Research article
DOI: https://doi.org/10.18721/JCSTCS.18206
UDC 004.89

IT PROJECT INFRASTRUCTURE SETUP AUTOMATION WITH HELP OF LARGE LANGUAGE MODELS

V.A. Ivlev ✉ , *I.V. Nikiforov* , *S.M. Ustinov*

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

✉ nevidd@yandex.ru

Abstract. This study conducts an analysis of existing large language models (LLMs) and AI agents, identifying Llama 2 as the most suitable model for automating IT project environment configuration. A mathematical model of the proposed method is introduced to automate IT infrastructure setup and reduce development time. The system architecture incorporates modules for natural language processing (NLP), configuration generation and command execution. The effectiveness of the method is evaluated through experiments across five key production scenarios, comparing two approaches: traditional infrastructure configuration tools and the proposed LLM-based method utilizing Llama 2. Experimental results demonstrate that the proposed method reduces configuration time up to 60%, decreases error rates from 25% to 8% and improves configuration quality approximately in 3 times. The article is relevant to IT professionals engaged in automating development and infrastructure configuration processes, as well as researchers exploring the application of artificial intelligence, particularly large language models, in the IT industry.

Keywords: Large Language Model, Llama 2, AI agent, IT infrastructure setup automation, Natural Language Processing, configuration generation, artificial intelligence

Citation: Ivlev V.A., Nikiforov I.V., Ustinov S.M. IT project infrastructure setup automation with help of large language models. Computing, Telecommunications and Control, 2025, Vol. 18, No. 2, Pp. 74–90. DOI: 10.18721/JCSTCS.18206

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.18206
УДК 004.89

АВТОМАТИЗАЦИЯ НАСТРОЙКИ ИНФРАСТРУКТУРЫ ИТ-ПРОЕКТА С ИСПОЛЬЗОВАНИЕМ LLM-МОДЕЛЕЙ

В.А. Ивлев ✉ , *И.В. Никифоров* , *С.М. Устинов*

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

✉ nevidd@yandex.ru

Аннотация. В исследовании проведен анализ существующих больших языковых моделей (LLM) и AI-агентов, на основе которого выбрана модель Llama 2 как наиболее подходящая для автоматизации настройки окружения ИТ-проекта. Предложена математическая модель метода, позволяющего автоматизировать процесс настройки ИТ-инфраструктуры и сократить время разработки ИТ-проекта. Разработана архитектура системы, включающая модули для обработки естественного языка (NLP), генерации конфигураций и выполнения команд. Оценена эффективность предложенного метода в экспериментах на пяти основных производственных сценариях. В ходе экспериментов сравнивались два подхода настройки ИТ-инфраструктуры: подход с использованием традиционных средств настройки инфраструктуры и подход с использованием предложенного в работе метода на основе LLM-модели Llama 2. Показано, что использование предложенного метода позволяет сократить время настройки до 60%, снизить количество ошибок с 25% до 8% и повысить качество настройки приблизительно в 3 раза. Статья представляет интерес для специалистов в области информационных технологий, занимающихся автоматизацией процессов разработки и настройки инфраструктуры, а также для исследователей, изучающих применение искусственного интеллекта, а именно больших языковых моделей, в ИТ-индустрии.

Ключевые слова: большая языковая модель, Llama 2, AI-агент, автоматизация настройки ИТ-инфраструктуры, обработка естественного языка, генерация конфигураций, искусственный интеллект

Для цитирования: Ivlev V.A., Nikiforov I.V., Ustinov S.M. IT project infrastructure setup automation with help of large language models // Computing, Telecommunications and Control. 2025. Т. 18, № 2. С. 74–90. DOI: 10.18721/JCSTCS.18206

Introduction

Nowadays software development plays a key role in many companies, as it enables the automation of processes, improves the quality of production lines and enhances the efficiency of management processes [1, 2]. At the same time, the software being developed grows larger in scale and more complex with each new task. As software complexity increases, companies face new challenges and management tasks related to configuring IT systems, aimed at maintaining software development and operational processes. This results in developers encountering a growing demand for effective tools to automate the configuration of IT infrastructure.

To address this problem, companies are trying to adopt methods, algorithms, approaches and systems capable of automating the configuration of IT project infrastructure. An optimal solution in this context could be a system that interprets human-readable task descriptions into command formats and automatically configures IT infrastructure by executing these commands. However, identifying or developing such a system is a non-trivial task. Many commercially available systems are proprietary and require skilled personnel [1, 3] to maintain or implement them.

To solve the problem of automating IT project infrastructure configuration a method based on a Natural Language Processing (NLP) model service was previously proposed. This method automates infrastructure configuration thereby accelerating various stages of IT project development [1]. The key component of this approach is its ability to interpret human-readable descriptions (unformalized text) into executable command sequences, which are then executed by the system to configure the IT infrastructure. However, the application of Large Language Models (LLMs) for NLP automation was not addressed, despite their potential to enhance the degree of automation and output quality.

To overcome this limitation, we propose an approach that employs an LLM as an interpreter to convert human-readable descriptions into executable commands. This serves as an implementation of the NLP architectural block within the method for automating IT infrastructure configuration in IT projects [3].

Relevance of the topic

Modern IT projects typically involve complex and diverse technologies [2], services and platforms [4, 5], including software and hardware components, such as automated workstation equipment, virtualization and containerization systems, infrastructure monitoring platforms, system-wide software (operating systems, Database Management System (DBMS), core infrastructure service software etc.), telephony and videoconferencing systems, data center engineering infrastructure, cybersecurity subsystems and others. Configuring such infrastructure is a labor-intensive task requiring specialized knowledge and expertise. Automating the deployment of IT infrastructure could potentially simplify environment configuration, thereby enhancing development productivity and reducing the time required to deliver the final product. Integrating tools for code generation based on natural language descriptions could further streamline this process, virtually eliminating the need for direct programmer involvement. Automation of IT infrastructure configuration using tools like LLMs has the potential to significantly reduce time [6]. LLMs can rapidly analyze project requirements formulated in natural language, propose optimal configurations and generate code for environment setup. A configuration process leveraging LLMs or LLM-based tools can minimize human errors associated with infrastructure setup, leading to more stable and reliable project operations.

