УДК 004.65:004.4
http://doi.org/10.18799/29495407/2026/1/118
Шифр специальности ВАК: 2.3.5
Научная статья

Влияние моделей параллелизма и механизма GIL на производительность Python-приложений при взаимодействии с аналитической СУБД

А.Ю. Демин[1,2], Д.И. Дмитрийчук[3], В.А. Зарубин[3]✉

[1] Национальный исследовательский Томский политехнический университет, Российская Федерация, г. Томск
[2] Национальный исследовательский Томский государственный университет, Российская Федерация, г. Томск
[3] Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО», Российская Федерация, г. Санкт-Петербург

✉vladislavzarubin113@gmail.com

Аннотация. В работе исследуется влияние моделей параллелизма и механизма GIL на производительность Python-приложений при взаимодействии с аналитической СУБД. Рассматриваются сценарии, включающие извлечение, обработку и запись данных, формирующие смешанную I/O- и CPU-нагрузку.

Ключевые слова: Python, параллелизм, многопоточность, многопроцессность, GIL, аналитическая СУБД, производительность

Для цитирования: Демин А.Ю., Дмитрийчук Д.И., Зарубин В.А. Влияние моделей параллелизма и механизма GIL на производительность Python-приложений при взаимодействии с аналитической СУБД. *Известия Томского политехнического университета. Промышленная кибернетика*, 2026, Т. 4, № 1, С. 35–41. http://doi.org/10.18799/29495407/2026/1/118

Конфликт интересов: отсутствует.

UDC 004.65:004.4
http://doi.org/10.18799/29495407/2026/1/118
Scientific paper

Impact of parallelism models and the GIL mechanism on the performance of Python applications when interacting with an analytical DBMS

A.Yu. Demin[1,2], D.I. Dmitriychuk[3], V.A. Zarubin[3]✉

[1] National Research Tomsk Polytechnic University, Tomsk, Russian Federation
[2] National Research Tomsk State University, Tomsk, Russian Federation
[3] National Research University ITMO, St Petersburg, Russian Federation

✉vladislavzarubin113@gmail.com

Abstract. This paper examines the impact of parallelism models and the Global Interpreter Lock on the performance of Python applications interacting with an analytical DBMS. The study focuses on typical workloads that include data retrieval, processing, and insertion, representing a combination of I/O-bound and CPU-bound operations.

Keywords: Python, parallelism, multithreading, multiprocessing, GIL, analytical DBMS, performance

For citation: Demin A.Yu., Dmitriychuk D.I., Zarubin V.A. Impact of parallelism models and the GIL mechanism on the performance of python applications when interacting with an analytical DBMS. *Bulletin of the Tomsk Polytechnic University. Industrial Cybernetics*, 2026, vol. 4, no. 1, pp. 35–41. http://doi.org/10.18799/29495407/2026/1/118

Conflict of interest: none.

Введение

Язык программирования Python [1] на сегодняшний день является одним из самых популярных инструментов в области анализа данных и построения аналитических систем. Это связано с его простотой, большим количеством готовых библиотек и удобством интеграции с различными системами хранения данных. Python активно используется для реализации ETL-процессов, подготовки данных, написания сервисной логики и взаимодействия с аналитическими базами данных, такими как ClickHouse [2].

Несмотря на все преимущества, у Python есть важное ограничение, которое напрямую влияет на его производительность в многопоточных задачах, – это глобальная блокировка интерпретатора (ГИТ) или Global Interpreter Lock (GIL) [3]. Этот механизм гарантирует, что в каждый момент времени только один поток выполняет байткод Python. С одной стороны, это упрощает реализацию интерпретатора и обеспечивает потокобезопасность, но с другой – сильно ограничивает использование многопоточности, особенно в CPU-интенсивных задачах. В результате разработчики часто вынуждены использовать альтернативные подходы, такие как multiprocessing, который обходит ограничение GIL за счёт запуска отдельных процессов, но при этом добавляет накладные расходы на создание процессов и обмен данными между ними.

В последние годы в сообществе Python активно обсуждается и развивается идея отказа от GIL [4]. Одним из ключевых направлений стало появление так называемого free-threaded Python – версии интерпретатора, в которой глобальная блокировка отсутствует. Теоретически это позволяет эффективно использовать многопоточность и получать реальное ускорение за счёт параллельного выполнения кода. Однако на практике остаётся открытым вопрос, насколько это действительно улучшает производительность в реальных сценариях, а не только в синтетических тестах.

