],$$

то есть метрика TSA равна среднему числу пройденных тестов.

Следствие 5. Наконец, объединим условия из предыдущих следствий, то есть предположим, что в каждой группе содержится всего один тест и все тесты равновероятны. Тогда

$$\mathrm{TSA}_\pi = \frac{1}{s}\sum_{k=1}^{s} X_k\,, \qquad \mathbb{E}\ \mathrm{TSA}_\pi = \frac{1}{s}\sum_{k=1}^{s} p_k\,, \qquad \mathbb{D}\ \mathrm{TSA}_\pi \leq \frac{1}{s^2}\sum_{k=1}^{s} p_k(1 - p_k),$$

то есть метрика TSA равна среднему числу пройденных тестов, при этом все тесты считаются независимыми.

Пусть с помощью одной языковой модели получено t различных программ π_l, $l \in \overline{1, t}$, решающих одну и ту же задачу и тестирующихся на одном и том же полном наборе тестов C. Тогда сами программы можно считать независимыми, а значения их метрик TSA_{π_l} – независимыми и одинаково распределёнными случайными величинами. Ввиду конечности их математического ожидания и дисперсии по центральной предельной теореме

$$\frac{\sum_{l=1}^{t} \mathrm{TSA}_{\pi_l}}{t} \xrightarrow{d} \mathcal{N}(\mathbb{E}\ \mathrm{TSA}_\pi, \mathbb{D}\ \mathrm{TSA}_\pi),$$

где математическое ожидание и дисперсия могут быть оценены по формулам выше.

2.4 Метрика Pass@1

Теперь рассмотрим свойства метрики Pass@1 [30] в рамках описанного вероятностного определения корректности кода. После введения случайных величин X_{k_j} аналогично тому, как это сделано для метрики TSA, метрика Pass@1 выражается в виде произведения

$$\mathrm{Pass@1}_\pi = \prod_{k=1}^{s}\prod_{j=1}^{n_k} X_{k_j} = \prod_{k=1}^{s} X_{k_{n_k}} = \prod_{k=1}^{s} X_k,$$

поскольку прохождение всех тестов в группе C_k равновероятно прохождению наиболее сложного теста в этой группе, согласно одному из предположений для вычисления дисперсии TSA. Нетрудно вычислить математическое ожидание и дисперсию метрики:

$$\mathbb{E}\ \mathrm{Pass@1}_\pi = \prod_{k=1}^{s} p_{k_{n_k}} = \prod_{k=1}^{s} p_k\,, \qquad \mathbb{D}\ \mathrm{Pass@1}_\pi = \prod_{k=1}^{s} p_k \cdot \left(1 - \prod_{k=1}^{s} p_k\right).$$

2.5 Анализ дисперсии метрик TSA и Pass@1

Пусть $p_{k_j} \leq p_{k_1} \leq p\ \forall\, k, j$. Оценим сверху дисперсию каждой из метрик корректности кода.

$$\widehat{w}_k - s_k \le \widehat{w}_k, \qquad s_k \le p\widehat{w}_k, \qquad (\widehat{w}_k - s_k)\cdot s_k \le \left(\frac{\widehat{w}_k^2}{2}\right)^2, \qquad \sum_{k=1}^{s} w_k^2 \in \left[\frac{1}{s};\ 1\right]$$

$$\mathbb{D}\ \mathrm{TSA}_\pi \le \sum_{k=1}^{s} \frac{w_k^2}{\widehat{w}_k^2}\cdot(\widehat{w}_k - s_k)\cdot s_k \le \beta \sum_{k=1}^{s} w_k^2,$$

где $\beta = p\cdot(1-p)$ при $p < 0.5$, иначе $\beta = 0.25$. В предположении, что $w_k = \frac{1}{s}\ \forall\, k \in \overline{1,s}$, достигается нижняя оценка суммы $\sum_{k=1}^{s} w_k^2 = \frac{1}{s}$, и оценка дисперсии улучшается: $\mathbb{D}\ \mathrm{TSA}_\pi \le \frac{\beta}{s}$.

Аналогичным образом оценивается дисперсия Pass@1: $\mathbb{D}\ \mathrm{Pass@1}_\pi \le \beta$, где $\beta = p^s\cdot(1-p^s)$ при $p^s < 0.5$, иначе $\beta = 0.25$. Отметим, что эта оценка не зависит от числа тестов в группах и от весов каждой группы входов w_k, поэтому, в отличие от оценки дисперсии TSA, она не улучшается в случае равновероятных групп входов.