LLMs are trained on huge datasets [7] enabling them to adapt to diverse use cases [8] and improve their performance over time [9]. This adaptability is particularly valuable in dynamic IT environments. Such LLM-driven systems can be applied to various aspects of IT infrastructure configuration, including parameter optimization [10], network setup, security management etc. This makes the natural-language-based code generation approach flexible and scalable for different project types. LLMs effectively generate code because of their big volume training data [11, 12], which incorporates code examples and technical documentation, allowing them to comprehend the syntax and logic of multiple programming languages [13]. They analyze query context to ensure solution accuracy and relevance and can interact with users to clarify requirements. Beyond code writing, LLMs can generate tests and documentation [14], positioning them as indispensable tools for developers [15]. Solutions employing LLMs to execute business-oriented tasks are termed AI agents.

Thus, the use of AI agents [16] for automating IT infrastructure configuration represents a promising approach that could substantially enhance the efficiency and reliability of IT projects. By automating operational tasks, this strategy frees human resources to focus on tactical and strategic challenges.

Degree of development of the topic and analysis of existing AI agents

Given the relevance of utilizing tools for code generation [17] based on natural language descriptions as a core module [18] of the proposed automation system, a study of existing AI agents and their underlying LLMs is conducted. Considering the specificity of the domain, a list of candidates (Table 1) potentially capable of solving code generation tasks from natural language descriptions is identified. It

Table 1

Comparative analysis of existing natural language-driven code generation tools

AI agent (LLM-model)	Model type	Model size	Architecture	Security and robustness	Broad applicability	Software integration	Open source	Multimodality	Computational optimization
Claude-3Opus	Model from Anthropic	Not disclosed	Transformer [19]	+	+	-	-	+	-
Gemini-1Ultra [20]	Model from Google	Not disclosed	Transformer	+	+	-	-	+	+
Mistral-Large	Model from Mistral	12.9B	Sparse Mixture of Experts	-	+	-	+	-	+
Llama 2 [21]	Model from Microsoft	7B, 13B, 70B	Transformer	-	+	+	+	-	+
Qwen-1.5	Model from Huawei	7B, 13B	Transformer	-	+	-	-	+	+
DeepSeek	Model from DeepSeek	Not disclosed	Transformer	-	+	+	-	-	+
Baichuan-2 Turbo	Model from Baichuan	7B, 13B	Transformer	-	+	-	-	-	+
Copilot	Model from OpenAI	Not disclosed	Transformer	-	+	+	-	-	+
Codex	Model from OpenAI	Not disclosed	Transformer	-	+	+	-	-	+
FauxPilot	Model from Replit	Not disclosed	Transformer	-	+	+	-	-	+
StarCoder	Model from BigCode	Not disclosed	Transformer	-	+	+	-	-	+
GPT-4-Turbo [22]	Model from OpenAI	Not disclosed	Transformer	+	+	+	-	-	+
EleutherAI GPT	Model from EleutherAI	1.3B, 2.7B, 6B, 20B	Transformer	-	+	-	+	-	+
Microsoft Turing	Model from Microsoft	Not disclosed	Transformer	+	+	-	-	-	+
IBM Watson NLU	Model from IBM	Not disclosed	Diverse NLP Models	+	+	-	-	-	+

is worth noting that alongside classical approaches, the Retrieval-Augmented Generation (RAG) methodology combines generative models with data retrieval from external sources to enhance code accuracy. However, its integration with LLMs necessitates a dedicated analysis of architectural considerations. This study focuses on evaluating the "pure" generative capabilities of the models, deferring the investigation of hybrid RAG-based systems to future work.

The study proposed limiting the comparative analysis to the most critical criteria for addressing IT environment configuration automation, namely: security and robustness of results, breadth of model applicability across diverse IT project domains, integration capabilities with third-party software, open-source availability, multimodality and optimization for internal computations within the model.

It should be noted that all analyzed AI agents and LLMs adequately account for and use context during task execution for code generation or NLP, where understanding broad context [23] can significantly enhance result quality [24]. At first glance, LLMs such as GPT-4-Turbo, Gemini-1Ultra and Microsoft Turing demonstrate broad applicability and can be employed for diverse tasks, ranging from text generation to integration with cloud services and business solutions. These models are universal and adaptable to a wide range of challenges. AI agents like Codex, Copilot and StarCoder are specialized in software development support, making them indispensable tools for programmers engaged in code automation and software solution creation. However, these criteria for LLMs and AI agents are of lesser significance as the listed tools lack open-source availability, precluding their modification and adaptation to task-specific requirements.

Thus, based on the conclusions drawn from the initial evaluation of LLMs and AI agents, the focus should shift to open-source tools such as Llama 2 [21], EleutherAI GPT and Mistral-Large. These provide researchers and developers with the ability to modify and adapt models for specialized tasks [25], which is critical for projects requiring model fine-tuning [26] for use in niche domains [27]. Notably, unlike EleutherAI GPT and Mistral-Large, Llama 2 offers software integration – specifically, functionality focused on programming assistance and code generation [28]. Considering this criterion alongside the aforementioned conclusions, Llama 2 considered as the preferred candidate for implementation as the script generation module in a system designed to automate IT infrastructure configuration for IT projects.

A method for automating IT project infrastructure configuration through the application of an LLM

The proposed method is formalized through the following parameters and objective function.

1. Method parameters:

D: set of all human-readable task descriptions;

C: set of all technical configurations;

$F(d)$: transformation function from task description $d \in D$ to configuration $c \in C$;

ε: probability that $F(d)$ is an incorrect configuration.

The set of valid technical configurations C_{corr} is defined as a subset C, where the probability of successful transformation exceeds a predefined threshold $1 - \varepsilon$:

$$T_{corr} = \left\{ c \in C \middle| \exists d \in D : F\left(d\right) = c \wedge P_{corr}\left(c \middle| d\right) \geq 1 - \varepsilon \right\},$$

where $P_{corr}(c|d)$ denotes the conditional probability, that configuration c generated from task description d, is correct.

2. Objective function – the target function minimizes infrastructure configuration time T_{auto}:

$$T_{auto} = T_{Llanna} + T_{exec},$$

where T_{Llanna} is the time spent generating the configuration; T_{exec} is the time required to execute the script on the actual infrastructure.

3. Model quality – model performance is evaluated using the accuracy metric:

$$Accuracy = \frac{\left|F\left(d\right) \cap F_{valid}\right|}{\left|D\right|},$$

where F_{valid} is the set of correct configurations.

Objective of the mathematical model – the proposed mathematical model aims to optimize the IT infrastructure automation process using Llama 2 LLM. Specifically, it focuses on:

$$M = \min\left(T_{auto}\right) \wedge \max\left(Accuracy\right) \wedge \min\left(\varepsilon\right),$$

that is, minimizing execution time (T_{auto}), increasing the accuracy of converting human-readable descriptions into valid configurations ($Accuracy$), reducing error probability (ε) and ensuring system stability. By effectively combining configuration generation and execution the model reduces reliance on software developer intervention, thereby enhancing productivity [30] and infrastructure configuration reliability. This method forms the foundation of the software system developed in this study to address automation challenges.

