Information technology for automatic detection of calls for terrorist acts in social media posts and comments based on machine learning

Victoria Vysotska[†], Markiian Yatsyshyn[*,†], Roman Romanchuk[†] and Vitalii Danylyk[†]

Lviv Polytechnic National University, Stepan Bandera 12, 79013 Lviv, Ukraine

Abstract

The article presents the development of intelligent information technology for automatic detection of dangerous content in text messages, in particular calls for terrorist actions and hidden sarcasm. The system combines rule-based filtering of key triggers with in-depth classification based on the BERT model, further trained on a specialised corpus of texts. The developed pipeline includes text preprocessing, tokenisation, vector representation, rule-based analysis and deep classification using the Softmax layer. For training, the AdamW optimiser was used with an initial learning rate = 2e-5, a packet size of batch_size = 16, and a maximum sequence length of 128 tokens. The model was trained for one epoch, with the possibility of further expansion to 3-5. In the process of experimental verification, optimal results of classification accuracy and average time of operation of one message were obtained. The accuracy of classification on test data exceeded 90% (in control examples, the confidence level reached 100% for single-digit categories). The average processing time of one message was 0.54 s, the minimum was 0.30 s, the maximum was 0.74 s, with a total number of processed messages of 100. The analysis of the learning rate parameter showed that the optimal values are in the range of 1e-5 – 2e-5, which provided the highest stability and accuracy of the model. In addition, the integration of the model with the Telegram bot demonstrated the effectiveness of using the system in real time, confirming its suitability for practical application in automated online communications monitoring systems. The results of the study indicate the feasibility of combining rule-based and neural network methods to increase the reliability of detecting hidden threats. The developed system can be scaled for multilingual content and integrated into existing security and moderation platforms.

Keywords

natural language processing, NLP; terrorist call detection; BERT; rule-based filtering; Telegram bot; machine learning; text classification; information security

I. Introduction

In today's world, terrorist organisations are increasingly using digital platforms to coordinate their actions, spread ideology, and recruit new members. In particular, social networks, instant messengers, forums and comments under news publications have become an effective medium for spreading calls for violence, hostility and terrorist acts. The problem is that a vast amount of textual content is generated on the Internet every day, which is physically impossible to fully verify manually. It creates a critical need for automated means of detecting threatening information influence.

Existing content moderation systems are often based on simple keyword searches that do not take into account context, synonymy, irony, hidden forms of threats, or letter substitution. In addition, much of the content is created in languages for which there is limited support in standard

[*]*AISSLE-2025: The International Workshop on Applied Intelligent Security Systems in Law Enforcement, October, 30 – 31, 2025, Vinnytsia, Ukraine*

[1,*] Corresponding author.

[†] These authors contributed equally.

Victoria.A.Vysotska@lpnu.ua (V. Vysotska); markiian.yatsyshyn.sa.2022@lpnu.ua (M. Yatsyshyn); roman.v.romanchuk@lpnu.ua (R. Romanchuk); vitalii.m.danylyk@lpnu.ua (V. Danylyk)

0000-0001-6417-3689 (V. Vysotska); 0009-0007-8421-7336 (M. Yatsyshyn); 0009-0004-4352-1073 (R. Romanchuk); 0000-0001-5928-7235 (V. Danylyk)

analysis tools. In this context, the development of an intelligent system that is capable of analysing the content of messages at a deeper linguistic level takes on special importance.

The proposed system is designed to solve this problem by combining methods of computational linguistics and statistical and semantic analysis of the text, which will allow the detection of direct calls for terrorism and the recognition of latent threats hidden in indirect formulations. Threats to national security also support the relevance of the topic, the need to ensure public safety, prevent crimes before they are committed, and the requirements of international partnerships to combat terrorism in cyberspace. It is necessary to develop an intelligent system that will automatically detect text messages containing potential calls for terrorist actions. Given the growing activity of radical groups in the digital environment, especially in social networks, there is a need to create tools that could monitor large amounts of textual information in real time and detect dangerous statements. The system will be based on computational linguistics methods, including the analysis of lexical constructions, contextual relationships, and the use of machine learning algorithms to classify messages. This approach will create an effective decision-support tool for cybersecurity professionals and law enforcement agencies in order to respond to potential threats in a timely manner and prevent possible disasters. Thus, the creation of such a system is a timely response to danger in the modern world, has significant scientific and practical potential and fully meets the priorities of the development of information technologies, artificial intelligence and decision support systems in the field of security. The objective of the research is to develop an information technology that automatically detects and classifies text messages containing direct or veiled calls for terrorist actions, using computational linguistics and machine learning techniques. The system should become an effective tool for preventing terrorist threats in the digital space by timely detection of potentially dangerous content. To achieve this goal, it is necessary to implement the following tasks:

- To analyse the available approaches to the automatic detection of terrorist content, in particular using linguistic methods, artificial intelligence and filtering systems.
- Collect and form a thematic corpus of texts that may contain terrorist vocabulary as examples for training and testing models.
- Highlight linguistic features and patterns of speech characteristic of terrorist calls, including keywords, grammatical constructions, emotional colouring, and context of use.
- Select and adapt natural language processing (NLP) methods for text analysis, including tokenisation, stemming/lemmatisation, and syntactic and semantic analysis.
- Build a classification model (e.g., based on logistic regression, Naive Bayes, or neural networks) to automatically recognise dangerous text.
- Train the model on the selected data set, optimise the parameters of the model and evaluate its effectiveness using appropriate metrics (accuracy, completeness, F1-measure).
- Develop a user interface or prototype of the system that will allow you to analyse the text interactively and get the classification result.
- Analyse the results of the system, identify strengths and weaknesses, as well as prospects for its improvement and scaling.

