

ПОДХОД К ОЦЕНКЕ КАЧЕСТВА ГЕНЕРАЦИИ СЦЕНАРИЕВ ПЕНТЕСТА ПРИ ПОМОЩИ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ

Конев А. А.¹, Паюсова Т. И.²

DOI: 10.21681/2311-3456-2025-6-147-157

Цель исследования: целью исследования является разработка, внедрение и валидация системы количественных метрик для оценки качества процесса саморефлексии больших языковых моделей, автоматизирующих генерацию сценариев тестирования на проникновение, а также изучение влияния механизмов внешней памяти (RAG) на глубину анализа, системность подхода и способность к итеративному улучшению.

Методы исследования: исследование проведено на основе сравнительного анализа 16 сценариев пентеста, выполненных с помощью YandexGPT и GigaChat в конфигурациях с механизмом RAG и без него. Для оценки использовались метрики саморефлексии, включая когерентность дискурса, индекс маркеров, полноту охвата и потенциал эксплуатации. Результаты сравнивались с эталонными прохождениями виртуальных машин (Basic Pentesting: 1, Dev-guru, Symfonos 2, Tomato: 1) для определения степени соответствия отраслевым стандартам.

Результаты исследования: анализ показал, что все модели демонстрируют высокую когерентность дискурса (1,0) и устойчивость к семантическому сдвигу (0,0), что подтверждает их способность к логически согласованному мышлению. Наибольшую активность саморефлексии проявила GigaChat без RAG (индекс маркеров – 0,273), однако наиболее высокие значения полноты охвата (0,2–0,6) зафиксированы у моделей с RAG. Ключевым результатом стало поведение YandexGPT с RAG, которая в сценарии Basic Pentesting: 1 идентифицировала уязвимость в сервисе ProFTPD 1.3.3c и предложила её эксплуатацию через Metasploit и proftpd_133c_backdoor, что свидетельствует о переходе к фазе активной атаки. GigaChat с RAG продемонстрировал наиболее зрелую саморефлексию, достигнув общего показателя улучшения 0,615 при анализе утечки .git. Полученные данные подтверждают, что RAG значительно расширяет область охвата и качество анализа, но для достижения автономного пентеста требуется интеграция с системами, способными выполнять и оценивать эксплуатацию уязвимостей.

Научная новизна заключается в разработке и применении комплексной системы метрик саморефлексии для оценки качества когнитивного процесса LLM в задачах пентеста. Впервые показано, что RAG не только повышает полноту охвата, но и может инициировать переход к фазе эксплуатации. Предложенный подход позволяет оценивать не только результат, но и качество внутреннего анализа, тем самым, открывая путь к созданию управляемых и прозрачных LLM-агентов для кибербезопасности.

Ключевые слова: тестирование на проникновение, БДУ ФСТЭК, обработка естественного языка, компьютерная лингвистика, Retrieval-Augmented Generation, RAG, саморефлексия, метрики качества.

Введение и постановка проблемы

Современные большие языковые модели (Large Language Model, LLM) всё чаще используются в качестве автономных агентов для решения сложных, многоэтапных задач, включая тестирование на проникновение (пентест). Однако эффективность LLM ограничена склонностью к ошибкам в рассуждениях, галлюцинациям и заикливанию на неудачных стратегиях [1–3]. Для повышения устойчивости и автономности таких агентов ключевое значение приобретает саморефлексия – метакогнитивная способность анализировать собственные действия и учиться на ошибках. Исследования показывают, что LLM, способные к саморефлексии, значительно улучшают производительность в задачах на рассуждение, исправляя логические и арифметические ошибки [4]. Архитектуры

подобные Reflexion демонстрируют, как агенты, получая обратную связь, генерируют текстовые объяснения своих ошибок и используют их для коррекции будущих действий [5]. Описанные механизмы особенно актуальны в тестировании на проникновение (пентесте), где каждая неудачная попытка эксплуатации уязвимости должна приводить к переоценке тактики, а не к повторению ошибок. Например, методы Self-RAG и ReAct расширяют этот подход, интегрируя рефлексию с использованием внешних инструментов и памяти – критически важных компонентов для киберопераций [6, 7].

Существующие подходы к оценке саморефлексии часто ограничиваются конечной точностью или количеством попыток, что не отражает глубину

1 Конев Антон Александрович, кандидат технических наук, доцент, заместитель директора Института системной интеграции и безопасности, доцент кафедры КИБЭВС, Томский государственный университет систем управления и радиоэлектроники, г. Томск, Россия. ORCID: <https://orcid.org/0000-0002-3222-9956>. Scopus Author ID: 23035057200. E-mail: kaa@fb.tusur.ru

2 Паюсова Татьяна Игоревна, доцент, Тюменский государственный университет, г. Тюмень, Россия. ORCID: <https://orcid.org/0000-0003-4923-1689>. Scopus Author ID: 57188574761. E-mail: t.i.payusova@utmn.ru

когнитивного процесса. В данной работе предлагается система количественных метрик, направленных на оценку качества саморефлексии: когерентность дискурса, индекс саморефлексивных маркеров, семантический сдвиг, полнота охвата, потенциал эксплуатации, специфичность и общий показатель улучшения. Метрики основаны на методах компьютерной лингвистики и семантического анализа. Эксперименты с моделями YandexGPT и GigaChat показывают, что предложенная система позволяет объективно сравнивать модели по глубине анализа, способности к итеративному улучшению и системности подхода. Результаты демонстрируют, что интеграция внешних знаний в формате генерации, дополненной поиском (Retrieval-Augmented Generation, RAG), усиливает влияние саморефлексии, особенно на метрики полноты и потенциала. Работа представляет собой шаг к прозрачной, управляемой и поддающейся количественной оценке саморефлексии LLM.

Обзор литературы по теме исследования

Современные исследования в области искусственного интеллекта демонстрируют растущий интерес к механизмам саморефлексии больших языковых моделей (LLM). Эти механизмы рассматриваются как ключевой фактор повышения точности, надёжности и способности к итеративному обучению. Тем не менее, большинство существующих подходов сосредоточено на внешних метриках эффективности, таких как Success Rate [7], Step Efficiency [7] или F1-score [8], что не позволяет оценить качество внутреннего когнитивного процесса. Это ограничение особенно критично в задачах, требующих глубокого анализа и стратегического мышления, например, в автоматизации пентеста.