Особый интерес представляют аналитические системы, где нагрузка обычно смешанная: с одной стороны, выполняются запросы к базе данных (например, SELECT), с другой – происходит обработка полученных данных, а затем результаты записываются обратно (INSERT). В таких сценариях одновременно присутствуют как операции ввода–вывода (I/O-bound), так и вычислительные задачи (CPU-bound). Это делает поведение различных моделей параллелизма менее очевидным и требует отдельного исследования.

Таким образом, возникает необходимость сравнить эффективность различных подходов к параллелизму в Python – однопоточного выполнения, многопоточности и многопроцессности – в условиях реальной аналитической нагрузки. Под многопоточностью в данном контексте понимается выполнение нескольких потоков в рамках одного процесса с общей областью памяти, что упрощает обмен данными между задачами, но в Python ограничивается механизмом GIL. В свою очередь, многопроцессность предполагает запуск нескольких независимых процессов, каждый из которых имеет собственное адресное пространство и интерпретатор, что позволяет обойти ограничения GIL, но приводит к дополнительным накладным расходам на создание процессов и межпроцессное взаимодействие. Кроме того, важно оценить, как на результаты влияет наличие или отсутствие GIL.

В данной работе исследуется влияние моделей параллелизма и механизма GIL на производительность Python-приложений при работе с аналитической базой данных. Основное внимание уделяется сценариям с конкурентной нагрузкой, включающим выполнение запросов, обработку данных и запись результатов, а также сравнению поведения стандартного интерпретатора Python и его free-threaded версии.

Постановка задачи

В рамках данной работы рассматривается система, в которой Python используется для выполнения операций над данными при взаимодействии с аналитической базой данных ClickHouse.

Сценарий обработки включает три основных этапа: чтение данных из базы данных (SELECT), обработку на стороне Python и последующую запись результатов обратно (INSERT). Такая схема отражает типичный рабочий процесс в аналитических задачах, где Python используется как слой обработки между источником данных и хранилищем результатов.

Ключевой особенностью является выполнение этих операций в условиях конкурентной нагрузки. Несколько задач могут выполняться одновременно, что требует выбора подходящей модели параллелизма и эффективного использования доступных вычислительных ресурсов.

В работе рассматриваются три варианта организации выполнения:

- однопоточное выполнение;
- многопоточность (несколько потоков в рамках одного процесса с общей памятью);
- многопроцессность (несколько независимых процессов с раздельной памятью).

Для оценки поведения системы используются различные типы нагрузки: с преобладанием операций чтения, записи и смешанный вариант. Это позволяет проанализировать, как выбранная модель параллелизма влияет на производительность в зависимости от характера задач.

Целью исследования является сравнение эффективности указанных моделей выполнения при работе с базой данных, а также анализ влияния GIL на итоговую производительность.

Модели параллелизма

Для выполнения задач в Python можно использовать несколько подходов к организации параллелизма. В данной работе рассматриваются три наиболее распространённые модели: однопоточное выполнение, многопоточность и многопроцессность [5]. Они по-разному используют ресурсы системы и ведут себя по-разному в зависимости от типа нагрузки.

Однопоточное выполнение (single-thread) – это самый простой вариант, при котором все операции выполняются последовательно в одном потоке. В этом случае отсутствуют накладные расходы, связанные с созданием потоков или процессов, синхронизацией и передачей данных между ними. Такая модель обычно показывает стабильное и предсказуемое поведение, но при этом не позволяет использовать несколько ядер процессора. Поэтому при увеличении нагрузки производительность начинает ограничиваться возможностями одного потока.

Многопоточность (multithreading) предполагает запуск нескольких потоков внутри одного процесса. Все потоки работают в общем адресном пространстве, поэтому обмен данными между ними происходит достаточно просто и быстро. На первый взгляд это выглядит как удобный способ ускорить выполнение задач. Однако в Python есть ограничение в виде GIL, из-за которого в каждый момент времени фактически исполняется только один поток Python-кода. В результате многопоточность не даёт существенного выигрыша в задачах, связанных с вычислениями. При этом она может быть полезна в сценариях, где много операций ввода–вывода, например при работе с базой данных, так как во время ожидания ответа один поток может «уступить» выполнение другому.