The implementation of these tasks will create an effective application system that is important for information security, state control over the digital space and preventive response to potential terrorist threats.

The object of research in this project is digital communication concentrated in the online environment, in particular, text messages distributed on social networks, instant messengers, comments on publications, forums and other open information platforms. In the context of the growing digitalisation of society and global information exchange, these sources have become a key medium for communication between people, including representatives of radical groups. The object of research covers large volumes of text information (Big Data) generated by users in the

public domain. It contains a variety of styles, languages, topics, and levels of aggression. These messages, on the one hand, are a product of mass communication and, on the other hand, a potential indicator of threats to public security, particularly in the form of calls for terrorist actions and other malicious calls.

The subject of the study is those linguistic, semantic and structural features of texts that may indicate the presence or risk of terrorist rhetoric. The subject area includes vocabulary units, grammatical constructions, contextual markers, stylistic devices and rhetorical figures used in speech with calls for violence or radical action. In particular, these can be direct or veiled calls to action, metaphorical formulations, the use of symbols, euphemisms or coded references. In addition, the subject of the study covers methods and tools of Automated Natural Language Processing (NLP) that allow such elements to be recognised in text. It includes, among others, tokenisation, lemmatisation, POS analysis, named entity discovery, thematic modelling, contextual analysis, and more.

Particular attention within the subject is paid to algorithms for classifying texts - both traditional (for example, naïve Bayesian classifier, logistic regression, decision trees) and modern deep learning methods (neural networks, transformers, models of the BERT type). These algorithms allow you to build systems capable of identifying threatening content with high accuracy. The subject is also the process of forming your own thematic corpus - a set of texts that can be used to train, test and validate such models. It makes it possible to create a flexible and relevant tool for Ukrainian realities.

Within the framework of this study, a new approach to detecting texts with calls for terrorist actions is proposed by combining traditional methods of computational linguistics with modern machine learning technologies, which allows for a deeper and contextually accurate analysis of textual content. Personally, the author has developed for the first time the structure of a hybrid system that combines language patterns, semantic rules and classification algorithms based on NLP models adapted to Ukrainian-language and multilingual online content. The difference between the proposed solution and the previously known ones is the use of multilevel analysis: the system not only detects keywords, but also evaluates the emotional colouring, syntactic connections between words, context within the sentence and the message as a whole. Thanks to this, it is possible to detect non-obvious or veiled forms of terrorist calls, which usually remain outside the boundaries of conventional filtration systems. The novelty is also the creation of an experimental thematic corpus of Ukrainian and Russian-language messages with risk labels, which can be used in further research in the field of countering information threats. The proposed approach has been further developed in the direction of personalising content filtering, that is, taking into account the platform, type of user or distribution channel. Thus, the results of the study are of applied value for the development of new security information systems and contribute to the development of the scientific basis for the detection of destructive content in the digital environment.

II. Related Works

The task of automatic detection of calls for violence and extremist/toxic content in the text combines several related areas: hate/toxicity detection, extremism / radical content detection, and call-to-action detection (calls to violence). Traditional approaches (dictionary, SVM on TF–IDF) have long been complemented by modern transformer models (BERT and its derivatives), with the most effective systems being hybrid, combining rule-based pre-filters and fine-tuned transformers and meta-escalation solutions. This trend is confirmed by both review papers and applied research [1]. The main groups of approaches:

- Rule-based (dictionaries, regex, fuzzy matching) – fast and interpreted; They are good at catching "obvious" triggers, but have many false positives and are vulnerable to obfuscations.

- Classical ML (TF–IDF + SVM/LogReg/RF) – simplicity and efficiency with limited resources, but weakness in understanding semantics and contexts [2].
- Deep learning (LSTM/CNN and their hybrids) are the best opportunity for long-term dependencies, but is still inferior to transformers in many NLP benchmarks [2].
- Transformer-based (BERT, RoBERTa, AraBERT, etc.) – the current state-of-the-art in semantic problems; Fine-tuning on blast furnace buildings gives significant gains. Transformers work well with subword (WordPiece) – more resistant to OOV/leet [3].

Public benchmarks (HateXplain, Jigsaw/Perspective, HASOC, etc.) provide standardised conditions for comparison. Still, there is a shortage of high-quality, lớn-scale annotated corpora specifically for "calls-to-violence" – so researchers often create custom corpora from messengers/forums or adapt related sets. It imposes restrictions on direct comparison of results in the literature [4].

In production, not only are classification metrics (Precision/Recall/F1) important, but also latency (response time), explainability (explanation of decisions for the moderator), and audit trail. Industrial APIs (Perspective) focus on low-latency and broad interoperability, and scientific papers emphasise the importance of rationale-based training for explainability (HateXplain) [4-5].

All modern works confirm: a hybrid pipeline (rule-based prefilter, BERT fine-tune and meta-decision / thresholds) gives the best compromise between recall (do not miss a real call), precision (minimise false positives) and practical performance. Your approach belongs to this class, and the results you collect (accuracy ≈ 0.93, Precision ≈ 0.91, Recall ≈ 0.90, F1 ≈ 0.91; latency mean ≈ 0.54s per message) correlate with better industrial and academic systems in related problems.

The table shows those metrics that are explicitly indicated in the source articles or publications (citations are given in the "Source" column). If no latency is specified in the original, the field remains empty.