Implementation of the IT infrastructure automation method

To integrate the Llama 2 LLM into the IT infrastructure automation process, a dedicated architecture is designed (Fig. 1). This architecture comprises several core components aimed at efficiently converting human-readable task descriptions into executable infrastructure configuration commands.

The architectural elements are analyzed in detail below.

1. *Data input module*. This module is responsible for receiving and preprocessing human-readable task descriptions. Descriptions may be provided as text files, API requests or via user interface. The module also performs text normalization, stop-word removal and tokenization to prepare data for model processing.

As the technology stack of this module for user interaction, the following is used: user interaction is implemented via a REST interface using the Spring Boot 3 framework, enabling rapid deployment of a reliable server-side application. Input data validation (task descriptions) adheres to the Java API Bean Validation specification (JSR 380). Task storage is managed in PostgreSQL database using Spring Data JPA. For CLI integration, the Picocli tool is employed alongside the Spring Shell framework.

2. *NLP module*. This module leverages the Llama 2 LLM [31] to analyze and interpret textual descriptions. The model converts text requests into structured data for configuration generation and may request user clarification for ambiguous inputs.

The following is used as the technology stack: integration with Llama 2 is achieved via REST interface or gRPC using Spring WebClient or gRPC-Java libraries. Text preprocessing (tokenization, entity extraction) utilizes Apache OpenNLP, augmented with custom rules. Complex scenarios employ spaCy's language parser via Java bindings.

3. *Configuration generation module*. Using data from the NLP module, this module generates executable commands or scripts for infrastructure configuration. Outputs include configuration files [32], service deployment commands and network parameter setups. Generated configurations undergo validation before execution.

The following is used as the technology stack: configuration files (YAML, JSON, scripts) are templated using tools Apache Velocity or Thymeleaf.

4. *Command execution module*. This module executes generated commands on target infrastructure, such as cloud platforms, servers or containerized environments. It monitors execution status and provides feedback.

The following is used as the technology stack: secure SSH connections are established via library JSch. Asynchronous task handling uses CompletableFuture and Project Reactor. Cloud orchestration leverages Kubernetes Java client. Parallel task management employs ThreadPoolExecutor class or reactive streams.

5. *Monitoring and feedback module*. Post-execution, this module collects configuration results, including errors and warnings [33] to refine future configurations and improve model performance.

The following is used as the technology stack: metrics are exported to system Prometheus and Grafana via Micrometer tool. Logging is implemented with Log4j2 library, integrated into an ELK stack (Elasticsearch, Logstash, Kibana) [34]. Notifications are dispatched via Spring Integration framework, supporting tools like Slack, E-mail and Telegram. It is important to note that, the developed tool does not always produce fully accurate automation scripts. Generated outputs serve as templates that users may manually refine to meet specific requirements.

Example of method application

To provide a clear visualization of the method's workflow a diagram has been developed (Fig. 2), illustrating the interaction between the system's core modules. The diagram encompasses the following stages.

1. Data input. The user submits a textual task description via an interface or API.

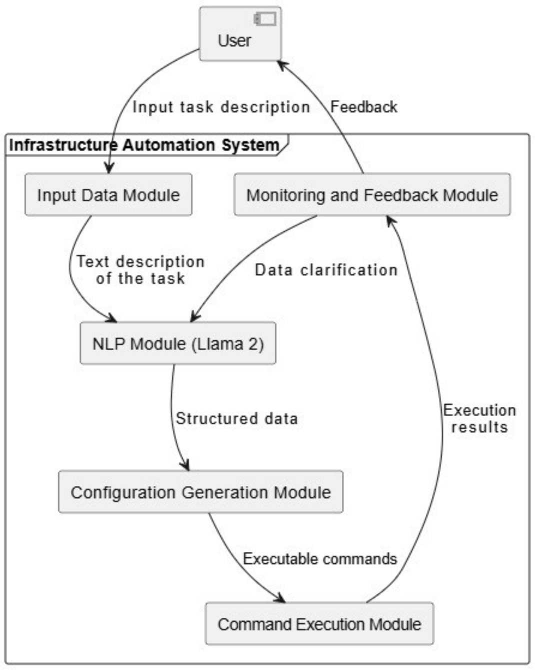


Fig. 1. Component diagram of the automation system with Llama 2 LLM integration

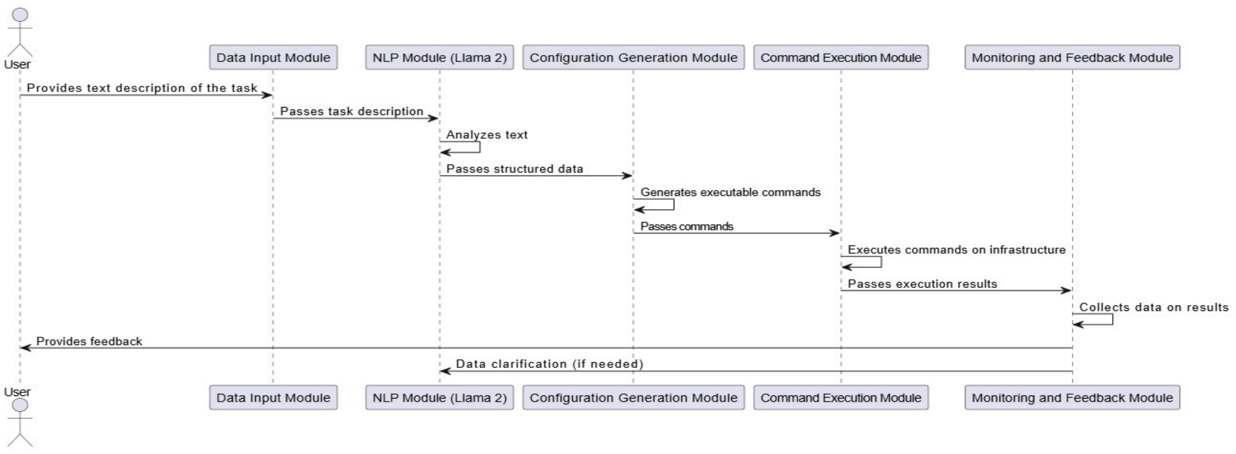


Fig. 2. MSC diagram of system component message exchange in the demonstration scenario

2. NLP. The Llama 2 model parses the text and converts it into structured data.

3. Configuration generation. Executable commands are generated based on the data derived from the model [35, 36].

4. Command execution. The commands are deployed on the target infrastructure.

5. Monitoring and feedback. The system collects execution results and delivers feedback to the user.

To demonstrate the method's application, consider an example of configuring infrastructure for a PostgreSQL database replication project (Fig. 3). Below is a flowchart of the method's workflow with each stage of the automation system's operation explained in detail.