Одним из первых подходов к формализации саморефлексии стала концепция вербального подкрепления, предложенная в работе [5] (Shinn et al., Reflexion), где агент текстом объясняет свои ошибки. Хотя этот метод показал рост эффективности в задачах на рассуждение, его основная метрика (успех в выполнении задачи) не оценивает качество самого объяснения. Аналогичный недостаток присущ и работе [8] (Self-Refine), где для оценки улучшений используются метрики BLEU, ROUGE и F1: они фокусируются на текстовом совпадении, а не на логической согласованности или исправлении фактических ошибок.

Ещё более узким является подход [6] (Self-RAG), где критика влияет на генерацию, но её влияние на последующие действия не анализируется. Это создаёт риск «имитации рефлексии», когда модель формально генерирует анализ, но не меняет стратегию. Подобную проблему отмечают авторы обзора [9], выделяя три стратегии улучшения вывода: итеративное уточнение, многоагентные дебаты и саморефлексию.

При этом они подчёркивают, что успех не гарантирует качество самоисправления: модель может повторять одну и ту же ошибку, оставаясь успешной в формальном смысле.

Критически важным аспектом является достоверность рассуждений. Работа [10] (Faithful Chain-of-Thought) вводит метрику Faithfulness, показывая, что модели часто генерируют цепочки рассуждений после получения ответа, то есть имитируют рефлексию. Это особенно опасно в контексте пентеста, где анализ должен быть всесторонним и глубоким, а не формальным. Работа [11] (CRITIC) предлагает способ проверки достоверности через внешние инструменты, например, симуляцию запуска nmap или sqlmap, позволяя оценить «прагматическую достоверность» плана.

Ещё одним ключевым фактором является калибровка уверенности. Исследования [12, 13] показывают, что LLM страдают от избыточной уверенности и часто не признают очевидных ошибок. Тем самым, обосновывается введение метрики Confidence-Calibration Gap (CCG), которая связана с «метакогнитивной зрелостью» [13]. Работа [14] подтверждает: калиброванные модели чаще признают ошибки, что делает рефлексию более продуктивной.

Стабильность и логическая целостность также играют важную роль. Метрика Consistency [15] оценивает устойчивость ответов при перефразировке, а работы [16, 17] исследуют фактическую непротиворечивость: анализ ошибки не должен противоречить самому плану (например, одновременно отвергать и использовать nmap). Эти исследования подчёркивают необходимость метрик, оценивающих внутреннюю согласованность, а не только внешний результат.

Перспективным направлением является использование LLM в качестве судей [18, 19]. Подход LLM-as-a-judge [18] позволяет оценивать глубину и полезность анализа, что может стать альтернативой поверхностным метрикам. Использование семантических эмбедингов, таких как Sentence-BERT [19], обеспечивает объективное измерение сходства планов, превосходящее простое текстовое совпадение.

Обзор литературы показывает, что, несмотря на значительный прогресс, отсутствует универсальная система количественной оценки саморефлексии, ориентированная на качество процесса, а не только на результат. Существующие метрики либо игнорируют внутреннюю логику, либо не учитывают прагматическую достоверность и логическую целостность. Данная работа призвана заполнить данный пробел, предлагая систему метрик, основанную на методах компьютерной лингвистики, для объективной и управляемой оценки качества саморефлексии в LLM-агентах для пентеста.

```
def update_active_context(self, target_context: str):
    context_parts = [target_context]
    context_parts.append(self._summarize_failed_attempts())
    recent_steps = self.full_history[-2:] if len(self.full_history) >= 2 else self.full_history[-1:]
    for entry in recent_steps:
        context_parts.append(f"\n--- ШАГ {entry['step_number']} ---")
        context_parts.append(f"Команда: {entry['final_step']}")
        context_parts.append(f"Результат: {entry['result']}")
    phase_info = f"\n--- ТЕКУЩАЯ ФАЗА ---\n{self.current_phase.upper()}\n"
    phase_info += f"Рекомендация: применять методы, релевантные фазе '{self.current_phase}'."
    context_parts.append(phase_info)
    self.active_context = "\n".join(context_parts)
```

Рис. 1. Реализация метода update_active_context

Архитектура модуля саморефлексии

Архитектура модуля построена на унифицированном интерфейсе для работы с различными LLM. Для YandexGPT используется класс YandexGPTClient, а для GigaChat – GigaChatClient. Оба класса инкапсулируют специфичные параметры API (ключ, URI модели, заголовки) и предоставляют единый метод call_model для отправки запросов и обработки ответов.

Центральную логику выполняет класс UniversalSelfReflectionAgent, который использует один из этих клиентов. Его центральным элементом является механизм итеративной саморефлексии. На каждом шаге пентеста LLM сначала генерирует исходный план с помощью generate_step. Затем запускается цикл анализа текущего плана с использованием метода propose_self_reflection, который генерирует текст, выявляющий ошибки, упущения и альтернативы. На основе анализа метод improve_step создает улучшенную версию плана. Цикл может повторяться многократно, что позволяет постепенно повышать качество стратегии.

Ключевым инновационным компонентом является механизм динамического управления контекстом. Метод update_active_context (рис. 1) формирует активный контекст, включающий: (1) постоянные факты о цели, (2) автоматически сгенерированные резюме неудач и (3) результаты последних 1-2 шагов. Это позволяет сохранять ключевые выводы, игнори-

руя шумные или устаревшие данные, и поддерживать долгосрочную согласованность стратегии. Вся полная история сохраняется в full_history и может быть экспортирована в JSON-файл для последующего анализа.