Зафиксируем значение α и для каждой из метрик определим количество групп входов s, при котором дисперсия гарантированно не превысит α. Для метрики TSA в случае одинаковых весов у всех групп входов получаем оценку $s > \frac{\beta}{\alpha} \ge \frac{1}{4\alpha}$. Для метрики Pass@1 при $\alpha \ge 0.25$ подходит любое s; при меньших α в случае $p = 1$ не существует гарантированно подходящих s, но при $0 < p < 1$ при $p^s < 0.5 \Leftrightarrow s > -\log_p 2$ подойдут значения $s > \log_p \frac{1-\sqrt{1-4\alpha}}{2}$.

Для фиксированных значений s и α также можно определить минимальное количество генерируемых программ t, при котором дисперсия усреднённой метрики этих программ в соответствии с центральной предельной теоремой не превысит заданного порога α. При неизвестном p в случае TSA имеет место оценка $t > \frac{1}{4\alpha s}$, а для Pass@1 справедливо $t > \frac{1}{4\alpha}$.

Отметим, что содержательно введённый параметр p является аналогом параметра base-rate p_r из работы [45] для вычисления Unskilled Reference Brier Score и для конкретной модели может быть определён аналогичным образом на практике.

Полученные оценки дисперсии приводят к следующим выводам.

- Если значение $p << 1$ (языковая модель заведомо плохо справляется с написанием корректного кода) или в случае выбора Pass@1 имеет место оценка $s > -\log_p 2$ (независимых групп тестов достаточно много), то оценка дисперсии обеих метрик заметно улучшается. При этом дисперсия Pass@1 убывает экспоненциально с убыванием p, а дисперсия TSA – лишь квадратично. Это вызвано тем, что значение метрики Pass@1 равно нулю для программ, не являющихся абсолютно корректными, тогда как значения TSA находятся на интервале $(0;\ 1)$ и позволяют сравнивать между собой слабые модели.
- Если значение p близко к единице (языковая модель хорошо справляется с написанием корректного кода), то при выборе одинаковых весов для всех групп тестов оценка дисперсии и минимального числа программ t для метрики TSA оказывается лучше, чем для Pass@1. Это свойство позволяет использовать TSA для более точной оценки способностей к написанию корректного кода качественными языковыми моделями за счёт меньшего числа запусков моделей и, как следствие, большей экономии ресурсов.
- Если значение p не известно или консервативно полагается равным 1, то дисперсия метрики TSA по-прежнему убывает с ростом числа независимых групп тестов s, тогда как оценка дисперсии Pass@1 не улучшается.

3. Эксперименты

Для анализа поведения метрик корректности кода Pass@1 и TSA проведены эксперименты с использованием кода, сгенерированного пятью языковыми моделями. Генерация и тестирование кода выполнены на основе набора данных HumanEval++, построенного в рамках настоящего исследования. В данном разделе описаны результаты проведённых экспериментов. Сгенерированные программы, построенный набор задач и код тестирующей среды доступны в репозитории на GitLab [58].

3.1 Построение набора данных

Одной из ключевых особенностей метрики TSA является необходимость в полном тестовом покрытии и разбиении тестов на независимые группы, соответствующие различным требованиям из условия задачи. Существующие открытые наборы данных задач, предназначенные для обучения и тестирования языковых моделей, не предоставляют разбиение тестов на группы по данному принципу. В данной работе за основу взят набор HumanEval+ [49], содержащий 164 задачи, покрытые более чем 125000 тестов. Выбор данных обоснован популярностью набора HumanEval [30] и количеством тестов в HumanEval+, достаточно для получения точных оценок качества кода.

Оригинальный набор HumanEval+ представлен в текстовом формате JSONL, где каждая строка является JSON объектом с описанием условия задачи и тестирующего кода на Python. В рамках данного исследования набор данных был разбит на отдельные файлы, соответствующие различным задачам и содержащие условие задачи на языке Python, код канонического решения, текстовые файлы с входными и выходными данными тестов (по одному тесту в строке). Описание условий задач и тестов приведено к единому формату для всех задач набора, что упрощает обработку сгенерированных решений на этапе тестирования. Тесты каждой задачи нумеруются с 1, а их разбиение на группы описывается отдельным файлом в формате YAML. Разбиение тестов на группы произведено автоматически путём классификации входных и выходных данных; код каждого классификатора подготовлен и верифицирован вручную. В результате классификации тестов каждой задаче соответствует от 3 до 8 групп тестов. Код классификатора для каждой задачи также стал частью нового набора HumanEval++ [59].

Полученный набор является расширяемым по множеству задач и тестов и разбиению тестов на группы. Формат описания файлов, необходимых для воспроизведения тестирования и расширения набора задач, описан в репозитории проекта на GitLab [58].

3.2 Генерация кода

Для генерации кода, решающего задачи из набора HumanEval++, использовались 5 языковых моделей: Phi-1 [8], Phi-2 [51-52], Phi-3 [53], Phi-4 [54-55] и Qwen2.5-Coder [56-57]. Каждая из моделей запускалась 5 раз для последующего анализа дисперсии метрик качества кода. Число весов моделей и ключевые параметры генерации приведены в табл. 1.