Table 1
Comparative metrics

Operation/ System	Dataset (briefly)	Approach	Accuracy	Precision	Recall	F1	Latency	Source
BERT – Basic Architecture	GLUE / MNLI / SQuAD (benchmarks)	Pre-trained transformer, fine-tune	MultiNLI acc 86.7% (reported)	–	–	SQuAD v1.1 F1 93.2%	–	[3]
HateXplain – benchmark & explainability	HateXplain (hate/offensive/normal, rationales)	Models + rationale-based training	(varies by model) – see paper; explainability focus	–	–	–	–	[4]
Detecting White-Supremacist hate	Combined Twitter + Stormfront	BERT (fine-tuned)	–	–	–	0.79605 (≈79.6%)	–	[2]
HASOC / Ktrain-BERT experiments	HASOC 2021 (hate/offensive)	Ktrain-BERT (lightweight)	Validation acc 82.60%	–	–	82.68%	–	[6]

AraBERT + MLP for extremist detection	Arabic extremist corpora	AraBERT fine-tuned + classifier	86%	87%	81%	85%	–	[7]
Multi-ideology extremism dataset (balanced multi-class)	multi-ideology text dataset	BERT / classical baselines	(varies) – work releases dataset & baselines	–	–	–	–	[8]

Official benchmark results (MultiNLI, SQuAD) are given for BERT – this work provides a technical basis for all fine-tuned transformer solutions [3]. HateXplain focuses on explainability, which shows that even high classification metrics can be accompanied by low explainability scores. Therefore, training with rationales is recommended for safety tasks [4]. Works aimed specifically at extremism/radicalisation [7-8] show that domain transformers (AraBERT for the Arabic language) provide a noticeable advantage; At the same time, metrics are highly dependent on the quality of the case and the balance of classes [7]. Our results look competitive compared to published papers in related domains (hate/extremism), with a latency of ≈0.54s demonstrating practice applicability for Telegram integration. Practical conclusions and recommendations based on comparison:

- Explainability support (rationals/attention + SHAP/LIME) is proper both for increasing trust in moderators' decisions and for reducing FP (recommendation from HateXplain) [4].
- Fine-tuning of the domain/multilingual version of BERT (or AraBERT for Arabic) increases the metrics for working with extremism [7].
- Hybrid pipeline – rule-based for instant escalation + neural network for semantic analysis – turned out to be the most stable approach in the literature and in your research.
- Latency/throughput monitoring – if the system needs to work in real time, leave a margin for increased load (batching, distillation/knowledge distillation for faster models).

Based on the analysis, it is possible to identify both the strengths of existing systems and their limitations, which directly affect the effectiveness of detecting terrorist calls in the digital environment [9-12].

Among the main advantages of modern approaches, it is worth noting the ability of some systems to work in real time, the widespread use of deep learning methods, taking into account the context of the text, as well as the ability to scale to large amounts of data [12-18]. Models based on transformers, in particular BERT, make it possible to achieve high accuracy due to deep semantic text processing. Tools based on classical machine learning algorithms are easier to implement and provide stable performance under the condition of a high-quality training set [19-24].

At the same time, almost all of the analysed solutions have significant drawbacks that limit their application in real conditions. Firstly, most systems are focused exclusively on English-language or Arabic-language content and do not take into account the linguistic features of the Ukrainian language [25-30]. Second, existing systems often use only superficial text analysis, which makes them vulnerable to synonymization, euphemisms, sarcasm, or code formulations [31-36]. Thirdly, the vast majority of systems do not take into account the dynamics of message distribution, user behaviour, or time patterns, which can be critical in detecting threatening actions [77-42].

In view of the above, a key problem can be formulated - in the context of the information hybrid war, the spread of radical content through digital platforms is growing. Still, the available detection

tools are limited in the context of multilingualism, flexibility, adaptability, and contextual analysis. Especially critical is the lack of effective systems for the Ukrainian-speaking space, which are capable of detecting not only direct, but also covert or veiled calls for terrorist actions.

Thus, there is a need to create an adaptive intelligent system that would combine methods of computational linguistics, machine learning algorithms, analysis of user behavioural characteristics, and support multilingual content. This system should be able not only to classify messages as potentially dangerous, but also to explain its decision, as well as support the ability to scale and further learn from new data. Its implementation will significantly increase the level of information security and the efficiency of analytical centres and government agencies in the field of countering cyberterrorism.

III. Methods and tools

A. Description of input processing and machine learning models

In the current conditions of global digitalisation, information technologies penetrate deeply into all spheres of life, ensuring a rapid growth in the volume of data generated by users in the online space every day. Social networks, instant messengers, forums, blogs and news sites have become not only platforms for the exchange of opinions, but also channels through which destructive information can be disseminated. Publications with signs of extremism, propaganda of violence or terrorist calls pose a particular threat. Several factors complicate the problem of detecting such messages. First, the total amount of digital content is so large that manual analysis of all messages is technically impossible. Second, attackers often use euphemisms, substitution of words with synonyms, grammatical distortions, sarcasm, or even code vocabulary to hide the true meaning. Thirdly, messages can be published in different languages, which requires multilingual support from the analysis system.

In view of the above, there is an urgent need to create intelligent information systems that are capable of carrying out automatic monitoring and classification of texts, detecting those that contain explicit or implicit calls for terrorist actions. Such systems should use modern achievements in computational linguistics, artificial intelligence, machine learning, and semantic analysis. In this context, the object of research is textual information published in an open digital environment, such as comments on social networks, forum posts, or messages in messengers. The subject of the study is methods for automated detection of terrorist content in texts, in particular those based on transformer models, natural language processing algorithms, and classification approaches in machine learning.