Также можно отметить реализацию механизма автоматической смены фаз пентеста, который учитывает этапы «разведка», «эксплуатация», «повышение прав», «постэксплуатация» и «закрепление». Для этого в код внедрена логика анализа как содержимого текущего шага, так и результата его выполнения с поиском ключевых терминов, относящихся к фазам, а также признаков неудач. При накоплении ошибок или обнаружении терминов, указывающих на возможности эксплуатации, LLM динамически переключается на фазу «эксплуатация» и адаптирует предложения дальнейших шагов.

Технически были расширены словари синонимов фаз (рис.2), реализована функция анализа шага и результата на содержание терминов из списка ключевых слов эксплуатации, а также внедрен счетчик неудачных попыток, который влияет на переключение. Это позволяет системе избегать застойных состояний и более гибко реагировать на изменения сценария.

В работе с API YandexGPT и GigaChat добавлен механизм обновления access token при получении кода авторизации 401, а также реализованы повторные

```
class UniversalSelfReflectionAgent:
    SELF_REFLECTION_MARKERS = [
        "добавить", "улучшить", "следует", "следует пересмотреть", "анализ показывает",
        "обнаружено", "ошибка", "упущенная возможность", "рекомендуется", "альтернативный",
        "коррекция", "недостаток", "правильно", "неправильно", "потенциал", "риск",
        "улучшение", "добавление", "рассмотреть", "проверить", "проанализировать", "изменить"
    ]
    EXPLOITATION_TERMS = [
        "эксплуатация", "exploit", "shell", "meterpreter", "privilege escalation",
        "reverse shell", "payload", "атаку"
    ]
    PHASES_MAP = {
        "разведка": {"разведка", "reconnaissance", "recon", "сканирование", "сбор информации", "поиск"},
        "эксплуатация": {"эксплуатация", "exploitation", "эксплойт", "атака", "эксплуатировать"},
        "повышение привилегий": {"повышение привилегий", "privilege escalation", "priv esc", "эскалация"},
        "постэксплуатация": {"постэксплуатация", "post-exploitation", "post-exploit", "пост этап"},
        "персистентность": {"персистентность", "persistence", "сохранение доступа"}
    }
```

Рис. 2. Пример словарей приложения

попытки запросов с экспоненциальной задержкой при сетевых ошибках. Для каждого запроса реализованы циклы обработки ошибок, логирование попыток и обоснованные таймауты. Описанный подход позволяет выдерживать высокую нагрузку и минимизировать влияние сбоев сети или ограничений платформы.

В рамках модуля фиксируется история сеансов, включая команды, результаты, сгенерированные шаги и метрики. Пользователь взаимодействует с системой через последовательные интерактивные диалоги и может итеративно улучшать шаги на основе саморефлексии, а также самостоятельно вводить результаты выполнения.

Генерация, дополненная поиском, на основе БДУ ФСТЭК

Для повышения релевантности, точности и нормативного соответствия генерируемых стратегий пентеста в архитектуру программного комплекса интегрирован механизм дополненной поисковой генерации (Retrieval-Augmented Generation, RAG), использующий в качестве внешнего источника знаний Банк данных угроз Федеральной службы по техническому и экспортному контролю (БДУ ФСТЭК России). Механизм RAG позволяет языковой модели динамически получать доступ к актуальной, структурированной информации о типовых уязвимостях, методах эксплуатации и последствиях, избегая «галлюцинаций» и обеспечивая соответствие национальным стандартам информационной безопасности.

Реализация RAG включает три ключевых этапа: подготовку источника знаний, векторизацию и семантический поиск. На первом этапе осуществляется парсинг официального документа БДУ в формате .docx с помощью библиотеки python-docx. Каждая запись об угрозе, содержащая идентификатор (например, УБИ.006), наименование, описание механизма реализации и рекомендации по защите, преобразуется в единый текстовый блок. Для обеспечения отказоустойчивости при недоступности файла используется заранее подготовленный набор эталонных угроз.

На втором этапе все текстовые описания угроз векторизуются с помощью предобученной мультимодальной модели paraphrase-multilingual-MiniLM-L12-v2 из библиотеки Sentence Transformers. Эта модель генерирует 384-мерные эмбединги, которые кодируют семантическое содержание текста, позволяя выявлять схожесть по смыслу даже при различии формулировок. Полученные векторы сохраняются в локальной векторной базе данных ChromaDB, оптимальной для прототипирования и интеграции с LLM. При каждой инициализации система принудительно пересоздаёт коллекцию, что исключает ошибки несоответствия размерности эмбедингов.

Третий этап – семантический поиск и интеграция контекста (рис. 3). При генерации шага пентеста или на этапе саморефлексии формируется поисковый запрос на основе текущего контекста (например,

```
class BDUFSTECRAG:
    """Механизм RAG для поиска релевантных угроз из БДУ."""

    def __init__(self, threats, model_name='paraphrase-multilingual-MiniLM-L12-v2'):
        self.threats = threats
        self.model = SentenceTransformer(model_name)
        self.client = chromadb.Client(Settings(anonymized_telemetry=False))
        self.collection_name = "bdufstec_threats"

        if self.collection_name in [col.name for col in self.client.list_collections()]:
            self.client.delete_collection(self.collection_name)
        self.collection = self.client.create_collection(self.collection_name)
        self._embed_and_store_threats()

    def _embed_and_store_threats(self):
        embeddings = self.model.encode(self.threats)
        self.collection.add(
            embeddings=embeddings.tolist(),
            documents=self.threats,
            ids=[f"threat_{i}" for i in range(len(self.threats))]
        )
        print(f"✅ {len(self.threats)} угроз добавлены в векторную базу.")

    def retrieve_relevant_threats(self, query, k=3):
        query_embedding = self.model.encode([query])
        results = self.collection.query(
            query_embeddings=query_embedding.tolist(),
            n_results=k
        )
        return results['documents'][0] if results['documents'] else []
```

Рис. 3. Реализация механизма RAG на базе БДУ ФСТЭК

«обход WAF», «эксплуатация LFI»). Запрос векторизуется той же моделью и передаётся в ChromaDB для поиска K ближайших соседей по косинусному расстоянию.