Табл. 1. Языковые модели и параметры генерации.
Table 1. Language models and generation parameters.

Модель	Число весов	Температура τ	top-k	top-p	max_new_tokens
Phi-1	$1.3 \cdot 10^9$	0.3	10	0.95	512
Phi-2	$2.7 \cdot 10^9$				
Phi-3	$3.8 \cdot 10^9$				
Phi-4	$3.8 \cdot 10^9$				
Qwen2.5-Coder	$7.6 \cdot 10^9$				

Для автоматизации тестирования языковых моделей в задаче генерации кода критически важно контролировать формат ответа нейросети. Основным инструментом управления этим форматом, помимо непосредственно параметров генерации, служит текст запроса модели. В контексте решения задач из HumanEval недостаточно точный запрос может приводить к генерации пустого тела функции, новых функций после или вместо требуемой, комментариев с пометкой TODO, лишнего кода тестов и примеров запуска функции, лишнего текста на естественном языке, текста в формате Markdown или даже корректного, но закомментированного кода. Для решения перечисленных проблем в рамках проведённых экспериментов использовался следующий запрос.

> You are an experienced Python developer.
> Given the Python code snippet below, implement the following function to solve the problem described in the docstring.
>
> Do NOT do the following actions:
> - change the problem statement in any way
> - duplicate the function definition or change its name or signature which are already provided in the task below
> - generate other global functions
> - generate any kind of explanations, including markdown-style text
> - use comments containing Python code
> - leave all kinds of TODO-like comments
> - leave an empty function body (including pass statements or comments)
> - write tests for your own code
> - print anything to stdout or stderr
>
> You are ONLY allowed to implement the given function below.
> The function prototype is already provided in the statement below, you only need to fill in the function body.
> Stop generating code as soon as the given function definition ends.
> You may use Python standard library, but don't forget to import the necessary libraries in that case.

После генерации полного ответа нейросети к нему последовательно применялись следующие преобразования для извлечения и нормализации кода.

- текст запроса удаляется из начала ответа;
- извлекается последний фрагмент на языке Markdown, начинающийся с тега ```python, если такой фрагмент имеется;
- удаляются строки кода, следующие после определения целевой функции, не имеющие отступа в начале и не являющиеся объявлениями функции;
- если тело целевой функции пустое и не содержит ничего, кроме, возможно, документирующей строки с условием задачи, то оно заменяется телом следующей функции при её наличии;
- если после объявления целевой функции имеются описания других функций, то последнее из описаний удаляется во избежание синтаксических ошибок, связанных с ограничением на число генерируемых токенов;
- если последний многострочный блок с документацией функции не завершён, то он удаляется;

- строки вида <YOUR CODE HERE> заменяются комментариями на языке Python;
- в конструкциях вида return" и if" восстанавливаются пробелы: return ' ' и if ' '.

Скрипт для генерации кода с помощью указанных языковых моделей и тексты сгенерированных программ до и после применения перечисленных эвристик расположены в репозитории проекта на GitLab [58].

3.3 Результаты тестирования

На этапе запуска полученного кода целевая функция в каждой задаче импортируется из соответствующего модуля, сгенерированного нейросетью, с помощью библиотеки importlib. Запуск функции на тестах производится в отдельном потоке, при этом для каждого теста установлено ограничение на максимальное время работы функции, равное 10 секундам, при превышении которого поток останавливается, а оставшиеся тесты текущей группы считаются непройденными. Аналогичное ограничение установлено на время прохождения всех тестов задачи, равное 20 секундам, при превышении которого задача считается полностью нерешённой. Для запуска тестов и вычисления метрик корректности кода разработана утилита командной строки, позволяющая настраивать некоторые аспекты тестирования, включая остановку запуска оставшихся тестов в группе после первой ошибки на тесте. Код утилиты и описание формата результатов её работы приведены в репозитории проекта на GitLab [58].

В табл. 2 приведены результаты работы всех пяти моделей и статистика значений соответствующих метрик качества кода, вычисленных на всех пяти фазах генерации для всех 164 задач набора данных. Выполнение тестов в каждой группе останавливалось после получения первой ошибки; веса групп тестов полагались равными. Значения метрик отмасштабированы на отрезок $[0;\ 100]$, для каждой метрики вычислено среднее значение, медиана, абсолютное (σ) и относительное (RSD) стандартное отклонение и доверительные интервалы с уровнем доверия 95% и 99%. Доверительные интервалы построены на основе распределения t-Стьюдента с 819 степенями свободы.