1. *Data input*. The user submits the task description: "Configure PostgreSQL with replication across two servers: a primary server and a backup server." An example of system logging at this stage is provided below (listing 1).

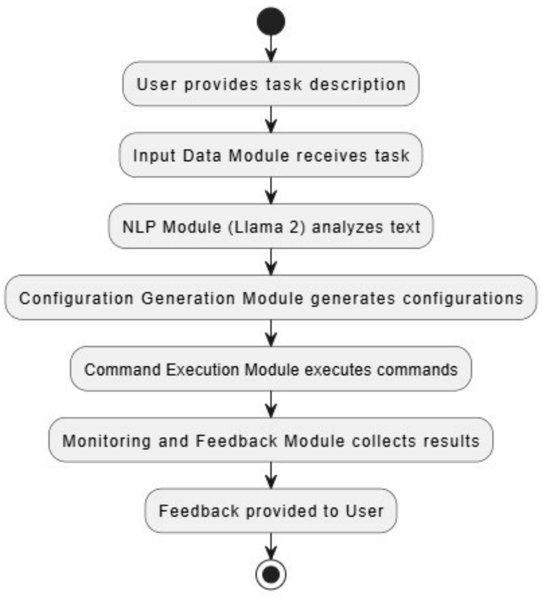

User provides task description
Input Data Module receives task
NLP Module (Llama 2) analyzes text
Configuration Generation Module generates configurations
Command Execution Module executes commands
Monitoring and Feedback Module collects results
Feedback provided to User

Fig. 3. Flowchart of the method's workflow

Listing 1. *Example of system logging during the data input stage.*

```
[INFO] [2025-02-20 21:59:45] User provided the task description:
"Set up PostgreSQL with replication on two servers: primary server and backup server."
```

2. *NLP*. The Llama 2 model analyzes the text and identifies key parameters: database type (PostgreSQL), server count (2) and replication mode. An example of system logging at this stage is provided below (listing 2).

Listing 2. *Example of system logging during the NLP processing stage.*

```
[INFO] [2025-02-20 22:00:03] Llama 2 model started text analysis.
[DEBUG] [2025-02-20 22:00:31] Identified key parameters:
- Database type: PostgreSQL
- Number of servers: 2
- Replication mode: master-slave
[INFO] [2025-02-20 22:00:35] Text analysis completed successfully.
```

3. *Configuration generation*. Based on the model's analysis, commands are generated to configure the PostgreSQL database (listing 3), including primary server setup, replication parameters and backup server creation.

Listing 3. *Example of a Llama 2 model-generated command for PostgreSQL database configuration.*

```
# Primary Server Configuration
sudo sed -i "s/#wal_level = replica/wal_level = replica/" /etc/postgresql/12/main/postgresql.conf
sudo sed -i "s/#max_wal_senders = 3/max_wal_senders = 3/" /etc/postgresql/12/main/postgresql.conf
sudo sed -i "s/#hot_standby = on/hot_standby = on/" /etc/postgresql/12/main/postgresql.conf
sudo sed -i "s/#hot_standby = on/hot_standby = on/" /etc/postgresql/12/main/postgresql.conf
sudo -u postgres psql -c "CREATE ROLE replicator WITH REPLICATION LOGIN PASSWORD
'*******';"
sudo systemctl restart postgresql

# Backup Server Configuration
sudo pg_basebackup -h primary_server -D /var/lib/postgresql/12/main -U replicator -P -v

echo "standby_mode = 'on'
primary_conninfo = 'host=primary_server port=5432 user=replicator password=******* '" | sudo tee /
var/lib/postgresql/12/main/recovery.conf

sudo chown -R postgres:postgres /var/lib/postgresql/12/main
sudo systemctl restart postgresql
```

An example of the automation system's console output at this stage is provided below (listing 4).

Listing 4. *Example of system logging during the configuration generation stage*.

```
[INFO] [2025-02-20 22:01:24] Configuration generation started.
[DEBUG] [2025-02-20 22:01:37] Generated commands for primary server (master) setup:
1. Configuration of `postgresql.conf`:
  - wal_level = replica
  - max_wal_senders = 3
  - hot_standby = on
2. Creation of replication user:
  - CREATE ROLE replicator WITH REPLICATION LOGIN PASSWORD *******;
[DEBUG] [2025-02-20 22:02:21] Generated commands for backup server (replica) setup:
1. Configuration of `recovery.conf`:
  - standby_mode = on
  - primary_conninfo = 'host=primary_server port=5432 user=replicator password=*******'
2. Initialization of the backup server:
  - pg_basebackup -h primary_server -D /var/lib/postgresql/12/main -U replicator -P -v
[INFO] [2025-02-20 22:03:07] Configuration generation completed successfully.
```

4. *Command execution*. The commands are executed on the target servers. The primary server is configured as a master and the backup server as a replica. An example of the automation system's console output at this stage is provided below (listing 5).

Listing 5. *Example of system logging during the command execution stage*.

```
[INFO] [2025-02-20 22:07:34] Command execution started on the target server (primary server).
[DEBUG] [2025-02-20 22:07:53] Commands executed on the primary server:
1. `postgresql.conf` parameters updated successfully.
2. User `replicator` created.
[INFO] [2025-02-20 22:08:06] Command execution started on the target server (backup server).
[DEBUG] [2025-02-20 22:08:21] Commands executed on the backup server:
1. `recovery.conf` parameters updated successfully.
2. Backup server initialized using `pg_basebackup`.
[INFO] [2025-02-20 22:08:44] Command execution completed successfully.
```

5. *Monitoring and feedback*. The system verifies the replication status and delivers a report to the user: “Replication configured successfully. Primary server: active. Backup server: synchronized.” An example of the automation system’s console output at this stage is provided below (listing 6).