По умолчанию извлекаются три наиболее релевантные угрозы, которые встраиваются в системный промпт LLM. Например, при анализе неудачной атаки через WAF, RAG может вернуть угрозу, связанную с фильтрацией входных данных, что позволяет модели сформулировать гипотезу: «Возможно, срабатывает правило 942100 (SQLi)», и скорректировать стратегию: «Попробую обфускацию через комментарии и кодирование». Таким образом, модель генерирует ответ, обогащённый нормативными знаниями, а не только своими внутренними данными.

Описание метрик саморефлексии

На основе обзора литературы установлено, что существующие подходы к оценке эффективности агентов часто ограничиваются внешними метриками, такими как успешность (success rate) или количество попыток. Это не позволяет оценить качество внутреннего процесса принятия решений. Современные исследования демонстрируют важность саморефлексии, но не предлагают универсальной системы её количественной оценки. В связи с этим, для объективного анализа итеративного механизма саморефлексии была сформирована система метрик, основанная на методах компьютерной лингвистики. Метрики направлены на измерение ключевых аспектов когнитивного процесса: от логической согласованности и факта анализа до системности подхода и практической направленности, что позволяет получить прозрачную и управляемую оценку качества саморефлексии.

Для количественной оценки эффективности механизма итеративной саморефлексии LLM предложена система метрик, основанная на методах компьютерной лингвистики. Пусть $P^{(i)}$ – план пентеста на итерации i , $R^{(i)}$ – текст саморефлексии (анализа) на итерации i , а T – множество всех итераций.

1. Когерентность дискурса (Discourse Coherence)

Мера отображает логическую связанность и семантическую согласованность плана между итерациями.

$$discourse_coherence^{(i)} = \cos(V_{tf-idf}(P^{(i)}), V_{tf-idf}(P^{(i-1)})), \quad (1)$$

где $V_{tf-idf}(P^{(i)})$ – векторное представление плана $P^{(i)}$ в пространстве $TF-IDF$; $\cos(\cdot, \cdot)$ – косинусное сходство между векторами. Значение близкое к 1 указывает на высокую стабильность плана.

2. Индекс саморефлексивных маркеров (Self-Reflection Markers Index)

Мера отображает наличие в тексте саморефлексии $R^{(i)}$ лингвистических признаков анализа.

$$marker_index^{(i)} = \frac{|\{m \in M | m \text{ входит в } R^{(i)}\}|}{|M|}, \quad (2)$$

где $M = \{\text{«следует», «анализ показывает», «обнаружено», «улучшить», «рекомендуется», ...}\}$ – заранее определённый набор маркеров. $|M|$ – общее количество маркеров. Значение 1.0 означает, что все маркеры присутствуют.

3. Семантический сдвиг (Semantic Shift)

Мера отображает степень изменения смысла и структуры плана между итерациями.

$$semantic_shift^{(i)} = 1 - discourse_coherence^{(i)}, \quad (3)$$

где высокое значение указывает на трансформационный характер улучшения. Низкое значение указывает на консервативное уточнение.

4. Полнота охвата (Coverage Completeness)

Мера того, насколько план охватывает стандартные фазы пентеста.

$$coverage_completeness^{(i)} = |F| \{f \in F | f \text{ упоминается в } P^{(i)}\}, \quad (4)$$

где $F = \{\text{разведка, эксплуатация, повышение прав, постэксплуатация, закрепление}\}$ – набор фаз пентеста, $|F| = 5$.

5. Потенциал эксплуатации (Exploitation Potential)

Мера отражает ориентацию плана на практическую эксплуатацию уязвимостей.

$$exploitation_potential^{(i)} = \min(1.0, \frac{count(E, P^{(i)})}{10}), \quad (5)$$

где $E = \{\text{«эксплуатация», «exploit», «shell», «privilege escalation», «reverse shell», ...}\}$ – набор терминов эксплуатации. $count(E, P^{(i)})$ – количество упоминаний терминов из E в плане $P^{(i)}$, нормализация проводится по 10-ти упоминаниям.

6. Специфичность (Specificity)

Мера отражает детализацию плана.

$$specificity^{(i)} = \frac{count(S, P^{(i)})}{10}, \quad (6)$$

где $S = \{\text{«nmap», «metasploit», «msfvenom», «CVE», «LFI», ...}\}$ – набор специфичных терминов, нормализация проводится по 10 упоминаниям.

7. Общий показатель улучшения (Overall Improvement Index)

Сводная метрика, обобщающая качество саморефлексии.

$$overall_improvement^{(i)} = \sum_{k \in K} w_k \times m_k^{(i)}, \quad (7)$$

где K – набор метрик (например, когерентность, индекс маркеров, полнота и т.д.), $m_k^{(i)}$ – значение метрики k на итерации i , w_k – вес метрики (например, $w_k = 0,2$ для всех, если $|K| = 5$).

Предложенная система метрик позволяет комплексно оценивать качество процесса саморефлексии LLM для пентеста, переходя от субъективной оценки к объективным, количественным показателям. Ключевой метрикой является когерентность

дискурса, которая с высокой точностью отражает внутреннюю логическую согласованность плана: высокие значения указывают на стабильную и осознанную работу модели, улучшающую план, не внося противоречий. Эта метрика является фундаментом для оценки надежности. Индекс саморефлексивных маркеров служит важным дополнением, подтверждая сам факт проведения анализа, что критически важно для валидации работы механизма. Вместе они позволяют отличить истинную саморефлексию от простой регенерации.

Семантический сдвиг предоставляет контекст для интерпретации когерентности, помогая различать консервативный (уточняющий) и трансформационный (перестраивающий) стили улучшения. Полнота охвата оценивает системность подхода, что особенно ценно для пентеста, где необходимо последовательно проходить все фазы. Метрика специфичность является важным индикатором детализации и технической точности плана пентеста. Она измеряет, насколько LLM использует конкретные, предметно-ориентированные термины. Потенциал эксплуатации фокусируется на практической направленности плана, оценивая, насколько стратегия ориентирована на выполнение конкретных действий по получению контроля над системой, а не на теоретическое обсуждение уязвимостей.