Табл. 2. Метрики качества кода, вычисленные для 5 моделей по всему набору данных.
Table 2. Code quality metrics for 5 models over the whole dataset.

Метрика		Модель				
		Phi-1	Phi-2	Phi-3	Phi-4	Qwen2.5-Coder
TSA	Среднее	25.98	13.25	12.94	49.36	85.92
	Медиана	0.00	0.00	0.00	40.00	99.99
	σ	37.29	25.49	25.72	43.79	28.14
	RSD	1.44	1.92	1.99	0.89	0.33
	95% ДИ	[23.4; 28.5]	[11.5; 15.0]	[11.2; 14.7]	[46.4; 52.4]	[84.0; 87.8]
	99% ДИ	[22.6; 29.3]	[11.0; 15.5]	[10.6; 15.3]	[45.4; 53.3]	[83.4; 88.5]
Pass@1	Среднее	15.85	4.76	5.00	37.20	74.63
	Медиана	0.00	0.00	0.00	0.00	100.00
	σ	36.52	21.28	21.79	48.33	43.51
	RSD	2.30	4.48	4.36	1.30	0.58
	95% ДИ	[13.4; 18.4]	[3.3; 6.2]	[3.5; 6.5]	[33.9; 40.5]	[71.7; 77.6]
	99% ДИ	[12.6; 19.1]	[2.8; 6.7]	[3.0; 7.0]	[32.8; 41.6]	[70.7; 78.6]

Полученные значения метрик на всём наборе данных позволяют сделать такие выводы:

- Задачи в исследуемом наборе довольно разнородны по сложности для используемых языковых моделей. Об этом свидетельствуют высокие значения абсолютного и относительного стандартного отклонения обеих метрик для всех моделей, кроме Qwen2.5-Coder – самой сильной модели среди представленных. Значения метрик TSA и Pass@1 для указанных моделей и задач имеют большой разброс на отрезке [0; 1].
- Для моделей с высоким качеством работы (Phi-4 и Qwen2.5-Coder) дисперсия TSA меньше дисперсии Pass@1, и с ростом качества разность дисперсий увеличивается. Высокое качество генерируемого кода соответствует большим значениям p верхней оценки вероятности прохождения произвольного теста случайной сгенерированной программой. Значения относительного стандартного отклонения для TSA всегда ниже, чем у Pass@1, что объясняется более высоким значением среднего TSA, поскольку TSA всегда не меньше, чем Pass@1.
- Для моделей с низким качеством работы (Phi-2 и Phi-3) значения TSA с меньшей скоростью убывают к нулю по сравнению с Pass@1. Это позволяет сравнивать между собой такие модели на задачах, где значение Pass@1 равно нулю.
- Модели Phi-2 и Phi-3 упорядочиваются разным образом по метрикам TSA и Pass@1. Это объясняется тем, что программы, сгенерированные Phi-2, в ряде случаев проходят некоторое количество тестов и являются по крайней мере синтаксически корректными, тогда как модель Phi-3 сумела полностью решить большее число задач, но не сгенерировала достаточно много частичных решений остальных задач, чтобы превзойти Phi-2 по метрике TSA. Отметим, что для этих двух моделей доверительные интервалы с уровнем доверия 95% и 99% для обеих метрик сильно пересекаются (мера Жаккара выше 0.84 во всех четырёх случаях), то есть ранжирование моделей по каждой из метрик не обладает высокой степенью достоверности, и качество работы Phi-2 и Phi-3 можно полагать примерно одинаковым.

Табл. 3 содержит информацию о распределении абсолютного и стандартного отклонения каждой метрики, вычисленных по 5 сгенерированным программам для каждой задачи.

Табл. 3. Среднее стандартное отклонение метрик качества кода для каждой задачи в данных.
Table 3. Mean standard deviation of code quality metrics over each task in the dataset.

Метрика		Модель				
		Phi-1	Phi-2	Phi-3	Phi-4	Qwen2.5-Coder
TSA	σ	12.73	11.13	9.12	12.95	8.77
	RSD	0.49	0.59	0.39	0.33	0.15
Pass@1	σ	9.49	7.27	7.49	12.29	12.07
	RSD	0.25	0.29	0.28	0.26	0.26

Для моделей с низким качеством работы (Phi-1, Phi-2 и Phi-3) дисперсия TSA оказывается выше, чем у Pass@1. Для модели Phi-4 со средним качеством генерируемого кода дисперсия обеих метрик примерно одинакова, а для Qwen2.5-Coder метрика TSA обладает сильно меньшей дисперсией, чем Pass@1. Это объясняется тем, что для программ с высокой степенью надёжности, не проходящих лишь небольшую часть тестов, покрывающих, как правило, краевые случаи, метрика Pass@1 принимает значение 0, тогда как TSA останется высоким, как и для абсолютно корректных программ. При этом модели, генерирующие качественный код, склонны чаще выдавать абсолютно корректный и почти корректный код. Так, для качественных моделей метрика TSA позволяет получить более точную оценку

корректности кода, а для моделей с низким качеством работы TSA даёт возможность более консервативного сравнения и ранжирования моделей на основе средних значений метрики и доверительных интервалов, нежели Pass@1.