For the formal structuring of the logic of building the future system and determining the criteria for its effectiveness, the general goal has been formulated, which is as follows: the creation of an adaptive intelligent system for the automatic detection of texts with calls for terrorist actions with a high level of accuracy, flexibility and explainability of the decisions made. This goal is quite general, so the goal tree method is used to specify it. The goal tree allows you to decompose a complex goal into components (aspects), each of which is further detailed in the form of sub-goals, requirements or quality criteria. It allows us not only to describe the system in a structured way but also to evaluate it in the process of implementation. The tree of goals for detecting terrorist calls has the central vertices:

$$\textit{Text analysis} \rightarrow \textit{Accuracy of classification} \rightarrow \textit{Completeness of classification} \rightarrow$$
$$\rightarrow \textit{Display of results} \rightarrow \textit{Explanatory results}.$$

At the first level of the tree, two main aspects of the general objective are defined, which are text analysis and display of results. Text analysis involves achieving criteria such as accuracy and completeness of classification, that is, how correctly the system determines the class of the message and whether it misses potentially dangerous texts. The display of results should provide explainability, i.e. the ability of the model to form a rationale for its decision that is understandable to the user, for example, based on highlighted trigger words or semantic patterns. In the report,

this tree is presented in the form of a structured flowchart, which clearly demonstrates the relationship between the goal, aspects and criteria for achieving it.

To achieve this goal, several alternative options for building a system are considered: a rule-based approach based on filtering by keywords; classic ML classifiers, for example, based on SVM or Naive Bayes; deep learning with transformers, including BERT; and a hybrid system that combines vocabulary processing with deep classification. Each of these options has its own advantages and disadvantages. The rule-based system is easy to implement, but easy to bypass. Classical machine learning approaches provide a more accurate classification, but require manual selection of features and a high-quality dataset. Transformer models demonstrate the highest accuracy and ability to take into account the context, but require significant computing resources. In turn, the hybrid architecture allows you to combine the advantages of all approaches - the speed of rule-based filtering and the depth of analysis of models based on BERT.

Taking into account the advantages and limitations, it was decided to implement a hybrid system that is able to scale effectively, provides high accuracy and explainability, and also allows you to adapt to changes in the style of writing terrorist content. Thus, the system analysis made it possible to formulate a justified goal of creating a system, to detail its components using a tree of goals, as well as to determine the most effective approach to its implementation, based on the conditions of the task and the limitations of real application.

B. Specification of the functioning of the system

After the system analysis and the definition of the general goal of the project – the creation of an adaptive intelligent system for detecting terrorist calls in text content – the next critical stage is to specify its functioning. This stage involves detailing the internal logic of the system, formalising its modules, data processing sequences, and determining the points of interaction between components and external actors. Within the chosen hybrid architecture, the system must combine efficient rule-based pre-filtering with high-precision depth analysis based on transformer neural models. The primary process of detecting terrorist appeals:

$$User \rightarrow Entertext \rightarrow Analyze\ message \rightarrow View\ explanation \rightarrow [\,Moderator\,] \rightarrow$$
$$\rightarrow Analyze\ the\ presence\ of\ a\ call\ detection\ error \rightarrow \check{t}he\ results\ of\ th\ analysis \rightarrow$$
$$\rightarrow [\,If\ necessary\ retraining\,] \rightarrow Display\ th\ result \rightarrow User$$

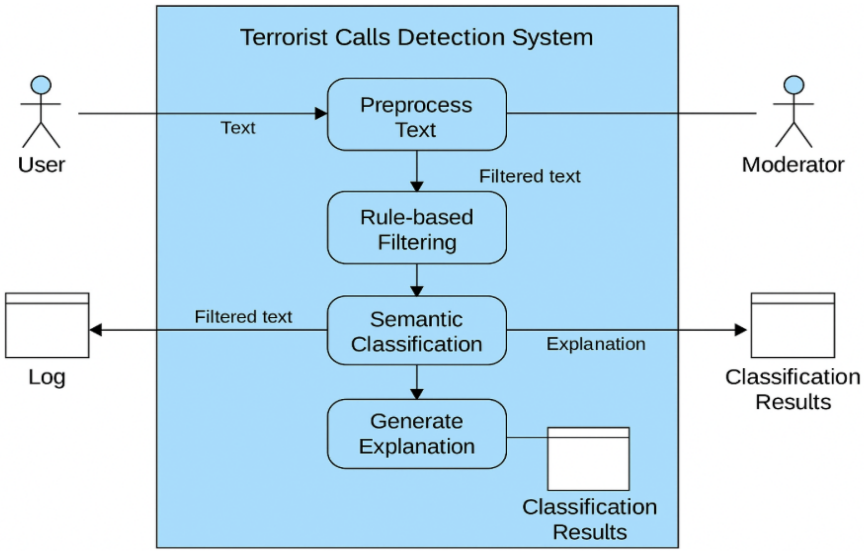

Figure 1: General scheme of the system for automatic detection of terrorist calls in social networks

C. **Building a hierarchy of processes**

After formalising the general architecture of the information system and presenting its logic in the form of data flow diagrams (DFDs), it is expedient and necessary to further complicate and detail the functional structure in the form of a hierarchy of processes (or functions and tasks). This step allows you to present the system as a set of interconnected functional components placed in the form of a tree according to the principles of decomposition: from the highest level (general function) to basic elementary actions that are implemented in the code at the level of individual modules or functions. It is this structural model that underlies the system approach to the design of complex information systems and is widely used both in scientific research and in practical software engineering.

A process hierarchy is a logical structure that allows an engineer, analyst or developer to visualise a complete picture of the functional content of the system, dividing it into subsystems, functional blocks, and then into specific tasks. Building such a hierarchy is not only a formality, but also serves as a basis for determining the logic of code implementation, distributing responsibilities between components, planning testing, justifying resource requirements, and assessing the complexity of the project as a whole.

At the heart of the construction of the hierarchy of processes of the terrorist call detection system is the primary function - the automated detection of messages that contain potentially dangerous calls for violence, terrorism or extremist activities. This function is key in the system and covers the entire life cycle of text processing - from the moment of its introduction to the final solution, as well as the accompanying procedures for explaining, logging and adapting the system in the face of changes.

