ЗАДАЧА ГЕНЕРАЦИИ КОДА ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (НА ПРИМЕРЕ POLKIT)

Назимов А. М.[1]

DOI:10.21681/3034-4050-2026-2-43-49

Ключевые слова: *большие языковые модели, Polkit, Text2DSL, структурированный контекст, валидация программ.*

Цель исследования: *формализация задачи автоматической генерации кода предметно-ориентированных языков (DSL) по описанию на естественном языке (Text2DSL) как самостоятельного класса задач и эмпирическая оценка роли структурированного контекста при генерации DSL-кода большой языковой моделью.*

Методы исследования: *эксперимент с двумя условиями (базовый режим и режим с контекстом) на датасете PolkitBench (4 204 верифицированные пары «запрос на естественном языке – правило Polkit»), трёхуровневая AST-валидация через парсер esprima, метрики синтаксической и семантической корректности.*

Результаты исследования: *включение структурированного контекста (BNF-грамматика, API-спецификация, словарь допустимых идентификаторов) повышает синтаксическую корректность с 80,5 % до 99,4 % (+23,4 %), семантическую корректность – с 60,4 % до 95,9 % (+58,7 %). Для класса задач Text2DSL включение формальной спецификации целевого языка в контекст запроса является необходимым и достаточным условием качественной генерации без дообучения модели.*

Научная новизна: *формализация задачи Text2DSL как отдельного класса задач генерации кода; датасет PolkitBench (4 204 верифицированные пары, трёхуровневая AST-валидация); эмпирическое обоснование критической роли структурированного контекста для качественной генерации DSL-кода.*

Введение

Управление политиками безопасности в современных программных системах реализуется посредством специализированных предметно-ориентированных языков (Domain-Specific Languages, DSL). К числу таких языков относятся Polkit – система авторизации привилегированных операций в Linux, SELinux policy language для мандатного управления доступом, OPA Rego для описания политик в облачной инфраструктуре, а также Terraform HCL для декларативного описания инфраструктуры.

Практическая разработка правил на указанных языках требует от администратора одновременного владения синтаксисом DSL, знанием допустимых идентификаторов API и пониманием семантики целевой системы. Ошибка в правиле может приводить как к отказу в обслуживании легитимных пользователей, так и к критическим нарушениям политики безопасности, включая несанкционированную эскалацию привилегий.

Существенный прогресс больших языковых моделей (БЯМ) в задачах генерации программного кода [1–6], а также в смежных задачах семантического преобразования текста в формальные представления (например, Text-to-SQL) [3, 12], создаёт предпосылки для автоматизации разработки DSL-правил на основе естественно-языковых описаний.

Однако между генерацией кода общего назначения и генерацией программ на DSL существует принципиальное различие. В большинстве практических случаев DSL характеризуется:

1) компактной и строго определённой формальной грамматикой;
2) конечным словарём допустимых идентификаторов;
3) детерминированной семантикой операторов, задаваемой спецификацией системы.

Указанные свойства позволяют формализовать задачу генерации DSL-кода как задачу отображения естественно-языкового описания в программу, удовлетворяющую строгим синтаксическим и семантическим ограничениям. В настоящей работе данная постановка обозначается термином Text2DSL.

Основные вклады настоящей работы заключаются в следующем:

1. Формализована задача Text2DSL как самостоятельный класс задач генерации кода с формальными требованиями синтаксической и семантической корректности.
2. Предложен и описан датасет PolkitBench, содержащий 4 204 верифицированные пары «естественно-языковой запрос – правило Polkit», прошедшие многоуровневую AST-валидацию.

1 Назимов Александр Михайлович, сотрудник Академии Федеральной службы охраны Российской Федерации. Орел, Российская Федерация. E-mail: s-nazim@list.ru

3. Экспериментально показано, что включение структурированного контекста (BNF-грамматика, API-спецификация и словарь допустимых идентификаторов) является критическим фактором качества генерации DSL-кода при использовании предварительно обученных языковых моделей без дополнительного дообучения.

Постановка задачи Text2DSL

Пусть Q – множество запросов на естественном языке, G – формальная грамматика целевого DSL, V – фиксированный словарь допустимых идентификаторов (API-имена, константы, перечисления). Задача Text2DSL определяется как построение отображения $f: Q \to P_g$, где P_g – множество программ, допустимых грамматикой G, такого что для произвольного запроса $q \in Q$ порождённая программа $p = f(q)$ удовлетворяет двум условиям:

1. Синтаксическая корректность: $\mathrm{parse}(p,G) = \mathrm{OK}$, т. е. программа p допускает разбор в абстрактное синтаксическое дерево (AST) согласно грамматике G.
2. Семантическая корректность: $\mathrm{ids}(p) \subseteq V$, т.е. все идентификаторы, используемые в p, принадлежат словарю V.