Значение и стабильность метрики TSA зависят от ряда дополнительных условий проведения эксперимента: конкретного разбиения тестов на группы, весов групп тестов и конкретного порядка тестов в группах. Напомним, что предположение об упорядоченности тестов по сложности внутри группы не требуется для вычисления метрики TSA, а лишь упрощает оценку её дисперсии в теоретических выкладках. Для исследования зависимости значений метрики TSA от перечисленных условий проведены дополнительные эксперименты, предполагающие варьирование следующих параметров:

- Остановка при первой ошибке: если программа не проходит некоторый тест в группе, то остальные тесты этой группы считаются непройденными. Этот параметр отражает предположение об упорядоченности тестов по сложности.
- Веса групп тестов могут полагаться как равными, так и пропорциональными размеру группы. Последняя опция связана с предположением об источнике тестов в HumanEval+, задающем необходимое распределение входных данных.
- Разбиение на группы тестов может быть опциональным: в случае использования одной группы для всех тестов в каждой задаче метрика TSA вырождается в среднее число пройденных тестов. Также исследуется разбиение, в котором сохранены две группы, наиболее популярные по числу тестов в текущем разбиении в HumanEval++, а третью группу составляют все остальные тесты.

В табл. 4-8 приведены результаты дополнительных экспериментов.

Табл. 4. Среднее значение и стандартное отклонение метрики TSA для модели Phi-1 при разных параметрах вычисления TSA.
Table 4. Mean and standard deviation of TSA for Phi-1 under different experimental conditions.

Параметры вычисления TSA		Число групп тестов		
Остановка при первой ошибке	Веса групп тестов	Все	3 наибольших	1
Да	Равные	(25.98, 37.29)	(22.28, 37.40)	(16.10, 36.47)
	Пропорциональные	(21.80, 38.61)	(21.23, 38.48)	
Нет	Равные	(28.70, 38.32)	(26.71, 38.71)	(24.79, 39.72)
	Пропорциональные	(25.16, 39.91)	(25.08, 39.80)	

Табл. 5. Среднее значение и стандартное отклонение метрики TSA для модели Phi-2 при разных параметрах вычисления TSA.
Table 5. Mean and standard deviation of TSA for Phi-2 under different experimental conditions.

Параметры вычисления TSA		Число групп тестов		
Остановка при первой ошибке	Веса групп тестов	Все	3 наибольших	1
Да	Равные	(13.25, 25.49)	(9.69, 24.18)	(4.80, 21.28)
	Пропорциональные	(9.37, 26.07)	(8.91, 25.77)	
Нет	Равные	(14.36, 26.96)	(12.23, 26.57)	(10.90, 27.85)
	Пропорциональные	(10.93, 27.88)	(10.93, 27.88)	

Табл.6. Среднее значение и стандартное отклонение метрики TSA для модели Phi-3 при разных параметрах вычисления TSA.
Table 6. Mean and standard deviation of TSA for Phi-3 under different experimental conditions.

Параметры вычисления TSA		Число групп тестов		
Остановка при первой ошибке	Веса групп тестов	Все	3 наибольших	1
Да	Равные	(12.94, 25.72)	(9.48, 24.40)	(5.13, 21.89)
	Пропорциональные	(9.48, 26.66)	(8.97, 26.09)	
Нет	Равные	(13.87, 26.93)	(11.76, 26.94)	(10.67, 28.17)
	Пропорциональные	(10.77, 28.39)	(10.75, 28.33)	

Табл. 7. Среднее значение и стандартное отклонение метрики TSA для модели Phi-4 при разных параметрах вычисления TSA.
Table 7. Mean and standard deviation of TSA for Phi-4 under different experimental conditions.

Параметры вычисления TSA		Число групп тестов		
Остановка при первой ошибке	Веса групп тестов	Все	3 наибольших	1
Да	Равные	(49.36, 43.79)	(44.29, 45.91)	(37.69, 48.10)
	Пропорциональные	(43.57, 47.30)	(42.73, 47.42)	
Нет	Равные	(52.04, 43.70)	(49.46, 45.35)	(47.03, 47.23)
	Пропорциональные	(47.48, 47.18)	(47.48, 47.18)	

Табл. 8. Среднее значение и стандартное отклонение метрики TSA для модели Qwen2.5-Coder при разных параметрах вычисления TSA.
Table 8. Mean and standard deviation of TSA for Qwen2.5-Coder under different experimental conditions.