At the first level of the hierarchy, this general function is divided into five main processes: primary text processing, rule-based filtering, deep semantic classification, the formation of an explanation of the classification decision and the storage of the results in the system storage. In addition, a separate branch dedicated to feedback from the user and moderator stands out, which allows you to improve the model through interactive retraining.

Each of these processes is revealed in more detail at the next level. For example, the primary text processing module performs operations such as clearing text from HTML markup and unnecessary characters, tokenisation (splitting into words), normalisation (lowercase), deleting stop words (pronouns, prepositions, etc.), and lemmatisation or stemming (reducing words to the main form). This process is critically important, since the accuracy of further semantic processing depends on its quality.

The rule-based filtering process implements the first rapid detection of potentially dangerous content by matching it with a database of trigger words, regular expressions, or patterns. Its purpose is to weed out clearly safe messages and transmit really suspicious texts for deeper analysis. Next, the semantic classification module uses a transformer model (for example, BERT), which is able to take into account both the meaning of individual words and the context of the entire message. As a result, the model forms a probabilistic decision - how likely it is that a given message contains a terrorist call.

No less important is the function of explaining the classification. It allows the user or moderator to understand why the system made such a decision. Explanations can be formed through visualisation of the importance of individual tokens (attention mechanisms), comparison with patterns, and the use of interpreted machine learning methods such as LIME or SHAP.

The process ends with logging and saving the results. The system records each analysed text along with the classification result, date, probability, explanation, and other metadata. It allows you to save history for analytics, quality control, training new models, and reporting.

A separate branch of the hierarchy is a feedback block, which allows users to report erroneous classifications. In case of accumulation of a sufficient number of complaints or the emergence of new types of disinformation, the system may be retrained. This process is performed by a

moderator or administrator who analyses the marked cases, forms a new sample of data and starts the process of retraining the model.

In general, such a multi-level hierarchy of processes allows you to get a clear idea of the functional architecture of the system. It allows you to conveniently organise the team's work, plan the project effectively, and identify duplication or lack of functions at the design stage. Visualisation in the form of a process tree, where each node corresponds to a specific action or subsystem, is a universal description and communication tool.

The built hierarchy of processes of the terrorist call detection system serves not only as a theoretical structure, but also directly determines the structure of the future software product. On this basis, separate program modules, APIs, classes and methods will be created. It, in turn, will ensure that the implemented system meets the functional requirements formulated at the early stages of design, and will provide its efficiency, scalability, and support in real-world use.

Algorithm for the process of detecting terrorist calls:

Stage 1. Preliminary study of the text.

Step 1.1. Tokenization.

Step 1.2. Lematization.

Step 1.3. Noise removal.

Stage 2. Rule-based filtering.

Step 2.1. Search for word triggers.

Step 2.2. Use regular expressions for search.

Stage 3. Deep classification.

Step 3.1. Vectorization.

Step 3.2. Class prognosis.

Stage 4. Saving results.

Stage 5. Analysis and display of results

D. Statement and justification of the problem

In today's information environment, the role of open digital platforms - such as social networks, instant messengers, forums and news feeds - has increased significantly. These tools play an essential role in the dissemination of knowledge, community organisation and communication, but at the same time pose a significant threat as channels for the spread of radical ideologies, disinformation campaigns and, in particular, calls for terrorist actions. In the context of globalisation, hybrid wars and the growing activity of hostile information influences, the problem of detecting terrorist calls in open sources has become highly relevant. Significant volumes of text messages, multilingualism, the use of disguised language, sarcasm, or encryption of threats in metaphors create additional difficulties for manual monitoring and require automation of analysis processes.

Against the backdrop of a growing threat, there is a need to create an intelligent information system capable of automatically analysing textual content and determining whether it contains potential signs of terrorist calls. Such a system should provide support for decision-making based on natural language processing algorithms, semantic analysis, machine learning, and explainable classification. Its implementation will allow a timely response to dangerous publications, reduce the workload on analysts, and also contribute to improving security in the digital space.

The purpose of the system is to detect dangerous textual content that may contain terrorist appeals through comprehensive linguistic and semantic analysis. The system should be integrated into the interfaces of online platforms, use modern deep learning models (including transformers such as BERT), explain its decisions to users and moderators, and learn based on feedback. It can be used in security services, content moderation centres, analytical institutions, and IT companies engaged in cyber defence.

Upon input, the system receives text messages in plain text format, as well as, if necessary, accompanying metadata - language, source, and date of creation. The initial data are the result of

the classification (whether there is a terrorist call in the text), the explanation of this result, the level of probability, the audit log files and the classification database. It is possible to transfer the result to the moderator for confirmation, as well as to initiate retraining of the model based on user complaints.

The intelligent component of the system is based on a combination of rule-based filtering (based on dictionaries, patterns, and regular expressions) and deep semantic analysis. The latter is implemented using the BERT model and trained on the corresponding case of dangerous and safe messages. Additionally, a classification explanation module is built in, which uses attention mechanisms or external tools for interpreting models (for example, SHAP or LIME), which allows you to understand why the model made a particular decision.

The system implements a number of business processes: text input, pre-processing, rule-based filtering, deep classification, explanation generation, logging of results, and feedback processing. In case of an erroneous classification, the user can mark the message as incorrectly recognised, which is automatically transmitted to the moderator. Once the error is confirmed, the case is added to a new training sample to retrain the model. This mechanism ensures the adaptability of the system to new challenges, linguistic patterns and manipulative techniques.