Listing 6. *Example of system logging during the monitoring and feedback stage*.

```
[INFO] [2025-02-20 22:09:02] Replication status monitoring started.
[DEBUG] [2025-02-20 22:09:11] Replication status check:
- Primary server: active, replication enabled.
- Backup server: synchronized with the primary server.
[INFO] [2025-02-20 22:09:49] Monitoring completed successfully.
[INFO] [2025-02-20 22:09:51] Report provided to the user:
"Replication successfully configured. Primary server: active. Backup server: synchronized."
[INFO] [2025-02-20 22:09:52] Task "Set up PostgreSQL with replication" completed successfully.
```

Experimental results

The experiment aimed to compare the proposed approach with traditional configuration tools. Traditional automation tools include: integrated development environments (IDEs), web configurators, administrative panels for information systems, bash/batch scripts, CLI utilities, manual scripting, GUI tools and others [37].

Key task-specific metrics are selected for comparison, including development time, error correction time, error counts at various stages (compilation, code review, others) and overall solution quality and efficiency.

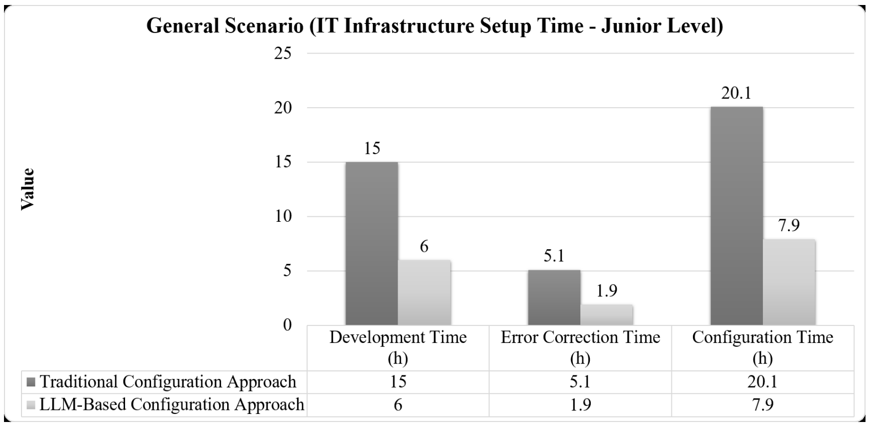
The experiment involved five distinct scenarios, each representing a standard infrastructure configuration task. These scenarios are chosen to cover core infrastructure components applicable to most IT projects:

1. PostgreSQL database configuration with primary and backup server replication.
2. Kubernetes setup with automatic scaling.
3. CI/CD pipeline configuration using GitLab and Jenkins.
4. Load balancer configuration across servers.
5. Monitoring system setup with Prometheus and Grafana metrics visualization [34].

System pre-configuration was performed on a hardware platform comprising an Intel Xeon E5-2666 v3 processor (2.90 GHz), 32 GB of RAM and an NVIDIA GeForce RTX 3050 Ti graphics processing unit (GPU) with 8 GB of dedicated memory. Preparing the baseline environment – including the installation and configuration of Java and Docker, subsequent deployment of the software implementing the proposed approach, installation of dependencies, runtime environment setup for the Llama 2 model, its initialization, and functional testing – required approximately 6–8 hours of a software engineer’s effort. Such configuration steps, essential for ensuring software compatibility and operational stability, are excluded from the experimental results, as they represent a one-time infrastructure preparation activity.

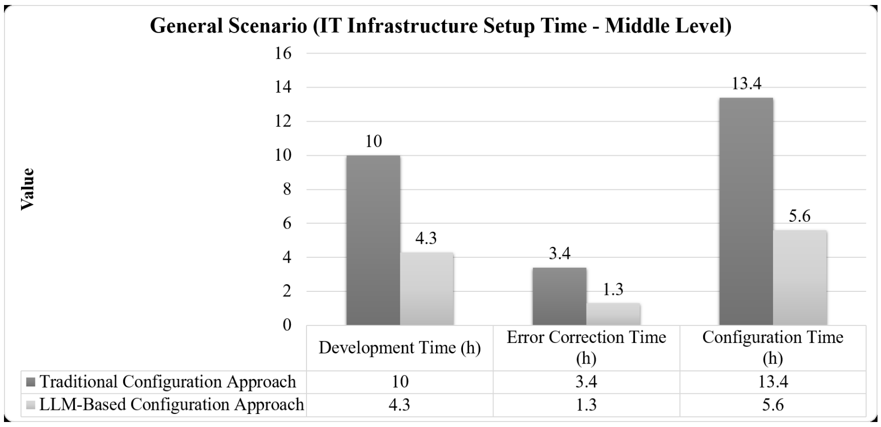
Each scenario is executed by three developers of varying expertise: junior, middle and senior engineers. This design allowed assessing how the method’s efficiency varies with developer experience. The experimental data included:

– average lines of code (LOC) per scenario;
– time spent on configuration using traditional tools;
– time spent on automated development and configuration;
– error counts during compilation, code review and other stages;
– solution quality and efficiency, measured as the ratio of LOC to error count.



	Development Time (h)	Error Correction Time (h)	Configuration Time (h)
■ Traditional Configuration Approach	15	5.1	20.1
■ LLM-Based Configuration Approach	6	1.9	7.9

Fig. 4. Comparison of configuration time
for the generalized IT infrastructure scenario by a junior level engineer



	Development Time (h)	Error Correction Time (h)	Configuration Time (h)
■ Traditional Configuration Approach	10	3.4	13.4
■ LLM-Based Configuration Approach	4.3	1.3	5.6

Fig. 5. Comparison of configuration time
for the generalized IT infrastructure scenario by a middle level engineer

Final results are aggregated into a generalized scenario combining data from all five cases, providing a comprehensive evaluation of the Llama 2 LLM effectiveness in infrastructure automation. The generalized scenario compared traditional and LLM-based approaches across all developer levels, highlighting overarching trends and automation advantages [6].

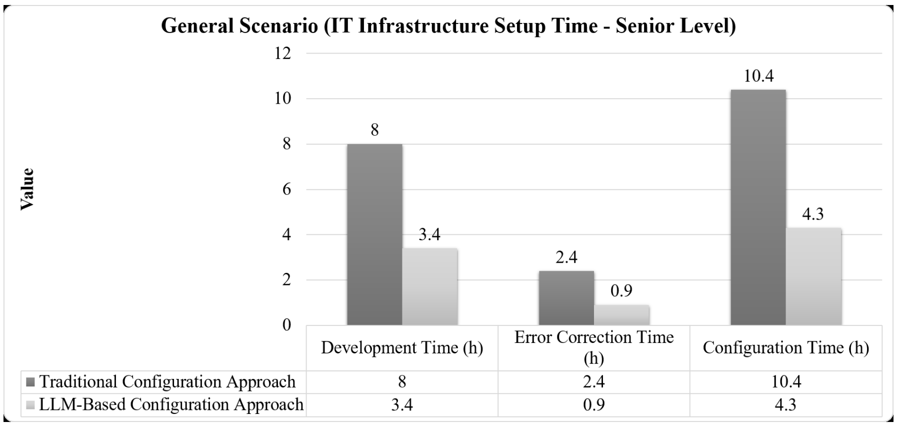
Analysis of development and configuration time

Research on language model scaling [38] and computational resource optimization [39] indicates that reduced computational costs and enhanced model performance indirectly shorten infrastructure development and configuration time. Our findings align with these insights.