Таким образом, задача Text2DSL может рассматриваться как частный случай семантического парсинга с жёсткими формальными ограничениями на пространство допустимых программ.

Задача Text2DSL отличается от общей задачи генерации программного кода по следующим причинам:

1. Фиксированная грамматика. Целевой DSL описывается компактной BNF-грамматикой с конечным числом продукций. Для Polkit грамматика содержит около 10 правил, тогда как грамматика языков общего назначения (например Python) включает сотни продукций.
2. Конечный словарь идентификаторов. Множество допустимых идентификаторов заранее известно и ограничено. В случае Polkit $|V| \approx 65$ (категории Systemd, login1, PackageKit, NetworkManager и др.).
3. Детерминированная семантика. Каждый элемент API имеет строго определённую интерпретацию. Например, значение polkit.Result.AUTH_SELF однозначно соответствует требованию аутентификации текущего пользователя и не допускает альтернативных интерпретаций.

Применение больших языковых моделей для задачи Text2DSL

Современные подходы к генерации кода с использованием БЯМ можно классифицировать следующим образом:

1. Zero-shot generation (генерация без примеров) – модель получает только текстовое описание задачи без примеров или дополнительной информации о целевом языке. Такой подход продемонстрирован на моделях GPT-3, Codex и StarCoder [2].
2. Few-shot prompting (генерация с несколькими примерами) – во входной запрос включается несколько примеров пар «вход – выход» [3, 12]. Подход эффективен для задач с регулярной структурой, однако ограничен размером контекстного окна.
3. Fine-tuning (дообучение) – адаптация модели на целевом корпусе, включая методы LoRA и QLoRA [9, 10]. Требует наличия размеченного набора данных и вычислительных ресурсов.
4. Context injection (включение контекста) – добавление во входной запрос специализированной информации: документации API, грамматики, справочников допустимых значений. Не требует дообучения модели.
5. Retrieval-Augmented Generation (генерация с извлечением информации) – извлечение релевантных фрагментов из базы знаний на этапе применения модели [3, 11].

В отсутствие явной спецификации DSL модель демонстрирует несколько характерных типов ошибок:

- псевдокод вместо DSL – модель порождает текст на естественном языке или декларативные конструкции вместо валидного JavaScript;
- генерация несуществующих элементов API – генерируются несуществующие методы и свойства (response.allow(), polkit.grant(), action.verb и т. д.);
- генерация несуществующих идентификаторов – значения `action.id` незначительно отличаются от реальных: `filesystem-disk-mount` вместо `filesystem-mount`.

Причина: обучающий корпус содержит гетерогенную информацию о множестве API, поэтому модель «восстанавливает» идентификаторы по статистическим закономерностям, не имея полной и непротиворечивой спецификации целевого DSL.

Гипотеза настоящего исследования состоит в том, что для класса задач Text2DSL включение во входной запрос структурированного контекста, содержащего:

- BNF-грамматику целевого DSL (G);
- API-спецификацию (допустимые свойства, методы, возвращаемые значения);
- словарь идентификаторов (V), является необходимым и достаточным условием для существенного повышения качества генерации без дообучения модели. Контекст выполняет функцию эпистемического ограничения: он задаёт замкнутое пространство допустимых конструкций и запрещает модели использовать знания из обучающего корпуса, не соответствующие спецификации.

С эпистемологической точки зрения данное ограничение можно интерпретировать как введение априорного знания о структуре целевого языка. Большая языковая модель в процессе генерации кода опирается на статистические закономерности обучающего корпуса, которые отражают совокупный опыт программирования на различных языках и использования многочисленных API. В отсутствие явной спецификации модель формирует гипотезы о структуре программы на основе вероятностных ассоциаций, что приводит к появлению синтаксически похожих, но фактически некорректных конструкций.

Включение формальной грамматики и спецификации API в контекст запроса выполняет функцию эпистемического ограничения пространства допустимых гипотез. Модель оказывается ограниченной набором допустимых конструкций, определённых спецификацией DSL, что переводит процесс генерации из режима статистической аппроксимации знаний в режим контекстно-ограниченного вывода. В результате пространство возможных программ существенно сужается, что приводит к росту синтаксической и семантической корректности генерируемого кода.