Тестирование метрик на примере прохождения уязвимых виртуальных машин

В рамках исследования были выбраны четыре виртуальные машины с платформы VulnHub: Basic Pentesting: 1, Devguru, Symfonos 2 и Tomato: 1. Выбор обусловлен стремлением оценить эффективность механизма саморефлексии LLM при решении задач пентеста с различной архитектурой и уровнем сложности.

Машины Basic Pentesting: 1 (уровень Easy) и Symfonos 2 (уровень Medium) имеют четкие, но

многоэтапные сценарии прохождения, что позволяет оценить системность подхода и способность к итеративному улучшению. Например, на Symfonos 2 основной вектор – эксплуатация уязвимости Samba (CVE-2007-2447), требующая точного следования логике «разведка → анализ уязвимости → эксплуатация». В то же время, машины Devguru (уровень Medium) и Tomato: 1 (уровень Medium-Hard) представляют собой сложные, многоуровневые среды с нелинейными путями к компрометации.

Devguru включает современные технологии, такие как Gitea и OctoberCMS, и имитирует реальную ИТ-инфраструктуру, что требует от LLM навыков работы с системой контроля версий и веб-приложениями. Tomato: 1 характеризуется нестандартными векторами, например, SSH Log Poisoning, и требует нестандартного мышления для прохождения заданий.

Комбинация машин обеспечивает всестороннюю валидацию: от оценки базовой способности к саморефлексии на предсказуемых сценариях до проверки устойчивости и адаптивности в сложных, реалистичных условиях. Таким образом, можно не только определить, достигает ли LLM цели, но и понять, как она к ней приходит, делая процесс саморефлексии прозрачным и поддающимся количественной оценке.

Результаты и обсуждение

Анализ показал, что все модели демонстрируют высокую когерентность дискурса (1.0) и отсутствие семантического сдвига (0.0). Это может свидетельствовать о стабильности логической структуры и способности удерживать фокус на цели. Поскольку каждый шаг выполнялся с ручным подтверждением, экспериментатор выступал в роли «корректора», возвращая модель к цели при отклонении. В будущем для повышения чувствительности можно рассмотреть переход на полностью автономный режим выполнения.

Таблица 1.

Сравнительный анализ результатов YandexGPT и GigaChat (с RAG и без)

Метрика/ Конфигурация	Когерентность дискурса	Индекс саморефлексивных маркеров	Семантический сдвиг	Полнота охвата	Потенциал эксплуатации	Специфичность	Общий показатель улучшения
YandexGPT (без RAG)	1,0	0,182	0,0	0,0	0,0	0,15	0,445
YandexGPT (с RAG)	1,0	0,136	0,0	0,2	0,3	0,25	0,427
GigaChat (без RAG)	1,0	0,273	0,0	0,0	0,1	0,30	0,465
GigaChat (с RAG)	1,0	0,227	0,0	0,2	0,0	0,45	0,504

Метрика индекса саморефлексивных маркеров, отражающая активность саморефлексии, варьировалась от 0,09 до 0,27. Наиболее высокие значения зафиксированы у GigaChat (без RAG), указывающие на его способность к глубокому анализу шагов даже без внешнего контекста. Все полученные средние значения метрик представлены в (табл. 1).

Одним из достижений эксперимента стало поведение YandexGPT с RAG на виртуальной машине Basic Pentesting: 1 (рис. 4). Модель не только корректно выполнила разведку (nmap), но и перешла к фазе эксплуатации. Она идентифицировала уязвимость в сервисе ProFTPD 1.3.3c и предложила использовать модуль exploit/unix/ftp/proftpd_133c_backdoor в Metasploit. Таким образом, показано, что интеграция RAG с БДУ позволяет модели не просто обнаруживать уязвимости, но и подбирать соответствующие векторы атаки, приближая её к роли автономного агента.

Анализ сгенерированных сценариев показал, что GigaChat с RAG демонстрирует наивысшую

специфичность, часто генерируя полные, готовые к выполнению команды (например, `wget --no-check-certificate https://10.0.2.20/.git/objects/pack/pack-*.pack -P /tmp/git_objects/`). Это указывает на то, что интеграция с RAG способствует не только расширению контекста, но и повышению практической ценности вывода модели.

Метрика полнота охвата также показала положительную динамику. GigaChat с RAG достиг наилучшего результата (до 0.2), последовательно пройдя путь от обнаружения .git до попытки извлечения флага из .pack-файлов. RAG расширяет область охвата и помогает моделям воссоздавать сложные сценарии, основанные на утечке исходного кода.

Тем не менее, фаза эксплуатации остаётся основным вызовом. У YandexGPT с RAG она была достигнута в единичном случае. В остальных сценариях, включая GigaChat, модели не демонстрировали устойчивого использования терминов эксплуатации (exploit, shell, RCE). Метрика потенциала эксплуатации оставалась низкой, т.е. переход к атаке пока

```
=====
ШАГ ПЕНТЕСТА 3
=====

[ ] Исходный шаг:
1. Команда: провести тестирование на эксплуатацию уязвимости в FTP-сервере.
2. Цель: использовать обнаруженную уязвимость для получения доступа к системе или для выполнения дополнительных атак.
3. Ожидаемый результат: получить доступ к системе через FTP-сервер или выполнить дополнительные атаки, например, загрузить вредоносный код.
4. Инструмент: использовать инструменты для эксплуатации уязвимостей, такие как Metasploit Framework или Cobalt Strike, для тестирования на эксплуатацию уязвимости proftpd_133c_backdoor.

[ ] Итерация саморефлексии 1

[?] Анализ (саморефлексия):
**Корректность:**
- Шаг выполнен корректно, обнаружена уязвимость proftpd_133c_backdoor в FTP-сервере.

**Улучшения:**
- Можно рассмотреть дополнительные меры для минимизации рисков, например, использование более современных инструментов для пентестинга или расширение плана тестирования для проверки других потенциальных уязвимостей.

**Альтернативы:**
- В качестве альтернативы можно было бы использовать другие инструменты для эксплуатации уязвимостей, например, Nmap для сканирования портов и выявления открытых FTP-серверов, а затем более детальный анализ с помощью Metasploit Framework или Cobalt Strike.

**Ошибки:**
- Ошибка в выполнении шага не обнаружена. Однако стоит убедиться, что все шаги пентестинга документированы и понятны для всех членов команды.
```

Рис. 4. Результат саморефлексии YandexGPT + RAG

Шаг 10 является важным этапом подготовки к последующему анализу содержимого репозитория. После сохранения всех необходимых файлов можно приступить к извлечению и изучению их содержимого, что потенциально приведёт к обнаружению уязвимостей или конфиденциальной информации.