For the generalized scenario, the traditional approach required 15 hours for junior developers, whereas the proposed method reduced this to 6 hours (Fig. 4). Similar trends are observed for middle level engineers (10 to 4.3 hours; Fig. 5) and senior engineers (8 to 3.4 hours; Fig. 6). These results demonstrate a 57–60% reduction in configuration time across all expertise levels.

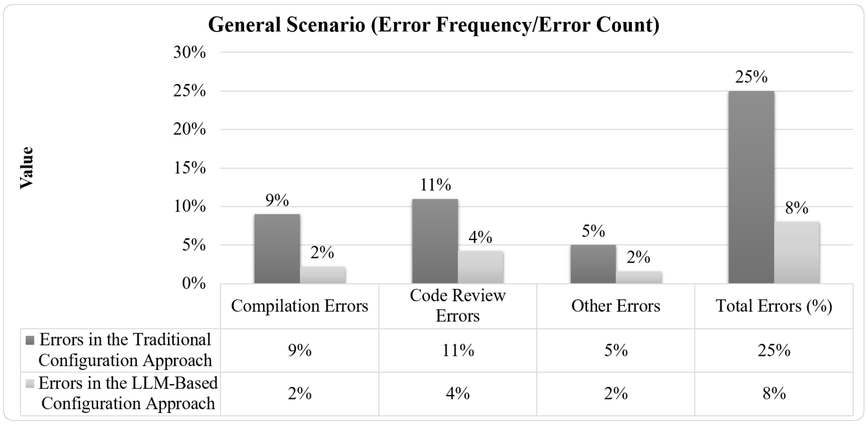
Error reduction

The LLM-based approach significantly reduced error rates across all development stages [40]. Under the traditional method, the total error rate for the generalized scenario was 25%, which dropped to 8% with the proposed method (Fig. 7). Errors decreased at every stage: compilation errors fall from 9%



	Development Time (h)	Error Correction Time (h)	Configuration Time (h)
■ Traditional Configuration Approach	8	2.4	10.4
■ LLM-Based Configuration Approach	3.4	0.9	4.3

Fig. 6. Comparison of configuration time
for the generalized IT infrastructure scenario by a senior level engineer



	Compilation Errors	Code Review Errors	Other Errors	Total Errors (%)
■ Errors in the Traditional Configuration Approach	9%	11%	5%	25%
■ Errors in the LLM-Based Configuration Approach	2%	4%	2%	8%

Fig. 7. Comparison of error rates in the generalized IT infrastructure configuration scenario

to 2% and code review errors from 11% to 4%. This confirms that LLM-driven automation enhances reliability by minimizing human error.

Solution quality and efficiency

For the generalized scenario, solution quality (measured as LOC to error ratio) improved by an average factor of 3.2 with the LLM-based method (Fig. 8). Calculation of the solution quality is made by the formula:

$$Q = \frac{A_{code}}{A_{error}},$$

where A_{code} is the LOC; A_{error} is the error ratio.

Solution efficiency, defined as the ratio of LOC to average configuration time, was 2.5 times higher compared to the traditional approach (Fig. 9). Calculation of the solution efficiency is made by the formula:

$$E = \frac{A_{code}}{T_{average}},$$

where A_{code} is the LOC; $T_{average}$ – average configuration time.

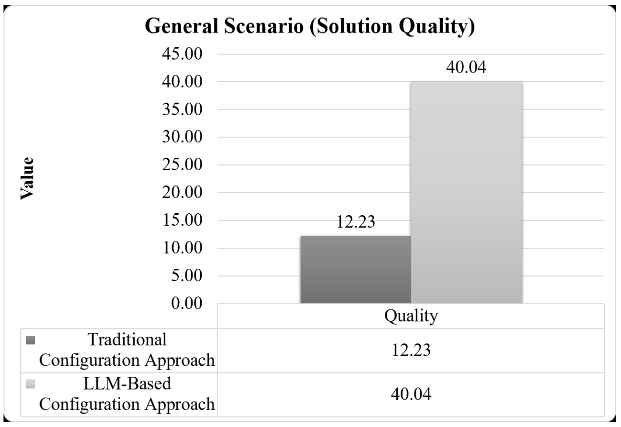


Fig. 8. Comparison of quality metrics for the generalized IT infrastructure scenario: traditional vs LLM-based approach

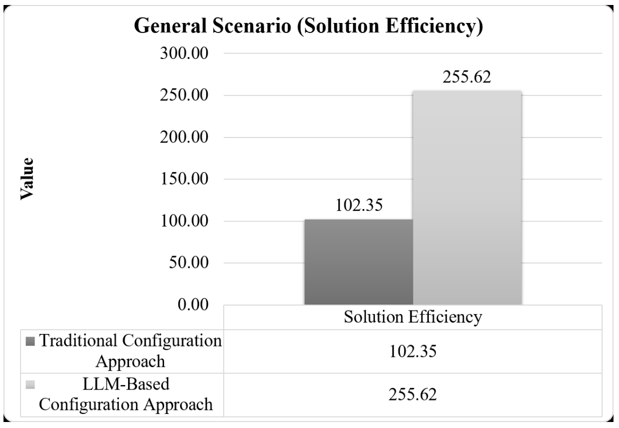


Fig. 9. Comparison of efficiency metrics for the generalized IT infrastructure scenario: traditional vs LLM-based approach

Thus, the results obtained for the generalized scenario demonstrate that the LLM not only reduces configuration time and lowers error rates, but also enhances overall solution quality and efficiency across all developer expertise levels.

General conclusions

The experiment demonstrates that the LLM-based approach reduces development time by up to 60%, lowers error rates from 25% to 8% and improves solution quality and efficiency by factors of 3.1 and 2.5, respectively. These results validate the hypothesis that LLMs like Llama 2 can effectively automate infrastructure configuration, particularly in complex, large-scale IT projects. The method offers substantial time and resource savings while enhancing solution reliability and quality compared to traditional approaches [41].

However, there are limitations for suggested approach usage. They are listed below.