Параметры вычисления TSA		Число групп тестов		
Остановка при первой ошибке	Веса групп тестов	Все	3 наибольших	1
Да	Равные	(85.92, 28.14)	(82.13, 33.79)	(75.31, 42.60)
	Пропорциональные	(82.57, 34.94)	(81.40, 36.20)	
Нет	Равные	(88.53, 25.52)	(87.56, 27.68)	(86.13, 31.36)
	Пропорциональные	(86.57, 30.75)	(86.48, 30.79)	

Значения метрики TSA, полученные в рамках дополнительных экспериментов, позволяют провести следующие наблюдения:

- Разброс стандартного отклонения TSA при изменении конфигурации эксперимента возрастает с повышением качества работы модели. При этом наибольшая дисперсия метрики наблюдается при использовании единственной группы тестов.
- Замена равных весов групп тестов на пропорциональные приводит к повышению дисперсии и понижению среднего значения TSA. Это свидетельствует о возможном завышении оценки корректности кода при равных весах для групп разного размера.
- Остановка тестирования программы при получении первой ошибки в группе тестов может занижать оценку качества на несколько пунктов. Это может означать, что для большинства задач тесты в группах имеют примерно одинаковую сложность.

4. Заключение

В настоящей работе предложена метрика TSA для оценки корректности программ на основе вероятностного определения корректности кода, а также проведён сравнительный анализ свойств этой метрики с метрикой Pass@1. Описанные теоретические свойства метрик подтверждаются экспериментами, в рамках которых с помощью 5 языковых моделей Phi-1, Phi-2, Phi-3, Phi-4 и Qwen2.5-Coder сгенерированы программы для решения задач из набора HumanEval. Для экспериментов использовалась новая версия этого набора HumanEval++, построенная в рамках исследования. По результатам проведённых экспериментов метрика TSA демонстрирует меньшую дисперсию по сравнению с Pass@1 при оценке качественных моделей и не требует тонкой настройки условий тестирования для точной оценки моделей с низким качеством работы.

Список литературы / References

[1]. IEEE Computer Society. IEEE Std 610.12-1990. IEEE standard glossary of software engineering terminology, 1991, pp. 1-84.

[2]. Hou X., Zhao Y., Liu Y., Yang Z., Wang K., Li Li, Luo X., Lo D., Grundy J., Wang H. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv preprint arXiv:2308.10620, 2023.

[3]. Hey T., Keim J., Koziolek A., Tichy W. F. NoRBERT. Transfer Learning for Requirements Classification. In 2020 IEEE 28th International Requirements Engineering Conference (RE), IEEE 2020, pp. 169-179.

[4]. Ma L., Liu S., Li Y., Xie X., Bu L. SpecGen: Automated Generation of Formal Program Specifications via Large Language Models. arXiv preprint arXiv:2401.08807, 2024.

[5]. Mandal S., Chethan A., Janfaza V., Mahmud S. M., Anderson T. A., Turek J., Tithi J. J., Muzahid A. Large Language Models Based Automatic Synthesis of Software Specifications. arXiv preprint arXiv:2304.09181, 2023.

[6]. Cassano F., Gouwar J., Nguyen D., Nguyen S., Phipps-Costin L., Pinckney D., Yee M. H., Anderson C. J., Feldman M. Q. et al. MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation. In 2023 IEEE Transactions on Software Engineering, **49**(7), IEEE 2023, pp. 3675-3691.

[7]. Chen F., Fard F. H., Lo D., Bryksin T. On the Transferability of Pre-trained Language Models for Low-resource Programming Languages. In 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), IEEE 2022, pp. 401-412.

[8]. Fan G., Chen S., Gao C., Xiao J., Zhang T., Feng Z. Rapid: Zero-shot Domain Adaptation for Code Search with Pre-trained Models. In 2024 ACM Transactions on Software Engineering and Methodology, ACM 2024, **33**(5), pp. 1-35.

[9]. Schäfer M., Nadi S., Eghbali A., Tip F. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. In 2023 IEEE Transactions on Software Engineering, IEEE 2023, **50**(1), pp. 85-105.

[10]. Steenhoek B., Gao H., Le W. Dataflow Analysis-Inspired Deep Learning for Efficient Vulnerability Detection. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, IEEE 2024, pp. 1-13.

[11]. Li Y., Ren Z., Wang Z., Yang L., Dong L., Zhong C., Zhang H. Fine-SE: Integrating Semantic Features and Expert Features for Software Effort Estimation. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, IEEE 2024, pp. 1-12.

[12]. Nappa A., Johnson R., Bilge L. et al. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In 2015 IEEE symposium on security and privacy, IEEE 2015, pp. 692-708.