Датасет PolkitBench

Целевым языком набора данных является Polkit – подмножество JavaScript (интерпретатор SpiderMonkey), используемое для описания правил авторизации привилегированных операций в Linux. API Polkit включает:

- action.id – строковый идентификатор действия (например, org.freedesktop.login1.reboot);
- subject.user – имя пользователя, запрашивающего авторизацию;
- subject.local, subject.active – булевы признаки локальности и активности сессии;
- subject.isInGroup(name) – метод проверки принадлежности к группе;
- polkit.Result.* – перечисление исходов: YES, NO, AUTH_SELF, AUTH_ADMIN.

Набор данных PolkitBench формируется в три этапа:

1. Шаблонная генерация – 50 шаблонов на русском и английском языках, параметризованных значениями групп, пользователей и действий, дают 5 000 уникальных запросов.
2. Генерация эталонных правил – для каждого запроса БЯМ Grok-4.1-fast (temperature= 0,0) генерирует правило Polkit, используя BNF-грамматику, API-спецификацию и механизм вызова функций.
3. AST – верификация (парсер esprima, 3 уровня): синтаксис → структура (polkit.addRule, параметры, возврат polkit.Result.*) → семантика (валидность свойств action/subject).

   Итого 4 204 валидных пары (84,1 % от 5 000).

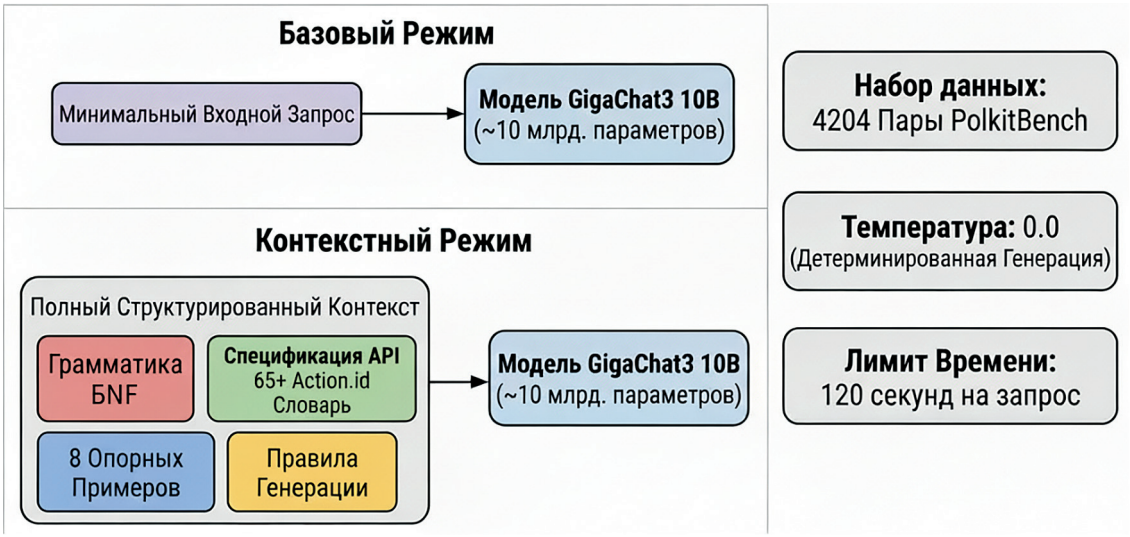

Рис. 1. Архитектура экспериментальной установки: сравнение базового и контекстного режимов генерации правил Polkit

Экспериментальные исследования

Для проверки гипотезы о критической роли структурированного контекста был проведён сравнительный эксперимент в двух режимах генерации: базовом (baseline) и режиме с контекстом (context-enhanced). Схема экспериментальной установки представлена на рисунке 1.

Во всех экспериментах на вход модели подавались текстовые запросы из датасета PolkitBench, представляющие собой описания правил авторизации на естественном языке. Для каждого из 4 204 запросов модель генерировала правило Polkit, которое затем сравнивалось с эталонным правилом из датасета и проходило автоматическую проверку корректности.

Базовый режим (baseline) – модель получает только минимальную инструкцию и текст запроса на естественном языке без дополнительной информации о синтаксисе или API целевого DSL.