Результат: ожидаются загруженные файлы pack-*.pack в директории /tmp/git_objects/, которые можно распаковать и исследовать далее.

```
[ ] Итерация саморефлексии 1
[?] Отправка запроса к API (попытка 1)...
[✓] Успешный ответ от API

[?] Анализ (саморефлексия):
## КРАТКИЕ ВЫВОДЫ ПО ШАГУ 10

### Корректность шага:
- **Положительный момент:** Команда выполнена успешно, получены метаданные о файлах репозитория через GET-запросы.
- **Недостаток:** Не были скачаны сами файлы *.pack, необходимые для дальнейшего глубокого анализа.

### Улучшения:
- Добавить команду для скачивания файлов *.pack:
  ...
  wget --no-check-certificate https://10.0.2.20/.git/objects/pack/pack-*.pack -P /tmp/git_objects/
  ...
- Проверить наличие дополнительных служебных файлов репозитория, например, *.config, *.description.

### Альтернативы:
- Можно использовать curl вместо wget:
  ...
  curl -k https://10.0.2.20/.git/objects/pack/pack-*.pack > /tmp/git_objects/pack-*.pack
  ...
```

Рис. 5. Пример саморефлексии от GigaChat + RAG

не является системной способностью, а зависит от качества RAG и конкретного сценария.

Обе модели показали высокий потенциал как интеллектуальные помощники. GigaChat, особенно с RAG, демонстрирует глубокую саморефлексию, предлагая альтернативы и анализируя возможные ошибки. Общий показатель улучшения достигал 0.513, что говорит о высоком качестве внутреннего анализа (рис.5).

Анализ логов GigaChat без RAG показал, что модель стабильно выполняет начальные этапы пентеста: `ntmap -p`, `gobuster`, `curl`. Она активно использует саморефлексию, предлагая улучшения вроде «добавить `-sV`», «рассмотреть `wget`», «проверить логи». Однако, как и YandexGPT без RAG, она не переходит к эксплуатации. Это подтверждает, что внешний контекст (RAG) необходим для продвижения в более сложные фазы.

YandexGPT и GigaChat, особенно с RAG, демонстрируют значительный прогресс в автоматизации пентеста. Они способны к последовательному мышлению, саморефлексии и формированию логически связанных сценариев. Для дальнейшего развития необходимо расширять RAG за счёт включения PoC-эксплоитов, базы уязвимостей CVE и/или NVD, внедрять триггеры на эксплуатацию и организовывать обратную связь по результатам выполнения команд.

Предложенные метрики оказались информативными и позволили выявить ключевые паттерны поведения моделей, особенно в условиях отсутствия полного успеха (например, получения флага). Метрики позволили не только оценивать конечный результат, но и анализировать качество когнитивного процесса модели.

В отличие от традиционных метрик (например, *success rate*), метрики позволили:

- Оценить качество процесса, даже если итоговая цель не достигнута.
- Выявить модели с высоким когнитивным потенциалом, которые, несмотря на отсутствие флага (итогового результата), демонстрируют логически обоснованный и структурированный подход.
- Проанализировать влияние RAG на качество саморефлексии (например, рост полноты охвата с 0.0 до 0.2).
- Обнаружить «имитацию рефлексии», когда модель использует маркеры (улучшить, рассмотреть), но не предлагает содержательных улучшений.

Особенно ценной оказалась когерентность дискурса. Она стала надёжным индикатором стабильности LLM. Модели с высокой когерентностью (1.0)

редко «зацикливались» и быстрее предлагали смену стратегии при неудаче.

В то же время, метрика «потенциал эксплуатации» требует дальнейшей доработки. Её низкие значения не всегда отражают реальные возможности модели, так как упоминание термина (например, *payload*) не гарантирует его применение. Тем не менее, она служит важным индикатором направления развития, т.к. рост метрики в будущем будет означать приближение к автономной эксплуатации.

Для дальнейшего развития системы возможна интеграция предложенных метрик с фреймворком MITRE ATT&CK, что позволит не только оценивать качество саморефлексии, но и сопоставлять действия большой языковой модели с реальными тактиками и техниками кибератак. Это обеспечит соответствие отраслевым стандартам и повысит практическую ценность LLM, трансформировав его из инструмента анализа в полноценного партнёра пентестера, способного демонстрировать покрытие тактик, таких как Initial Access, Execution, Persistence и Exfiltration. Такой подход откроет путь к автоматизированной отчётности и оценке полноты пентеста в терминах киберугроз.

Также целесообразно рассмотреть использование современных семантических эмбедингов, таких как Sentence-BERT, BERTScore или OpenAI embeddings, для повышения точности оценки смыслового содержания. Замена традиционного *TF-IDF* на эти технологии позволит более точно выявлять схожесть по смыслу, учитывая контекст слов, синонимию и даже интенцию высказывания. Это особенно важно для метрик когерентности дискурса и семантического сдвига, где требуется понимание не просто лексического совпадения, а логической преемственности мысли.

Кроме того, перспективным направлением является автоматизация промпт-инжиниринга и реализация джейлбрейкинга LLM на основе анализа метрик саморефлексии. Например, если индекс саморефлексивных маркеров оказывается ниже порогового значения, система может автоматически модифицировать промпт, добавляя фразы вроде: «Обязательно используйте фразы «следует пересмотреть» и «анализ показывает» или «Предложите минимум три альтернативных действия». Аналогично, при низком значении потенциала эксплуатации можно активировать триггеры на фазу атаки, подсказывая модели: «Рассмотрите возможность использования Metasploit для эксплуатации найденной уязвимости». Такой замкнутый цикл адаптации превращает метрики из отдельных индикаторов в активные управляющие сигналы, формируя систему самонастраивающегося LLM-агента.