Многопроцессность (multiprocessing) решает эту проблему за счёт использования нескольких процессов вместо потоков. Каждый процесс имеет собственный интерпретатор и не зависит от других, поэтому ограничение GIL на него не распространяется. Это позволяет действительно выполнять задачи параллельно на нескольких ядрах процессора. Однако за это приходится платить: создание процессов требует больше ресурсов, а обмен данными между ними сложнее и медленнее, так как используется межпроцессное взаимодействие. В задачах, где нужно часто передавать данные между исполнителями, это может стать узким местом.

Таким образом, выбор модели параллелизма зависит от конкретного сценария. Однопоточное выполнение подходит для простых или небольших задач, многопоточность – для I/O-нагруженных сценариев, а многопроцессность – для вычислительно сложных задач, где важно задействовать несколько ядер процессора. В рамках данной работы все три подхода рассматриваются и сравниваются на одинаковой нагрузке, что позволяет оценить их поведение в реальных условиях.

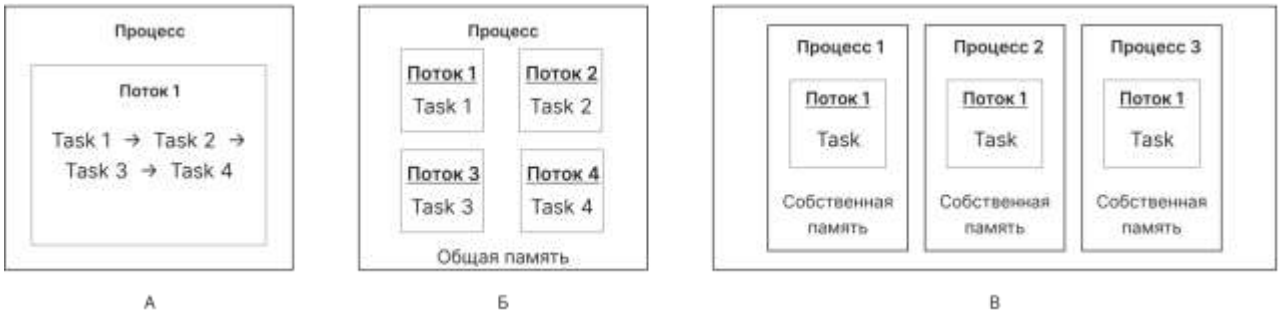

Рис. 1. *Модели параллелизма: однопоточное выполнение (А), многопоточность в рамках одного процесса с общей памятью (Б) и многопроцессность с отдельными адресными пространствами (В)*

Fig. 1. *Models of parallelism: single-threaded execution (A), multithreading within a single process with shared memory (Б), and multiprocessing with separate address spaces (B)*

Цель, методика и реализация эксперимента

Целью эксперимента являлось сравнение эффективности различных моделей параллелизма в среде Python 3.14 при выполнении задач конкурентного взаимодействия с аналитической СУБД. Рассматривались последовательное выполнение, выполнение с использованием потоков и выполнение с использованием процессов. Реализация выполнялась с использованием модулей стандартной библиотеки Python – threading и multiprocessing.

Экспериментальная нагрузка формировалась как последовательность операций извлечения, обработки и записи данных. В рамках одной задачи выполнялось получение подмножества строк из исходной таблицы, последующая обработка и запись результата в отдельную таблицу. Каждая задача соответствовала фиксированному диапазону идентификаторов, что определяло обрабатываемый набор данных. Задачи не пересекались и выполнялись независимо.

Рассматривались два варианта нагрузки, различающиеся соотношением операций чтения и записи. В первом случае преобладала обработка извлеченных данных при ограниченном объеме записи. Во втором случае увеличивался объем записываемых данных, что приводило к росту доли операций записи. При этом структура выполнения задач оставалась неизменной.

Степень параллелизма задавалась числом одновременно выполняемых задач и принимала значения от 1 до 16. В каждой конфигурации использовался фиксированный набор из 32 задач. Распределение задач определялось выбранной моделью выполнения. При последовательном выполнении задачи обрабатывались поочередно, при использовании потоков распределялись между потоками одного процесса, при использовании процессов – между несколькими процессами.