From a technical point of view, the system is designed as a client-server architecture with a REST API for integration into third-party services. On the server side, computing modules are implemented: classifier, filter, explainer, database and logging module. The user interface provides the ability to enter messages, view results, and provide feedback. The architecture allows scaling, horizontal expansion, classification history preservation, and analytics.

The expected effects of the implementation of such a system are: reducing the human burden on moderation, increasing the efficiency of threat detection, reducing the number of erroneous decisions, increasing user trust in digital platforms, and integrating analytical information into the work of law enforcement agencies or cyber defence organisations. Thus, the development and implementation of this system is justified from the point of view of social benefit, technical feasibility and strategic importance in the conditions of information warfare.

Taking into account the results of the system analysis, the requirements for the system are defined: the ability to work with unstructured text, ensuring high accuracy of classification, the availability of explanations of solutions, the ability to preserve history and support for retraining. On this basis, a conceptual model has been formed that combines modules of linguistic processing, neural network classification, interpretation, logging and administration, which together form a single adaptive, scalable and secure information system.

E. Methods and means of solving the problem

In today's digital environment, where a significant part of communications is carried out through text messages on social networks, instant messengers, forums and news feeds, the issue of information security is becoming especially relevant. One of the most dangerous forms of threats is the spread of calls for terrorist actions, which can have direct or veiled wording, be formulated in the form of hints, metaphors or code phrases, which makes it much more difficult to detect them by traditional methods. The growth of information traffic, multilingual content and limited human resources necessitate the creation of automated systems capable of effectively detecting such messages in real time.

The primary function of the intelligent information system under development is to detect terrorist calls in the text environment. Such a system should analyse the text, identify potentially dangerous content, classify messages according to the level of risk, generate explanations of the results of the analysis and provide the possibility of adaptation based on feedback. To implement this functionality, it is necessary to choose appropriate methods of knowledge presentation, logical inference, decision-making algorithms and proper software tools.

Let us describe the developed information technology based on mathematical formulas in order to formalise its operation. Since the system combines rule-based filtering and deep classification (BERT), the model can be represented as follows:

1. Presentation of the text. Each T message is converted into a vector representation:

$$T=\{w_1, w_2, \ldots, w_n\}, \tag{1}$$

where w_i – is message tokens. The BERT model maps them into contextual vectors:

$$E(T)=\{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n\}, \vec{e}_i \in R^d. \tag{2}$$

2. Rule-based filtering. A dictionary-template approach is used:

$$R(T)=\begin{cases} 1, if\ \exists w_i \in T : w_i \in D_{terror} \\ 0, otherwise \end{cases} \tag{3}$$

where D_{terror} is a set of key trigger words/patterns. This module performs fast pre-screening.

3. Deep classification (BERT + Softmax). The CLS vector from the BERT is fed to the full-bonded layer:

$$z=W \cdot \overrightarrow{e_{CLS}}+b. \tag{4}$$

Then the probability of the text belonging to the class is calculated:

$$P(y=k|T)=\frac{\exp(z_k)}{\sum_{j=1}^{K}\exp(z_j)}, k \in \{0,1\}, \tag{5}$$

where $y=1$ – the message contains a terrorist call.

4. Final decision. The final decision is made on the basis of a combination of both modules:

$$F(T)=\alpha \cdot R(T)+(1-\alpha) \cdot P(y=1|T), \tag{6}$$

where $\alpha \in [0,1]$ is the weight factor of the rule-based filter.

5. Quality metrics. To evaluate efficiency, Accuracy, Recall, and F1 score are used:

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN}, Recall=\frac{TP}{TP+FN}, F1=2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}. \tag{7}$$

6. Explainability. The attention mechanisms in BERT allow you to calculate the weights of the importance of words:

$$\alpha_{ij}=\frac{\exp(e_{ij})}{\sum_{k=1}^{n}\exp(e_{ik})}, \tag{8}$$

where α_{ij} – is the contribution of the word w_j to the representation of the word w_i. These values are applied to visualise the most critical tokens when explaining a decision.

Thus, information technology is formalised as a hybrid mathematical model that combines rule-based rules (logical functions) and probabilistic classification based on BERT (softmax model). It allows you to effectively detect both direct and covert calls for terrorist actions. In the course of the work, it is considered which approaches to the presentation of knowledge are relevant for this task, what logical inference mechanisms can be effectively used, what classification algorithms are advisable to use, and which software implementation tools provide the highest performance and flexibility in creating such a system. A comparative analysis of alternative approaches will also be

carried out, on the basis of which the choice of the optimal combination of methods and technologies for solving the task will be substantiated.

F. Methods of presenting knowledge in the system

In decision support systems (DSS) focused on the analysis of text content, a special role is played by the way knowledge is presented, since it forms the basis for further data processing, logical inference and generation of decisions. The presentation of knowledge in such systems can be implemented by various methods - from simple dictionary structures to complex models of semantics representation, depending on the level of complexity of the task, the volume of input data and the required degree of flexibility and adaptability.

Within the framework of this project, a combined approach to knowledge presentation has been chosen, which combines rule-based methods (rules built on the basis of keywords, phrases, and regular expressions) and statistical representation of information implemented using deep learning models, in particular, transformers. The rule-based approach allows you to provide a basic level of filtering based on predefined trigger tokens, patterns, or patterns that characterise terrorist calls in an explicit form. At the same time, it is limited in detecting complex, veiled or atypical wording that is actively used to bypass filters. To overcome these limitations, the system uses knowledge representation based on semantic vectors obtained using the pre-trained transformer model BERT (Bidirectional Encoder Representations from Transformers). In this approach, each message is encoded as a multidimensional vector representation that preserves the contextual meaning of the words in the sentence. This form of knowledge presentation allows you to identify semantic connections and hidden dependencies between words, as well as recognise synonymy, sarcasm, irony, and other forms of manipulative presentation. Thus, the chosen strategy of knowledge presentation combines explicit (declarative) knowledge in the form of rules and implicit (procedural) knowledge in the form of weighted connections in a neural network. It allows you to implement an adaptive system that is capable not only of accurate recognition of known patterns of terrorist rhetoric, but also of detecting new dangerous trends in the text environment based on contextual analysis. This hybrid approach to knowledge presentation provides flexibility, scalability, and high-quality classification while maintaining explainability for the end user.