Заключение

Проведённое исследование демонстрирует, что большие языковые модели, в особенности при интеграции с механизмом RAG, способны к высококачественной саморефлексии и последовательному выполнению начальных фаз тестирования на проникновение. Разработанная система метрик позволяет объективно оценивать не только конечный результат, но и глубину когнитивного процесса: логическую согласованность, полноту охвата и потенциал перехода к эксплуатации. Метрики формируют многоуровневую оценочную шкалу, позволяющую сравнивать модели и управлять процессом их совершенствования.

Полученные данные подтверждают, что RAG значительно расширяет контекстуальные возможности моделей, что критически важно для работы

с отраслевыми базами угроз, например, такими как БДУ ФСТЭК. В частности, RAG может инициировать или ускорить переход к фазе активной атаки, это является важным шагом к автономности (предложение от YandexGPT при интеграции с RAG применить эксплойт `proftpd_133c_backdoor`).

Ключевым ограничением остаётся низкая активность в фазе эксплуатации, что указывает на необходимость дальнейшего развития архитектур, способных к автономному исполнению и оценке атакующих действий. Представленный подход открывает путь к созданию прозрачных, управляемых и безопасных LLM-агентов, которые могут выступать надёжными помощниками специалистам по кибербезопасности, повышая эффективность и качество пентеста.

Работа выполнена при финансовой поддержке Министерства науки и высшего образования РФ в рамках базовой части государственного задания ТУСУРа на 2023–2025 гг. (проект № FEWM-2023-0015).

Литература

1. Конев А. А., Паюсова Т. И. Большие языковые модели в информационной безопасности и тестировании на проникновение: систематический обзор возможностей применения // Научно-технический вестник информационных технологий, механики и оптики. 2025. Т. 25. №. 1. С. 42–52. DOI: 10.17586/2226-1494-2025-25-1-42-52.
2. Gholami Y. Large language models (LLMs) for cybersecurity: A systematic review // World Journal of Advanced Engineering Technology and Sciences. 2024. Т. 13. №. 1. С. 057–069. DOI: 10.30574/wjaets.2024.13.1.0395.
3. Happe A., Cito J. Getting pwn'd by ai: Penetration testing with large language models // Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering. 2023. С. 2082–2086. DOI: 10.1145/3611643.3613083.
4. Renze M., Guven E. Self-reflection in large language model agents: Effects on problem-solving performance // 2024 2nd International Conference on Foundation and Large Language Models (FLLM), IEEE. 2024. С. 516–525. DOI: 10.1109/FLLM63129.2024.10852426.
5. Shinn N. et al. Reflexion: Language agents with verbal reinforcement learning // Advances in Neural Information Processing Systems. 2023. Т. 36. С. 8634–8652. DOI: 10.5555/3666122.3666499.
6. Asai A. et al. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection // International Conference on Learning Representations. 2024. DOI: 10.48550/ARXIV.2310.11511.
7. Yao S. et al. REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS // 11th International Conference on Learning Representations, ICLR. 2023. DOI: 10.48550/ARXIV.2210.03629.
8. Madaan A. et al. Self-refine: Iterative refinement with self-feedback // Advances in Neural Information Processing Systems. 2023. Т. 36. С. 46534–46594. DOI: 10.48550/ARXIV.2303.17651.
9. Pan L. et al. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies // Transactions of the Association for Computational Linguistics. 2024. Т. 12. С. 484–506. DOI: 10.1162/tacl_a_00660.
10. Lyu Q. et al. Faithful Chain-of-Thought Reasoning // Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers). 2023. С. 305–329. DOI: 10.18653/v1/2023.ijcnlp-main.20.
11. Gou Z. et al. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing // The Twelfth International Conference on Learning Representations. DOI: 10.48550/ARXIV.2305.11738.
12. Desai S., Durrett G. Calibration of Pre-trained Transformers // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020. С. 295–302. DOI: 10.18653/v1/2020.emnlp-main.21.
13. Jiang Z. et al. How can we know when language models know? On the calibration of language models for question answering // Transactions of the Association for Computational Linguistics. 2021. Т. 9. С. 962–977. DOI: 10.1162/tacl_a_00407.
14. Zhao Z. et al. Calibrate before use: Improving few-shot performance of language models // International conference on machine learning, PMLR. 2021. С. 12697–12706. DOI: 10.48550/ARXIV.2102.09690.
15. Chen Y. et al. Measuring and Improving Chain-of-Thought Reasoning in Vision-Language Models // Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). 2024. С. 192–210. DOI: 10.18653/v1/2024.naacl-long.11.
16. Liu Y. et al. On Improving Summarization Factual Consistency from Natural Language Feedback // Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2023. С. 15144–15161. DOI: 10.18653/v1/2023.acl-long.844.

17. Elazar Y. et al. Measuring and improving consistency in pretrained language models // Transactions of the Association for Computational Linguistics. 2021. T. 9. C. 1012–1031. DOI: 10.1162/tacl_a_00410.
18. Zheng L. et al. Judging llm-as-a-judge with mt-bench and chatbot arena // Advances in neural information processing systems. 2023. T. 36. C. 46595–46623. DOI: 10.48550/ARXIV.2306.05685.
19. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019. C. 3982–3992. DOI: 10.18653/v1/D19-1410.

AN APPROACH TO ASSESSING THE QUALITY OF PENTEST SCENARIO GENERATION USING LARGE LANGUAGE MODELS

Konev A. A.³, Payusova T. I.⁴

Keywords: Penetration Testing, FSTEC BDU, Natural Language Processing, Computational Linguistics, Retrieval-Augmented Generation, RAG, Self-Reflection, Quality Metrics.