Все запуски выполнялись при одинаковых входных данных и неизменных параметрах их генерации, что обеспечивало сопоставимость результатов. Объем исходных данных составлял 200000 записей для каждого сценария. Эксперимент проводился сериями итераций для каждой конфигурации. Каждая серия включала этап предварительного выполнения, необходимый для стабилизации состояния системы, и этап измерения, в рамках которого фиксировались значения метрик. Для каждой итерации регистрировались время выполнения и пропускная способность. Во всех зафиксированных запусках ошибки выполнения отсутствовали.

Эксперименты выполнялись на вычислительной системе со следующими характеристиками:

- процессор: Apple M4;
- оперативная память: 24 ГБ;
- операционная система: macOS Sequoia 15.6.

Во всех запусках использовалась одна и та же аппаратная и программная конфигурация.

Результаты

Результаты измерений для режима выполнения с отключенным GIL приведены в таблице. В качестве основного показателя использовалось среднее время выполнения одной итерации эксперимента, рассчитанное по серии измерительных запусков. Дополнительно приводится пропускная способность, выраженная в количестве обработанных задач в секунду.

Таблица. *Результаты измерений*

Table. *Measurement results*

Сценарий Scenario	Режим Mode	Число задач Number of tasks	Среднее время, с Average time, s	Пропускная способность, tasks/s Throughput, tasks/s
Чтение/Reading	single	1	0,512	62,512
	threading	1	0,500	64,052
		2	0,288	111,028
		4	0,205	156,171
		8	0,191	168,463
		16	0,257	126,924
	multiprocessing	1	0,734	43,774
		2	0,425	75,459
		4	0,349	92,627
		8	0,441	73,081
		16	0,547	58,630
Запись/Recording	single	1	0,413	77,550
	threading	1	0,435	73,964
		2	0,306	105,877
		4	0,175	183,205
		8	0,153	210,307
		16	0,199	166,371
	multiprocessing	1	0,555	58,421
		2	0,365	88,371
		4	0,293	109,174
		8	0,340	94,441
		16	0,496	64,525

В сценарии чтения рост производительности при увеличении числа задач в многопоточной модели связан с перекрытием операций ввода–вывода при работе с СУБД [6]. Потоки совмещают отправку запросов и ожидание ответа, не допуская простоев. После достижения определенного значения числа задач наблюдается снижение эффективности, что связано с насыщением системы: ограничениями пропускной способности СУБД и ростом накладных расходов на планирование и синхронизацию потоков. Для многопроцессной модели дополнительным фактором выступают накладные расходы на создание процессов и межпроцессное взаимодействие, что ограничивает масштабирование [7]. Последовательное выполнение не использует параллелизм и демонстрирует стабильные значения времени выполнения.