G. Mechanisms of logical inference

In the context of the development of an intelligent system for detecting terrorist calls, logical inference mechanisms are a critical component that ensures that an informed decision is made based on the input data provided. They determine precisely how the system transforms the incoming text information into a classification result - that is, in response to the question of whether the message contains dangerous content.

Within the framework of the proposed hybrid architecture, two main types of logical inference are used: rule-based inference and machine learning-based inference. Each of the approaches plays a specific role in the overall structure of the treatment.

Deductive inference is implemented through a set of formalised rules based on keywords, phrases, contextual patterns, and regular expressions. For example, the presence in the text of phrases such as "blow up", "call to arms", "explosives", and "eliminate" can unambiguously identify the message as potentially dangerous. The system scans the incoming text for compliance with these rules and, if triggered, generates a warning about a possible terrorist call. This level of logic is fast and effective at detecting obvious threats, but vulnerable to workarounds or veiled statements.

Inductive logical inference, in turn, is implemented using a deep neural network based on the BERT model, which has been trained on a labelled corpus of texts. During training, the model automatically forms internal rules (based on context analysis, grammar, and semantics), which it subsequently applies to classify new messages. In the process of inference, the model compares the context of the message with previously learned patterns and makes decisions with the appropriate

level of probability. In contrast to rigid rules, this approach allows the model to identify non-trivial forms of threat expression, including sarcasm, euphemisms, ambiguity, and metaphors.

Special attention should be paid to explainability mechanisms, which are implemented through additional tools such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations). They allow the user or moderator to understand which words or phrases influenced the classification decision. Not only does this increase the credibility of the system, but it also allows you to identify false positives or algorithm abuses.

In general, the combination of formal (deductive) and educational (inductive) inference, complemented by explanation modules, allows for high flexibility, adaptability and efficiency in detecting terrorist calls. This approach meets the requirements of modern cybersecurity and is resistant to language transformations, stylistic variations, and bypass communication techniques.

H. Decision-making algorithms

In the terrorist call detection system, one of the key components is practical decision-making algorithms that ensure accurate recognition of dangerous content in text messages. Taking into account the complexity of the subject area, the ambiguity of natural language, linguistic manipulations and a large amount of input data, it was decided to implement a two-level classification architecture that combines rule-based filtering with deep semantic analysis based on transformer models, in particular BERT.

Rule-based filtering plays the role of the initial stage of message processing. This mechanism is based on predefined dictionaries, lists of dangerous words, triggers, phrases and patterns that are considered typical of terrorist rhetoric or calls for violence. In addition, regular expressions are used to identify non-standard or disguised constructs. The main advantages of this approach are its speed, transparency, and ease of configuration, scaled by updating the rule base without the need for a complete retraining of the system. However, rule-based filtering is not able to effectively work with atypical or veiled expressions, does not understand the context and does not distinguish the meaning of phrases depending on the situation. Because of this, the rule-based module acts as a primary filter, which either immediately filters out clearly threatening messages or passes them to the next level of processing.

Deep classification is implemented using the BERT model – an advanced architecture based on attention mechanisms. It allows the model to "read" text simultaneously in both directions, forming an understanding of the context of words. In particular, messages that contain veiled calls for violence, metaphorical threats, sarcasm or manipulative rhetorical devices. This approach allows the model not only to recognise obvious cases, but also to identify hidden signs of danger that are not always noticeable even to humans.

The choice of BERT among other machine learning architectures was due to its high performance in a wide range of NLP tasks, including classification, question answering, sentiment analysis, and identification of named entities. In addition, transformers work better with long texts than classic models such as LSTM or CNN. They are independent of word order and are capable of establishing remote dependencies between words. It is critical in detecting terrorist calls, as threats can be vague, implicitly formulated, or hidden in complex syntactic constructs.

Special attention is paid to the possibility of explaining the results of the classification. It is an essential requirement, not only from an ethical point of view, but also to increase the credibility of the system, especially if its results are to be used in security services or in administrative decision-making. The system implements two main approaches to explainability: attention-visualisation and the use of SHAP. The attention mechanism built into BERT allows for the highlighting of the words or phrases that have most influenced the model's decision, thus visualising the decision-making process. SHAP, in turn, proposes a local explanation method that is based on game theory and allows you to calculate the contribution of each feature (in this case, a word) to the final result. It enables analysts and moderators to better understand why a particular message is classified as unsafe, and it also makes it easier to identify false positives or new linguistic patterns.

Thus, the set of algorithms - rule-based filtering and transformer classification - provides a high level of accuracy, contextual understanding and transparency. This approach makes it possible to combine the speed and simplicity of classical methods with the intellectual depth and flexibility of modern deep models. It is critically important in an environment where system decisions can have real consequences for public security and human rights.

I. Implementation tools

In the development of a system for automatic detection of terrorist calls in text content, the choice of implementation tools plays a key role, since they determine the performance, scalability, flexibility and convenience of further maintenance of the software. Python was chosen as the primary programming language, and it is the de facto standard in the fields of natural language processing (NLP) and artificial intelligence (AI). Python provides a robust ecosystem of libraries, syntax simplicity, a broad community, and the ability to quickly integrate with other platforms. In addition, this language is ideal for experimental prototyping, which is essential in the process of exploring new models and methods. The key libraries for implementing NLP and machine learning modules are HuggingFace Transformers, SpaCy, and Scikit-learn. HuggingFace's Transformers library provides access to state-of-the-art pre-trained models, including BERT, RoBERTa, DistilBERT, GPT, and more. It allows you to easily implement deep learning models, change configurations, train them on your own dataset, and integrate them with other system components. It is this library that is used to implement the main module of deep classification, which detects terrorist calls based on contextual analysis.