Инструкция имеет следующий вид:

Ты генератор polkit правил для Linux. ФОРМАТ: Верни ТОЛЬКО код polkit правила в блоке ```javascript ... ```

Режим с контекстом (context-enhanced) – модель получает тот же запрос из датасета PolkitBench, однако дополнительно во входной контекст включается формализованная спецификация целевого DSL.

Контекст содержит:

- BNF-грамматику языка правил Polkit;
- API-спецификацию (допустимые свойства и методы);
- словарь допустимых идентификаторов action.id;
- восемь эталонных примеров корректных правил;
- правила генерации.

Модель получает следующую инструкцию:

Ты генератор polkit правил. Используй ТОЛЬКО информацию из контекста ниже. НЕ используй свои знания о polkit.

В эксперименте использовались все 4 204 верифицированные пары из датасета PolkitBench. Каждая пара включает текстовый запрос на естественном языке, описывающий требуемую политику авторизации, и соответствующее эталонное правило Polkit.

На этапе эксперимента в качестве входных данных модели использовались только текстовые запросы из датасета, тогда как эталонные правила применялись исключительно для последующей проверки корректности генерации. Для каждого запроса модель генерировала одно правило Polkit, после чего полученный код автоматически проверялся на синтаксическую и семантическую корректность посредством трёхуровневой AST -валидации.

Генерация выполнялась моделью GigaChat 3 10B при значении параметра temperature = 0,0, что обеспечивает детерминированный режим вывода и исключает стохастические вариации между запусками. Для каждого запроса устанавливалось ограничение времени генерации, равное 120 секундам.

Параметры эксперимента:

- модель: GigaChat 3 10B;
- набор данных: все 4 204 пары из PolkitBench;
- temperature: 0,0 (детерминированная генерация);
- ограничение по времени: 120 секунд на запрос.

Между условиями варьировалась только одна независимая переменная – наличие или отсутствие структурированного контекста во входном запросе.

В базовом режиме 19,5 % сгенерированных программ содержат синтаксические ошибки JavaScript. Для выявления причин возникновения этих ошибок был проведён качественный анализ некорректно сгенерированных правил Polkit. Анализ показал, что большинство ошибок связано с отсутствием у модели явного представления о синтаксисе и API целевого DSL, вследствие чего модель воспроизводит конструкции, статистически похожие на правила управления доступом, но не соответствующие реальной спецификации языка Polkit. По результатам анализа ошибки можно разделить на три основных подкласса.

1. Генерация псевдокода – модель порождает декларативные конструкции (allow {user = root;}) вместо императивного JavaScript.
2. Неверная структура – отсутствует обязательный вызов polkit.addRule() или некорректный формат callback-функции.

Таблица 1.

Сравнение условий Baseline и Context – enhanced (N = 4204)

Метрика	Базовый режим	С контекстом	Δ, %
Синтаксическая корректность	80,5 % (3386)	99,4 % (4178)	+23,4 %
Семантическая корректность	60,4 % (2540)	95,9 % (4031)	+58,7 %

3. Некорректные возвращаемые значения – вместо polkit.Result.YES модель генерирует polkit.Result.Denied, response.allow() и иные несуществующие конструкции.

Включение структурированного контекста практически полностью устраняет синтаксические ошибки: синтаксическая корректность достигает 99,4 %. Остаточные 0,6 % ошибок обусловлены превышением ограничения по времени и граничными случаями с вложенными условиями.

Обсуждение полученных результатов

Полученные результаты подтверждают гипотезу о критической роли структурированного контекста для задачи Text2DSL. Прирост семантической корректности на 58,7 % демонстрирует, что без явного задания спецификации модель не располагает достаточными сведениями о целевом DSL и подменяет его синтаксис конструкциями из обучающего корпуса (декларативный псевдокод, конструкции SELinux, собственные построения). Существенный рост точности воспроизведения строковых идентификаторов свидетельствует, что словарь V в контексте переключает модель от генерации произвольных идентификаторов к выбору из заранее определённого перечня допустимых значений.

К ограничениям подхода относятся следующие:

1. Необходимость формальной спецификации. Метод включения контекста работает только при наличии BNF-грамматики и словаря V. Для DSL без формальной документации построение контекста представляет нетривиальную задачу.
2. Масштабируемость словаря. В эксперименте $|V| \approx 65$. При $|V| > 10^3$ (например, для SELinux type enforcement) контекст может превысить допустимый размер контекстного окна модели. В таких случаях необходим гибридный подход (RAG + context injection).
3. Одна модель. Эксперимент проведён на единственной модели (~10B параметров). Для обобщения результатов необходимо исследование на моделях различного масштаба.
4. Остаточные галлюцинации. Даже при наличии контекста модель генерирует 1 413 невалидных action.id, что указывает на необходимость дополнительных механизмов (постобработка, grammar–constrained decoding [7, 11]).