[13]. Ardito L., Coppola R., Barbato L., Verga D. A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review. Scientific Programming 2020, pp. 1-26.

[14]. Buse R. P. L., Weimer W. R. Learning a Metric for Code Reliability. IEEE Transactions on Software Engineering, IEEE 2009, **36**(4), pp. 546-558.

[15]. Peitek N., Apel S., Parmin C. et al. Program Comprehension and Code Complexity Metrics: An fMRI Study. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE 2021, pp. 524-536.

[16]. Markovtsev V., Long W., Mougard H. et al. STYLE-ANALYZER: fixing code style inconsistencies with interpretable unsupervised algorithms. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE 2019, pp. 468-478.

[17]. Raemaekers S., Van Deursen A., Visser J. Measuring software library stability through historical version analysis. In 2012 28th IEEE International Conference on Software Maintenance (ICSM), IEEE 2012, pp. 378-387.
[18]. Zheng J., Cao B., Ma Z., Pan R., Lin H., Lu Y., Han X., Sun L. Beyond Correctness: Benchmarking Multi-dimensional Code Generation for Large Language Models. arXiv preprint arXiv:2407.11470, 2024.
[19]. Chen L., Guo Q., Jia H., Zeng Z., Wang X., Xu Y. et al. A Survey on Evaluating Large Language Models in Code Generation Tasks. arXiv preprint arXiv:2408.16498, 2024.
[20]. Elnashar A., Moundas M., Schmidt D. C., Spencer-Smith J., White J. Evaluating the Performance of LLM-Generated Code for ChatGPT4 and AutoGen Along with Top-Rated Human Solutions. In 2024 19th International Conference on Software Technologies (ICSOFT), 2024, pp. 258-270.
[21]. Jiang J., Wang F., Shen J., Kim S., Kim S. A Survey on Large Language Models for Code Generation. arXiv preprint arXiv:2406.00515, 2024.
[22]. Yetiştiren B., Özsoy I., Ayerdem M., Tüzün E. Evaluating the Code Quality of AI-assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. arXiv preprint arXiv:2304.10778, 2023.
[23]. Huang D., Qing Y., Shang W., Cui H., Zhang J. EffiBench: Benchmarking the Efficiency of Automatically Generated Code. In: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C. (eds.) Advances in Neural Information Processing Systems, Curran Associates, Inc., 2024, **37**, pp. 11506-11544.
[24]. Peng Y., Wan J., Li Y., Ren X. COFFE: A Code Efficiency Benchmark for Code Generation. In ACM International Conference on the Foundations of Software Engineering (FSE), ACM 2025.
[25]. Niu C., Zhang T., Li C., Luo B., Ng V. On Evaluating the Efficiency of Source Code Generated by LLMs. In Proceedings of the 2024 IEEE/ACM 1st International Conference on AI Foundation Models and Software Engineering (FORGE), ACM 2024, pp. 103-107.
[26]. Coignion T., Quinton C., Rouvoy R. A Performance Study of LLM-Generated Code on Leetcode. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE), ACM 2024, pp. 79-89.
[27]. Qiu R., Zeng W. W., Ezick J., Lott C., Tong H. How Efficient is LLM-Generated Code? A Rigorous & High-Standard Benchmark. 13th International Conference on Learning Representations (ICLR). (2025)
[28]. Liu J., Xie S., Wang J., Wei Y., Ding Y., Zhang L. Evaluating Language Models for Efficient Code Generation. First Conference on Language Modeling, 2024.
[29]. Du M., Luu A.T., Ji B., Liu Q., Ng S.K. Mercury: A Code Efficiency Benchmark for Code Large Language Models. In Globerson A., Mackey L., Belgrave D., Fan A., Paquet U., Tomczak J., Zhang C. (eds.) Advances in Neural Information Processing Systems, Curran Associates, Inc. 2024, vol. **37**, pp. 16601-16622.
[30]. Chen M., Tworek J., Jun H. et al. Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374, 2021.
[31]. Li Y., Choi D., Chung J. et al. Competition-Level Code Generation with AlphaCode. Science, 2022, vol. **378**(6624), pp. 1092-1097.
[32]. Hendrycks D, Burns C., Kadavath S. et al. Measuring Mathematical Problem Solving with the MATH Dataset. arXiv preprint arXiv:2103.03874, 2021.
[33]. Rajkumar N., Li R., Bahdanau D. Evaluating the Text-to-SQL Capabilities of Large Language Models. arXiv preprint arXiv:2204.00498, 2022.
[34]. Olausson T.X., Inala J.P., Wang C. et al. Is Self-Repair a Silver Bullet for Code Generation? arXiv preprint arXiv:2306.09896, 2023.
[35]. Zan D., Chen B., Zhang F. et al. Large Language Models Meet NL2Code: A Survey. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 2023, **1**, pp. 7443-7464.
[36]. Ouyang L., Wu J., Liang X. et al. Tranining language models to follow instructions with human feedback. Advances in Neural Information Processing Systems 35, 2022, pp. 27730-27744.
[37]. Le H., Wang Y., Gotmare A.D. et al. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. Advances in Neural Information Processing Systems 35, 2022, pp. 21314-21328.
[38]. Shojaee P., Jain A., Tipirneni S., Reddy C.K. Execution-based Code Generation using Deep Reinforcement Learning. arXiv preprint arXiv:2301.13816, 2023.
[39]. Liu J., Zhu Y., Xiao K. et al. RLTF: Reinforcement Learning from Unit Test Feedback. arXiv preprint arXiv:2307.04349, 2023.