1. *Effectiveness in low-complexity tasks*. In scenarios requiring the installation and configuration of a single component, traditional approaches utilizing pre-engineered scripts demonstrate superior efficiency. This is attributed to the overhead of time spent configuring AI agents when optimized pre-existing solutions tailored to specific infrastructural conditions are already available.

2. *Resource intensity of infrastructure*. To ensure acceptable query processing speeds, a cloud infrastructure supporting LLMs is required, including the allocation of GPU-accelerated instances. This

requirement introduces significant operational expenditures (OPEX) associated with leasing and maintaining computational resources.

3. *Uncertainty in outcomes for complex queries*. When generating configurations for high-level or multi-component tasks, non-deterministic outputs may arise, necessitating mandatory verification and manual changes (if required) by engineers. This limitation reduces system autonomy and increases overall deployment time.

Conclusion

This study proposes a method for automating project IT infrastructure configuration, leveraging the LLM Llama 2 to convert human-readable task descriptions in natural language into executable commands. The approach reduces the time required for IT infrastructure setup by up to 60% compared to traditional tool-based methods. A software architecture implementing the proposed method is developed in the form of an AI agent, which has demonstrated its practical efficiency.

Experimental results revealed that automation via the proposed method significantly reduces the error rate in generated software configurations. Specifically, the traditional manual configuration approach resulted in an error rate of 25%, whereas the proposed method reduced this figure to 8%. These findings highlight the substantial advantages of the proposed method over conventional tool-based configuration techniques.

Further experimental evaluations quantified the quality of automated configuration using Llama 2, demonstrating an improvement in 3 times on average compared to traditional tools. Additionally, the efficiency of the solution increased by a factor of 2.5.

Future research directions include integrating LLMs [20] with other automation tools such as configuration management systems (e.g., Ansible, Terraform) and container orchestration platforms (e.g. Kubernetes). It is also critical to explore the potential of fine-tuning the models for domain-specific tasks [42], which could enhance their accuracy and adaptability.

REFERENCES

1. **Ivlev V.A., Nikiforov I.V., Yusupova O.A.** Automation method for configuring IT infrastructure for IT projects. *International Conference on Digital Transformation: Informatics, Economics, and Education* (*DTIEE2023*), 2023, Vol. 12637, Pp. 67–73. DOI: 10.1117/12.2680779

2. **Ustinova V.E., Lutsenko A.S., Shpak A.V. et al.** A method for finding the correspondence between a railway station model and its visual representation based on graphs. *Computing, Telecommunications and Control*, 2024, Vol. 17, No. 4, Pp. 64–77. DOI: 10.18721/JCSTCS.17406

3. **Vijayakumar K., Arun C.** Automated risk identification using NLP in cloud based development environments. *Journal of Ambient Intelligence and Humanized Computing*, 2017, Pp. 1–13. DOI: 10.1007/s12652-017-0503-7

4. **Anil R. et al.** Palm 2 technical report. *arXiv:2305.10403*, 2023. DOI: 10.48550/arXiv.2305.10403

5. **Ivlev V.A., Mironenkov G.V., Nikiforov I.V., Ustinov S.M.** Ispol'zovanie modeli GPT-3 dlia generatsii informatsionno-tekhnologicheskoi infrastruktury proekta na osnove neformalizovannykh trebovanii [Using the GPT-3 model to generate a project's information technology infrastructure based on informal requirements]. *Sovremennye tekhnologii v teorii i praktike programmirovaniia* [*Modern technologies in the theory and practice of programming*], 2024, Pp. 174–176.

6. **Allamanis M., Barr E.T., Devanbu P., Sutton C.** A survey of machine learning for big code and naturalness. *arXiv:1709.06182*, 2017. DOI: 10.48550/arXiv.1709.06182

7. **Ouyang L., Wu J., Jiang X., Almeida D., Wainwright C.L., Mishkin P., Zhang C., Agarwal S., Slama K., Ray A., Schulman J., Hilton J., Kelton F., Miller L., Simens M., Askell A., Welinder P., Christiano P., Leike J.,**

Lowe R. Training language models to follow instructions with human feedback. *arXiv:2203.02155*, 2022. DOI: 10.48550/arXiv.2203.02155

8. **Lewis M., Liu Y., Goyal N., Ghazvininejad M., Mohamed A., Levy O., Stoyanov V., Zettlemoyer L.** BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv:1910.13461*, 2019. DOI: 10.48550/arXiv.1910.13461

9. **Srivastava A. et al.** Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv:2206.04615*, 2022. DOI: 10.48550/arXiv.2206.04615

10. **Ziegler D.M., Stiennon N., Wu J., Brown T.B., Radford A., Amodei D., Christiano P., Irving G.** Fine-tuning language models from human preferences. *arXiv:1909.08593*, 2019. DOI: 10.48550/arXiv.1909.08593

11. **Chen M. et al.** Evaluating large language models trained on code. *arXiv:2107.03374*, 2021. DOI: 10.48550/arXiv:2107.03374

12. **Feng Z., Guo D., Tang D., Duan N., Feng X., Gong M., Shou L., Qin B., Liu T., Jiang D., Zhou M.** CodeBERT: A pre-trained model for programming and natural languages. *arXiv:2002.08155*, 2020. DOI: 10.48550/arXiv:2002.08155

13. **Black S., Biderman S., Hallahan E., Anthony Q., Gao L., Golding L., He H., Leahy C., McDonell K., Phang J., Pieler M., Sai Prashanth USVSN, Purohit S., Reynolds L., Tow J., Wang B., Weinbach S.** GPT-NeoX-20B: An open-source autoregressive language model. *arXiv:2204.06745*, 2022. DOI: 10.48550/arXiv:2204.06745

14. **Ivlev V.A., Mironenkov G.V., Nikiforov I.V.** Sozdanie infrastruktury proekta s pomoshch'iu neironnoi seti [Creating a project infrastructure using a neural network]. *Nedelia nauki IKNK* [*Institute of Computer Science and Cybersecurity Science Week*], 2024. Pp. 24–26.