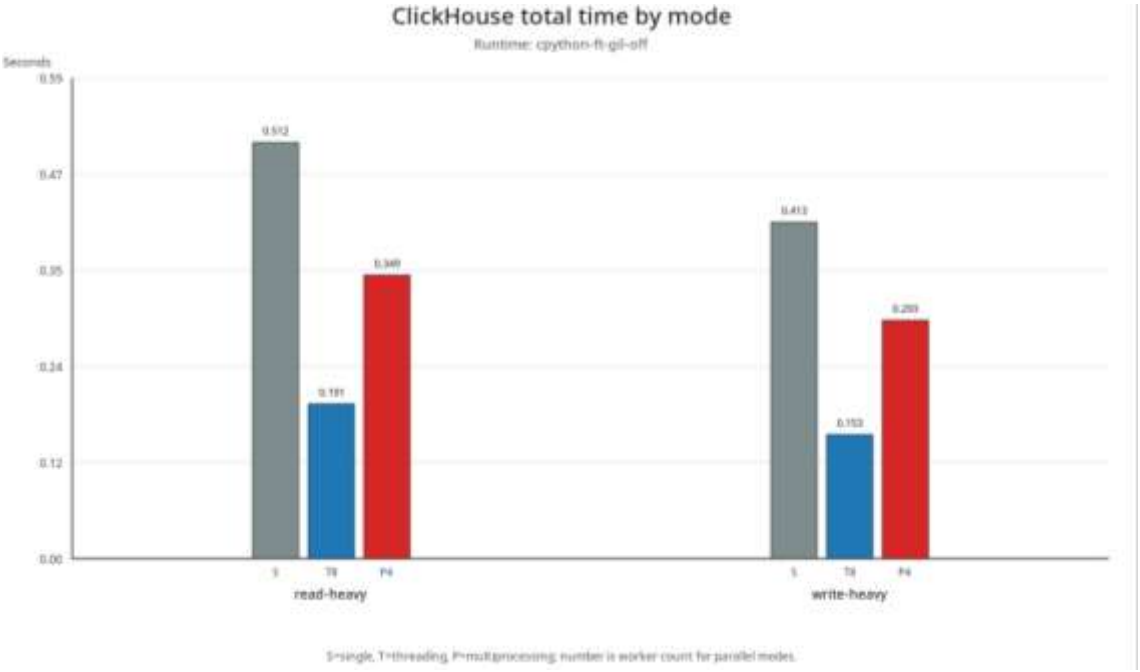

Рис. 2. *Время чтения и записи для различных моделей параллелизма*
Fig. 2. *Read and write times for different concurrency models*

В сценарии записи многопоточная модель также показывает рост производительности за счет параллельного выполнения операций записи и ожидания ответа от СУБД. При увеличении числа задач происходит насыщение системы, связанное с ограничениями подсистемы ввода–вывода и самой СУБД, а также увеличением накладных расходов на управление потоками. В многопроцессной модели дополнительные издержки на обмен данными между процессами и управление процессами приводят к более раннему достижению предела масштабируемости и последующему снижению производительности.

Преимущество многопоточной модели по сравнению с многопроцессной обусловлено характером нагрузки. Основная часть времени выполнения приходится на операции ввода–вывода [8] при взаимодействии с СУБД, поэтому возможность перекрытия этих операций в потоках дает выигрыш. Использование процессов не дает дополнительного эффекта, так как вычислительная часть задачи невелика, при этом возрастает стоимость создания процессов и межпроцессного обмена данными.

Выводы

В режиме выполнения с отключенным GIL многопоточная модель показывает наилучшие результаты в обоих сценариях. Однако влияние отключения GIL ограничено характером нагрузки [9]. При преобладании операций ввода–вывода выигрыш от параллельного выполнения потоков достигается и без значительной вычислительной конкуренции. Увеличение числа задач приводит к росту производительности до момента насыщения ресурсов, после чего наблюдается снижение эффективности [10].

Многопроцессная модель обеспечивает ускорение относительно последовательного выполнения, однако масштабируется хуже из-за накладных расходов на создание процессов и межпроцессное взаимодействие. Максимальные значения достигаются при меньшем числе задач, после чего рост издержек приводит к ухудшению показателей. Последовательное выполнение не использует параллелизм и во всех конфигурациях уступает другим моделям.

Сравнение моделей выполнения показывает, что многопоточная модель более эффективно использует ресурсы в условиях рассматриваемой нагрузки. Оптимальные значения степени параллелизма различаются: для многопроцессной модели они достигаются при меньшем числе задач из-за более высоких накладных расходов, тогда как многопоточная модель допускает большее число одновременно выполняемых задач до момента насыщения системы.

В качестве отдельного направления рассматривается сравнение полученных результатов с асинхронной моделью выполнения (asyncio), что позволит оценить её поведение в условиях аналогичной нагрузки.

СПИСОК ЛИТЕРАТУРЫ
1. *Python Documentation*. URL: https://docs.python.org/3/ (дата обращения: 09.03.2026).
2. *ClickHouse Documentation*. URL: https://clickhouse.com/docs (дата обращения: 09.03.2026).
3. Global Interpreter Lock (GIL). *Python Documentation*. URL: https://docs.python.org/3/glossary.html#term-global-interpreter-lock (дата обращения: 09.03.2026).
4. *PEP 703 – Making the Global Interpreter Lock Optional in CPython*. URL: https://peps.python.org/pep-0703/ (дата обращения: 09.03.2026).
5. Concurrent Execution. *Python Documentation*. URL: https://docs.python.org/3/library/concurrency.html (дата обращения: 09.03.2026).
6. Жолдыбай А., Айтуов А. Параллелизм в python для высоконагруженной обработки на многопроцессорных системах. *Universum: технические науки*, 2025, № 5 (134), С. 33–35. URL: https://7universum.com/ru/tech/archive/item/20073 (дата обращения: 09.03.2026).
7. *Python Multiprocessing vs Multithreading*. URL: https://www.bairesdev.com/blog/python-multiprocessing-vs-multithreading/ (дата обращения: 09.03.2026).
8. Threading – Thread-based parallelism. *Python Documentation*. URL: https://docs.python.org/3/library/threading.html (дата обращения: 09.03.2026).
9. *Python behind the scenes #13: the GIL and its effects on Python multithreading*. URL: https://tenthousandmeters.com/blog/python-behind-the-scenes-13-the-gil-and-its-effects-on-python-multithreading/ (дата обращения: 09.03.2026).
10. *Mitigating GIL Bottlenecks in Edge AI Systems*. URL: https://arxiv.org/abs/2601.10582 (дата обращения: 09.03.2026).