[40]. Shen Bo, Zhang J., Chen T. et al. PanGu-Coder2: Boosting Large Language Models for Code with Ranking Feedback. arXiv preprint arXiv:2307.14936, 2023.
[41]. Papineni K., Roukos S., Ward T., Zhu W.-J. BLEU: A Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics, 2002, pp. 311-318.
[42]. Lin C. Y. ROUGE: A Package for Automatic Evaluation of Summaries. In Text Summarization branches out, 2004, pp. 74-81.
[43]. Banerjee S., Lavie A. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgements. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, 2005, pp. 65-72.
[44]. Ren S., Guo D., Li S. et al. CodeBLEU: A Method for Automatic Evaluation of Code Synthesis. arXiv preprint arXiv:2009.10297, 2020.
[45]. Spiess C., Gros D., Pai K. S. et al. Calibration and Correctness of Language Models for Code. arXiv preprint arXiv:2402.02047, 2024.
[46]. Valentin T., Madadi A., Sapia G., Böhme M. Incoherence as Oracle-less Measure of Error in LLM-Based Code Generation. arXiv preprint arXiv:2507.00057, 2025.
[47]. Rice H.G. Classes of Recursively Enumerable Sets and Their Decision Problems. Transactions of the American Mathematical Society, 1953, **74**(2), pp. 358-366.
[48]. Athiwaratkun B., Gouda S. K., Wang Z. Multi-lingual Evaluation of Code Generation Models. arXiv preprint arXiv:2210.14868, 2022.
[49]. Liu J., Xia C. S., Wang Y., Zhang L. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In: Oh A., Naumann T., Globerson A., Saenko K., Hardt M., Levine S. (eds.) Advances in Neural Information Processing Systems, Curran Associates, Inc., 2023. **36**, pp. 21558-21572.
[50]. Austin J., Odena A., Nye M., Bosma M., Michalewski H., Dohan D., Jiang E., Cai C., Terry M., Le Q., Sutton C. Program Synthesis with Large Language Models. arXiv preprint arXiv:2108.07732, 2021.
[51]. Javaheripi M., Bubeck S., Abdin M., Aneja J., Bubeck S., Mendes C. C. T. et al. Phi-2: The surprising power of small language models. Microsoft Research Blog, 2023, **1**(3).
[52]. Suriya G. et al. Textbooks are all you need. arXiv preprint arXiv:2306.11644, 2023.
[53]. Abdin M. et al. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. arXiv preprint arXiv:2404.14219, 2024.
[54]. Abdin M. et al. Phi-4 technical report. arXiv preprint arXiv:2412.08905, 2024.
[55]. Abouelenin A. et al. Phi-4-Mini Technical Report: Compact yet Powerful Multimodal Language Models via Mixture-of-LoRAs. arXiv preprint arXiv:2503.01743, 2025.
[56]. Yang An et al. Qwen2 Technical Report. arXiv preprint arXiv:2407.10671, 2024.
[57]. Binyuan H. et al. Qwen2.5-Coder Technical Report. arXiv preprint arXiv:2409.12186, 2024.
[58]. Авагян Д.А. Реализация метрики TSA на базе набора задач HumanEval++. Доступно по ссылке: https://gitlab.com/Mathematician2000/tsa-analysis, дата обращения: 06.03.2026.
[59]. Авагян Д.А. Открытый набор задач HumanEval++ на HuggingFace Hub. Доступно по ссылке: https://huggingface.co/datasets/Mathematician/HumanEval-PlusPlus, дата обращения: 06.03.2026.

Информация об авторах / Information about authors

Давид Арменович АВАГЯН – аспирант кафедры алгоритмических языков факультета вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова. Сфера научных интересов: нейросетевая генерация кода, метрики качества программ.

David Armenovich AVAGIAN – postgraduate student of the department of algorithmic languages at the faculty of computational mathematics and cybernetics of Lomonosov Moscow State University. Research interests: code generation with neural networks, code quality metrics.