15. **Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M., Zhou Y., Li W., Liu P.J.** Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv:1910.10683*, 2019. DOI: 10.48550/arXiv:1910.10683

16. **Schick T., Dwivedi-Yu J., Dessì R., Raileanu R., Lomeli M., Zettlemoyer L., Cancedda N., Scialom T.** Toolformer: Language models can teach themselves to use tools. *arXiv:2302.04761*, 2023. DOI: 10.48550/arXiv:2302.04761

17. **Gururangan S., Marasović A., Swayamdipta S., Lo K., Beltagy I., Downey D., Smith N.A.** Don't stop pretraining: Adapt language models to domains and tasks. *arXiv:2004.10964*, 2020. DOI: 10.48550/arXiv:2004.10964

18. **Wolf T., Debut L., Sanh V., Chaumond J., Delangue C., Moi A., Cistac P., Rault T., Louf R., Funtowicz M., Davison J., Shleifer S., von Platen P., Ma C., Jernite Y., Plu J., Xu C., Le Scao T., Gugger S., Drame M., Lhoest Q., Rush A.** Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, Pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6

19. **Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I.** Attention is all you need. *arXiv:1706.03762*, 2017. DOI: 10.48550/arXiv.1706.03762

20. **Chowdhery A. et al.** PaLM: Scaling language modeling with pathways. *arXiv:2204.02311*, 2022. DOI: 10.48550/arXiv.2204.02311

21. **Touvron H. et al.** Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*, 2023. DOI: 10.48550/arXiv:2307.09288

22. **Brown T.B. et al.** Language models are few-shot learners. *arXiv:2005.14165*, 2020. DOI: 10.48550/arXiv:2005.14165

23. **Gao L., Biderman S., Black S., Golding L., Hoppe T., Foster C., Phang J., He H., Thite A., Nabeshima N., Presser S., Leahy C.** The Pile: An 800GB dataset of diverse text for language modeling. *arXiv:2101.00027*, 2020. DOI: 10.48550/arXiv:2101.00027

24. **Le Scao T. et al.** BLOOM: A 176B-parameter open-access multilingual language model *arXiv:2211.05100*, 2022. DOI: 10.48550/arXiv:2211.05100

25. **Wei J., Wang X., Schuurmans D., Bosma M., Ichter B., Xia F., Chi E., Le Q., Zhou D.** Chain-of-thought prompting elicits reasoning in large language models. *arXiv:2201.11903*, 2022. DOI: 10.48550/arXiv:2201.11903

26. **Black S., Leo G., Wang P., Leahy C., Biderman S.** GPT-neo: Large scale autoregressive language modeling with mesh-tensorflow (1.0). *Zenodo*, 2021. DOI: 10.5281/zenodo.5297715.

27. **Bengio Y., Louradour J., Collobert R., Weston J.** Curriculum learning. *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, Pp. 41–48.

28. **Rozière B. et al.** Code Llama: Open foundation models for code. *arXiv:2308.12950*, 2023. DOI: 10.48550/arXiv:2308.12950

29. **Rajani N.F., McCann B., Xiong C., Socher R.** Explain yourself! Leveraging language models for commonsense reasoning. *arXiv:1906.02361*, 2019. DOI: 10.48550/arXiv:1906.02361

30. **Armitage J., Kacupaj E., Tahmasebzadeh G., Swati, Maleshkova M., Ewerth R., Lehmann J.** MLM: A benchmark dataset for multitask learning with multiple languages and modalities. *arXiv:2008.06376*, 2020. DOI: 10.48550/arXiv:2008.06376

31. **Devlin J., Chang M.-W., Lee K., Toutanova K.** BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018. DOI: 10.48550/arXiv:1810.04805

32. **Ivlev V.A., Mironenkov G.V., Nikiforov I.V., Kovalev A.D.** Generatsiia informatsionno-tekhnologicheskoi infrastruktury proekta na osnove neformalizovannykh trebovanii [Generation of the project's information technology infrastructure based on informal requirements]. *Sovremennye tekhnologii v teorii i praktike programmirovaniia* [*Modern technologies in the theory and practice of programming*], 2023, Pp. 242–244.

33. **Bommasani R. et al.** On the opportunities and risks of foundation models. *arXiv:2108.07258*, 2021. DOI: 10.48550/arXiv:2108.07258

34. **Nikiforov I.V., IUsupova O.A., Voinov N.V., Kovalev A.D., Tkachuk A.S., Varlamov D.A., Geras'kin E.V.** *Programmnye instrumenty obrabotki i vizualizatsii dannykh. Elasticsearch, Logstash, Kibana, Grafana, Prometheus* [*Software tools for data processing and visualization. Elasticsearch, Logstash, Kibana, Grafana, Prometheus*]. St. Petersburg: POLITEKH-PRESS, 2023. DOI: 10.18720/SPBPU/2/id23-74

35. **Fried D., Aghajanyan A., Lin J., Wang S., Wallace E., Shi F., Zhong R., Yih W.-t., Zettlemoyer L., Lewis M.** InCoder: A generative model for code infilling and synthesis. *arXiv:2204.05999*, 2022. DOI: 10.48550/arXiv:2204.05999

36. **Nijkamp E., Pang B., Hayashi H., Tu L., Wang H., Zhou Y., Savarese S., Xiong C.** CodeGen: An open large language model for code with multi-turn program synthesis. *arXiv:2203.13474*, 2022. DOI: 10.48550/arXiv:2203.13474

37. **Sajja P.S.** Computer-assisted tools for software development. In: *Essence of Systems Analysis and Design: A Workbook Approach*, 2017, Pp. 93–105. DOI: 10.1007/978-981-10-5128-9_5

38. **Kaplan J., McCandlish S., Henighan T., Brown T.B., Chess B., Child R., Gray S., Radford A., Wu J., Amodei D.** Scaling laws for neural language models. *arXiv:2001.08361*, 2020. DOI: 10.48550/arXiv:2001.08361

39. **Hoffmann J. et al.** Training compute-optimal large language models. *arXiv:2203.15556*, 2022. DOI: 10.48550/arXiv:2203.15556

40. **Hindle A., Barr E., Gabel M., Su Z., Devanbu P.** On the naturalness of software. *Communications of the ACM*, 2016, Vol. 59, No. 5, Pp. 122–131.

41. **Lu S. et al.** CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. *arXiv:2102.04664*, 2021. DOI: 10.48550/arXiv:2102.04664

42. **Bahdanau D., Cho K., Bengio Y.** Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014. DOI: 10.48550/arXiv:1409.0473

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Ivlev Vladislav A.
Ивлев Владислав Александрович
E-mail: nevidd@yandex.ru

Nikiforov Igor V.
Никифоров Игорь Валерьевич
E-mail: igor.nikiforovv@gmail.com
ORCID: https://orcid.org/0000-0003-0198-1886

Ustinov Sergey M.
Устинов Сергей Михайлович
E-mail: usm50@yandex.ru
ORCID: https://orcid.org/0000-0003-4088-4798

Submitted: 13.03.2025; Approved: 13.05.2025; Accepted: 26.05.2025.
Поступила: 13.03.2025; Одобрена: 13.05.2025; Принята: 26.05.2025.