The purpose of the research: the aim of the study is to develop, implement and validate a system of quantitative metrics for assessing the quality of the self-reflection process of large language models that automate the generation of penetration testing scenarios, as well as to study the influence of external memory mechanisms (RAG) on the depth of analysis, the systematicity of the approach and the ability to iteratively improve.

Research methods: The study was conducted based on a comparative analysis of 16 pentest scenarios performed using YandexGPT and GigaChat in configurations with and without the RAG mechanism. Self-reflection metrics were used for the assessment, including discourse coherence, marker index, completeness of coverage, and exploitation potential. The results were compared with reference virtual machine passes (Basic Pentesting: 1, Devguru, Symfonos 2, Tomato: 1) to determine the degree of compliance with industry standards.

Research results: The analysis showed that all models develop high discourse coherence (1.0) and resistance to the semantic step (0.0), which confirms their ability for logical coherent thinking. The highest self-reflection activity was implemented by GigaChat without RAG (marker index - 0.273), but the highest values of completeness of coverage (0.2–0.6) were recorded for models with RAG. The key result was the behavior of YandexGPT with RAG, which in the Basic Pentesting: 1 scenario identified a vulnerability in the ProFTPD 1.3.3c service and offered to connect it via Metasploit and proftpd_133c_backdoor, indicating the transition to the active response phase. GigaChat with RAG demonstrated the most mature self-reflection, achieving an overall improvement of 0.615 when analyzing the .git leak. The findings confirm that RAG significantly enhances coverage and quality analysis, but achieving standalone pentesting requires integration with variables capable of performing and assessing vulnerabilities.

Scientific novelty: the scientific novelty lies in the development and application of a comprehensive system of self-reflection metrics to assess the quality of the LLM cognitive process in pentest tasks. It has been shown for the first time that RAG not only increases the completeness of coverage, but can also initiate the transition to the exploitation phase. The proposed approach allows us to assess not only the result, but also the quality of internal analysis, thereby opening the way to creating manageable and transparent LLM agents for cybersecurity.

References

1. Konev, A., & Payusova, T. (2025). Large language models in information security and penetration testing: a systematic review of application possibilities. Scientific and Technical Journal of Information Technologies Mechanics and Optics, 25(1), 42–52. DOI: 10.17586/2226-1494-2025-25-1-42-52.
2. Gholami, N. Y. (2024). Large Language Models (LLMs) for Cybersecurity: A Systematic review. World Journal of Advanced Engineering Technology and Sciences, 13(1), 057–069. DOI: 10.30574/wjaets.2024.13.1.0395.
3. Happe, A., & Cito, J. (2023). Getting pwn'd by AI: penetration testing with large language models. Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. DOI: 10.1145/3611643.3613083.
4. Renze, M., & Guven, E. (2024). Self-Reflection in large Language model Agents: Effects on Problem-Solving Performance. 2nd International Conference on Foundation and Large Language Models (FLLM), 516–525. DOI: 10.1109/FLLM63129.2024.10852426.
5. Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K. & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 36, 8634–8652. DOI: 10.5555/3666122.3666499.
6. Asai, A., Wu, Z., Wang, Y., Sil, A., & Hajishirzi, H. (2023). Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arXiv (Cornell University). DOI: 10.48550/ARXIV.2310.11511.
- 3 Anton A. Konev, Ph.D. of Technical Sciences, Associate Professor, Deputy Director of the Institute of System Integration and Security, Associate Professor of the Department of CIBEVS, Tomsk State University of Control Systems and Radioelectronics, Tomsk, Russia. ORCID: <https://orcid.org/0000-0002-3222-9956>. Scopus Author ID: 23035057200. E-mail: kaa@fb.tusur.ru
- 4 Tatyana I. Payusova, Associate Professor, Tyumen State University, Tyumen, Russia. ORCID: <https://orcid.org/0000-0003-4923-1689>. Scopus Author ID: 57188574761. E-mail: t.i.payusova@utmn.ru

7. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023, January). React: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR). DOI:10.48550/ARXIV.2210.03629.
8. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., ... & Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 46534-46594. DOI: 10.48550/ARXIV.2303.17651.
9. Pan, L., Saxon, M., Xu, W., Nathani, D., Wang, X., & Wang, W. Y. (2024). Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12, 484–506. DOI: 10.1162/tac1_a_00660.
10. Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., ... & Callison-Burch, C. (2023, November). Faithful chain-of-thought reasoning. In The 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AAACL 2023). DOI: 10.18653/v1/2023.ijcnlp-main.20.
11. Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., & Chen, W. (2024). Critic: Large language models can self-correct with tool-interactive critiquing. DOI:10.48550/ARXIV.2305.11738.
12. Desai, S., & Durrett, G. (2020). Calibration of Pre-trained Transformers. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 295–302. DOI:10.18653/v1/2020.emnlp-main.21.
13. Jiang, Z., Araki, J., Ding, H., & Neubig, G. (2021). How can we know when language models know? On the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 9, 962-977. DOI:10.1162/tac1_a_00407.
14. Zhao, Z., Wallace, E., Feng, S., Klein, D., & Singh, S. (2021, July). Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning* (pp. 12697–12706). PMLR. DOI:10.48550/ARXIV.2102.09690.
15. Chen, Y., Sikka, K., Cogswell, M., Ji, H., & Divakaran, A. (2024). Measuring and improving chain-of-thought reasoning in vision-language models. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1. 192-210. DOI: 10.18653/v1/2024.naacl-long.11.
16. Liu, Y., Deb, B., Teruel, M., Halfaker, A., Radev, D., & Awadallah, A. H. (2023). On improving summarization factual consistency from natural language feedback. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. 1. 15144–15161. DOI:10.18653/v1/2023.acl-long.844.
17. Elazar, Y., Kassner, N., Ravfogel, S., Ravichander, A., Hovy, E., Schütze, H., & Goldberg, Y. (2021). Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9, 1012–1031. DOI: 10.1162/tac1_a_00410.
18. Zheng, L., Chiang, W. L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., ... & Stoica, I. (2023). Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36, 46595–46623. DOI: 10.48550/ARXIV.2306.05685.
19. Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3982–3992. DOI: 10.18653/V1/D19-1410.