Выводы

В настоящей работе формализована задача Text2DSL – автоматической генерации кода предметно-ориентированных языков по описанию на естественном языке – как самостоятельный класс задач генерации кода. Определены формальные требования синтаксической и семантической корректности, предложена система метрик (синтаксическая и семантическая корректность).

Представлен набор данных PolkitBench, содержащий 4 204 верифицированные пары «запрос – правило Polkit», прошедших трёхуровневую AST-валидацию.

Проведённый эксперимент показал, что включение структурированного контекста (BNF-грамматика, API-спецификация, словарь идентификаторов) во входной запрос БЯМ повышает синтаксическую корректность генерации с 80,5 % до 99,4 % (+23,4 %), семантическую корректность – с 60,4 % до 95,9 % (+58,7 %). Результаты демонстрируют, что для задач Text2DSL включение формальной спецификации целевого языка является ключевым фактором качества генерации, позволяющим достичь высоких показателей без дообучения модели.

В дальнейшем планируется: дообучение модели на PolkitBench (LoRA/QLoRA) для снижения числа ошибок; применение декодирования с грамматическими ограничениями для гарантии синтаксически корректного кода.

Литература

1. Rozière B., Gehring J., Gloeckle F., et al. Code Llama: Open Foundation Models for Code // arXiv. – 2023. – DOI: 10.48550/arXiv.2308.12950.
2. Li R., Allal L. B., Zi Y., et al. StarCoder: May the Source Be with You! // Transactions on Machine Learning Research. – 2023. – DOI: 10.48550/arXiv.2305.06161.
3. Zan D., Chen B., Zhang F., Lu D., Wu B., Guan B., Wang Y., Lou J.-G. Large Language Models Meet NL2Code: A Survey // Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL). – 2023. – P. 7443–7464. – DOI: 10.18653/v1/2023.acl-long.411.
4. Austin J., Odena A., Nye M., et al. Program Synthesis with Large Language Models // arXiv. – 2021. – DOI: 10.48550/arXiv.2108.07732.
5. Wang Y., Wang W., Joty S. CodeT5: Identifier-Aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation // Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2021. – DOI: 10.18653/v1/2021.emnlp-main.685.
6. Guo D., Ren S., Lu S., et al. GraphCodeBERT: Pre-training Code Representations with Data Flow // International Conference on Learning Representations (ICLR). – 2021. – DOI: 10.48550/arXiv.2009.08366.

7. Scholak T., Schucher N., Bahdanau D. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models // Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2021. – DOI: 10.18653/v1/2021.emnlp-main.779.
8. Hu E. J., Shen Y., Wallis P., et al. LoRA: Low-Rank Adaptation of Large Language Models // International Conference on Learning Representations (ICLR). – 2022. – DOI: 10.48550/arXiv.2106.09685.
9. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient Finetuning of Quantized Large Language Models // Advances in Neural Information Processing Systems (NeurIPS). – 2023. – DOI: 10.48550/arXiv.2305.14314.
10. Chen X., Lin Y., Schürmann C. Neural Code Generation with Grammar Constraints // International Conference on Learning Representations (ICLR). – 2021. – DOI: 10.48550/arXiv.2010.00904.
11. Zhong R., Yin P., Yu T., et al. Semantic Parsing for Code Generation: A Survey // Findings of the Association for Computational Linguistics (ACL Findings). – 2022. – DOI: 10.18653/v1/2022.findings-acl.2.
12. Yin P., Neubig G. StructCoder: Structure-Aware Code Generation with Language Models // Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL). – 2022. – DOI: 10.18653/v1/2022.acl-long.39.
13. Barke S., James M., Polikarpova N. Grounded Copilot: How Programmers Interact with Code-Generating Models // Proceedings of the International Conference on Software Engineering (ICSE). – 2023. – DOI: 10.1109/ICSE48619.2023.00015.
14. Pourreza M., Rafiei D. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction // Advances in Neural Information Processing Systems (NeurIPS). – 2023. – DOI: 10.48550/arXiv.2304.11015.