Информация об авторах
Антон Юрьевич Демин, кандидат технических наук, доцент отделения информационных технологий Инженерной школы информационных технологий и робототехники, Национальный исследовательский Томский политехнический университет, Российская Федерация, 634050, г. Томск, пр. Ленина, 30; доцент кафедры теоретических основ информатики Института прикладной математики и компьютерных наук, Томский государственный университет, Российская Федерация, 634050, Томск, пр. Ленина, 36; ad@tpu.ru; https://orcid.org/0000-0001-7071-1711
Дарья Игоревна Дмитрийчук, магистрант Факультета программной инженерии и компьютерной техники, Национальный исследовательский университет ИТМО, Российская Федерация, 197101, г. Санкт-Петербург, Кронверкский пр., 49, лит. А; d.dmitriychuk@gmail.com
Владислав Александрович Зарубин, магистрант Факультета программной инженерии и компьютерной техники, Национальный исследовательский университет ИТМО, Российская Федерация, 197101, г. Санкт-Петербург, Кронверкский пр., 49, лит. А; vladislavzarubin113@gmail.com

Поступила: 10.03.2026
Принята: 18.03.2026
Опубликована: 30.03.2026

REFERENCES
1. *Python Documentation*. Available at: https://docs.python.org/3/ (accessed: 9 March 2026).
2. *ClickHouse Documentation*. Available at: https://clickhouse.com/docs (accessed: 9 March 2026).
3. Global Interpreter Lock (GIL). *Python Documentation*. Available at: https://docs.python.org/3/glossary.html#term-global-interpreter-lock (accessed: 9 March 2026).
4. *PEP 703 – Making the Global Interpreter Lock Optional in CPython*. Available at: https://peps.python.org/pep-0703/ (accessed: 9 March 2026).
5. Concurrent Execution. *Python Documentation*. Available at: https://docs.python.org/3/library/concurrency.html (accessed: 9 March 2026).
6. Zholdybay A., Aituov A. Python concurrency for high-load multicore processing. *Universum: technical sciences*, 2025, no. 5 (134), pp. 33–35. Available at: https://7universum.com/ru/tech/archive/item/20073 (accessed: 9 March 2026).
7. *Python Multiprocessing vs Multithreading*. Available at: https://www.bairesdev.com/blog/python-multiprocessing-vs-multithreading/ (accessed: 9 March 2026).
8. Threading – Thread-based parallelism. *Python Documentation*. Available at: https://docs.python.org/3/library/threading.html (accessed: 9 March 2026).
9. *Python behind the scenes #13: the GIL and its effects on Python multithreading*. Available at: https://tenthousandmeters.com/blog/python-behind-the-scenes-13-the-gil-and-its-effects-on-python-multithreading/ (accessed: 9 March 2026).
10. *Mitigating GIL Bottlenecks in Edge AI Systems*. Available at: https://arxiv.org/abs/2601.10582 (accessed: 9 March 2026).

Information about the authors
Anton Yu. Demin, Cand. Sci. (Techn.), Associate Professor, National Research Tomsk Polytechnic University, 30, Lenin avenue, Tomsk, 634050, Russian Federation; Associate Professor, National Research Tomsk State University, 36, Lenin avenue, Tomsk, 634050, Russian Federation; ad@tpu.ru; https://orcid.org/0000-0001-7071-1711
Darya I. Dmitriychuk, Master's Student, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, 49, bldg. A, Kronverksky avenue, St Petersburg, 197101, Russian Federation; d.dmitriychuk@gmail.com

Vladislav A. Zarubin, Master's Student, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, 49, bldg. A, Kronverksky avenue, St Petersburg, 197101, Russian Federation; vladislavzarubin113@gmail.com

Received: 10.03.2026
Revised: 18.03.2026
Accepted: 30.03.2026