THE PROBLEM OF CODE GENERATING DOMAIN-SPECIFIC LANGUAGES USING LARGE LANGUAGE MODELS (USING POLKIT AS AN EXAMPLE)

***Nazimov A. M.*[2]**

Keywords: *large language models, Polkit, Text2DSL, structured context, program validation.*

Abstract

Purpose of the study: *to formalize the task of automatic generation of domain-specific language (DSL) code from natural language descriptions – referred to as Text2DSL – as an independent class of code generation problems, and to empirically evaluate the role of structured context in DSL code generation by a large language model.*

Methods of research: *the study is based on an experimental evaluation conducted under two conditions (baseline mode and context-enhanced mode) using the PolkitBench dataset, which contains 4,204 verified pairs of natural language requests and corresponding Polkit rules. A three-level AST validation procedure based on the esprima parser was employed, along with quantitative metrics of syntactic and semantic correctness.*

Results: *the inclusion of structured context (BNF grammar, API specification, and a dictionary of valid identifiers) increases syntactic correctness from 80.5 % to 99.4 % (+23.4 %) and semantic correctness from 60.4 % to 95.9 % (+58.7 %). The results demonstrate that for the class of Text2DSL tasks, incorporating the formal specification of the target language into the prompt context constitutes a necessary and sufficient condition for achieving high-quality DSL code generation without additional model fine-tuning.*

Scientific novelty: *the study formalizes the Text2DSL problem as a distinct class of code generation tasks; introduces the PolkitBench dataset consisting of 4,204 verified pairs validated through a three-level AST analysis; and provides empirical evidence of the critical role of structured context in enabling accurate DSL code generation by large language models.*

References

1. Rozière B., Gehring J., Gloeckle F., et al. Code Llama: Open Foundation Models for Code // arXiv. – 2023. – DOI: 10.48550/arXiv.2308.12950.
2. Li R., Allal L. B., Zi Y., et al. StarCoder: May the Source Be with You! // Transactions on Machine Learning Research. – 2023. – DOI: 10.48550/arXiv.2305.06161.
3. Zan D., Chen B., Zhang F., Lu D., Wu B., Guan B., Wang Y., Lou J.-G. Large Language Models Meet NL2Code: A Survey // Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL). – 2023. – P. 7443–7464. – DOI: 10.18653/v1/2023.acl-long.411.
4. Austin J., Odena A., Nye M., et al. Program Synthesis with Large Language Models // arXiv. – 2021. – DOI: 10.48550/arXiv.2108.07732.
5. Wang Y., Wang W., Joty S. CodeT5: Identifier-Aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation // Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2021. – DOI: 10.18653/v1/2021.emnlp-main.685.

2 Alexandr M. Nazimov, Academy of the Federal Security Service of the Russian Federation. Orel, Russian Federation. E-mail: s-nazim@list.ru

6. Guo D., Ren S., Lu S., et al. GraphCodeBERT: Pre-training Code Representations with Data Flow // International Conference on Learning Representations (ICLR). – 2021. – DOI: 10.48550/arXiv.2009.08366.
7. Scholak T., Schucher N., Bahdanau D. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models // Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2021. – DOI: 10.18653/v1/2021.emnlp-main.779.
8. Hu E. J., Shen Y., Wallis P., et al. LoRA: Low-Rank Adaptation of Large Language Models // International Conference on Learning Representations (ICLR). – 2022. – DOI: 10.48550/arXiv.2106.09685.
9. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient Finetuning of Quantized Large Language Models // Advances in Neural Information Processing Systems (NeurIPS). – 2023. – DOI: 10.48550/arXiv.2305.14314.
10. Chen X., Lin Y., Schürmann C. Neural Code Generation with Grammar Constraints // International Conference on Learning Representations (ICLR). – 2021. – DOI: 10.48550/arXiv.2010.00904.
11. Zhong R., Yin P., Yu T., et al. Semantic Parsing for Code Generation: A Survey // Findings of the Association for Computational Linguistics (ACL Findings). – 2022. – DOI: 10.18653/v1/2022.findings-acl.2.
12. Yin P., Neubig G. StructCoder: Structure-Aware Code Generation with Language Models // Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL). – 2022. – DOI: 10.18653/v1/2022.acl-long.39.
13. Barke S., James M., Polikarpova N. Grounded Copilot: How Programmers Interact with Code-Generating Models // Proceedings of the International Conference on Software Engineering (ICSE). – 2023. – DOI: 10.1109/ICSE48619.2023.00015.
14. Pourreza M., Rafiei D. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction // Advances in Neural Information Processing Systems (NeurIPS). – 2023. – DOI: 10.48550/arXiv.2304.11015.