The SpaCy library is used for text preprocessing. It provides fast and efficient tokenisation, lemmatisation, definition of parts of speech, and Named Entity Recognition. It allows you to build a lightweight yet robust word processing pipeline before passing it to the classifier. SpaCy integrates well with other libraries and also allows you to create your own processing components for specific languages or domains.

To implement auxiliary classical machine learning algorithms, in particular, rule-based filters or basic classifiers, the system uses the Scikit-learn library. It provides modules for building SVM, Naive Bayes, decision trees, clustering, data normalisation, and model quality assessment. In our case, Scikit-learn is also helpful in analysing results, cross-validating, constructing accuracy, completeness, and F1 measure metrics.

Flask or Streamlit is used to implement the user interface and provide interactive interaction with the system. Flask is a classic web framework that allows you to build a REST API used as an entry point for integration with other web services or platforms. It is an ideal choice for a server-side implementation that needs to be scalable and independent. Streamlit, on the other hand, allows you to quickly prototype a user interface without in-depth knowledge of web development. Streamlit efficiently implements input fields, buttons, graphs, explainability visualisation (for example, highlighting essential words) and adjusting model parameters.

When comparing TensorFlow and PyTorch – the two main deep learning frameworks – PyTorch was preferred. The main reasons for this choice are better support in the Transformers library, more developer-friendly syntax, dynamic compute graphics (which makes it easier to debug models), a broad community, and stable integration with HuggingFace. PyTorch provides more flexibility when experimenting, in particular when working with custom loss functions or non-standard architectures. While TensorFlow has powerful tools for productive use of models in production, such as TensorFlow Serving or TensorFlow Lite, it is less convenient in the R&D phase.

Thus, the selected tools - Python, HuggingFace Transformers, SpaCy, Scikit-learn, Flask/Streamlit and PyTorch - provide an optimal balance between performance, flexibility, ease of implementation, and support for explainable AI. They make it possible to implement all the functionality of the system - from text processing and deep analysis to output of results and interactive feedback - with a high level of control over the process, rapid prototyping and scalability.

J. Comparative analysis of analogues

In the field of automatic detection of terrorist calls in text content, a number of approaches have been proposed in recent years, each of which has its own advantages and limitations. Among the most common, rule-based systems, models based on convolutional neural networks (CNN) and recurrent neural networks (LSTM) are worth mentioning. Each of these methods deserves attention, but their comparative analysis allows us to clearly justify the choice of a hybrid model that combines rule-based filtering with transformer deep classification.

The rule-based approach is based on predefined rules, keyword lists, regular expressions, and linguistic patterns. Its main advantages are ease of implementation, transparency, and explainability. Such systems quickly respond to new rules and easily adapt to the requirements of moderators. At the same time, their main disadvantage is low flexibility: rule-based methods are not able to detect synonymy, sarcasm, grammatical variations, slang, encryption or contextual ambiguities. In addition, they do not scale when working with large amounts of text or with multilingual content. These shortcomings make rule-based systems insufficient as an independent solution in security tasks.

Another class of models - CNN (Convolutional Neural Networks) - although it has demonstrated high efficiency in computer vision tasks, has certain limitations in the field of NLP. CNN models can capture local patterns and phrases well, but they cannot model long-term dependencies in text. It is especially critical for the tasks of classifying terrorist calls, which are often disguised in complex syntactic constructions or stretched over several sentences. Compared to transformers, CNN models lose flexibility in processing context and explaining decisions.

LSTM (Long Short-Term Memory) is a type of recurrent neural network that has long been the standard in sequential word processing tasks. LSTMs work well with sequential language structures and are able to model a certain amount of context, making them better than CNNs for longer pieces of text. However, LSTMs are slower to learn, less scalable, have a complex architecture, and, importantly, are inferior to modern transformers in accuracy and stability of results. In addition, LSTMs do not have a natural explanation mechanism, which makes them difficult to use in sensitive areas related to safety.

The hybrid model chosen in the work combines the strengths of all these approaches. Rule-based filtering serves as the first protective layer, allowing you to quickly filter obviously dangerous content using linguistic templates. This approach also allows the user to configure rules manually in response to new threats. The second layer, Deep Classification Based on the Transformer Model (BERT), provides accurate and contextually sensitive text analysis, including the ability to detect veiled calls, synonymy, and multilingualism. This approach also supports explainability through attention mechanisms, which allows you to explain why a particular decision was made, which is critical in automated moderation tasks.

Thus, the hybrid model combines the advantages of both worlds: the transparency and simplicity of the rule-based approach with the power and contextual sensitivity of transformers. Such a symbiosis allows you to achieve high accuracy, scalability, adaptability to new threats and reliability, which is especially important in information security.

We will describe in more detail the main stages of our pipeline for detecting terrorist calls in social networks. At the same time, we will compare the features of each of the stages for the analysis of English-language ($T =$ I call to take up arms and fig h t) and Ukrainian-language ($T =$Закликаю взяти зброю і боротися) texts to identify terrorist calls.

Stage 1. Presentation of the text. The purpose of this stage is to formalise the T text message in the form of mathematical objects (vectors, matrices) suitable for further processing by machine learning algorithms.

1.1. Tokenisation (individual tokens / WordPiece). Let the incoming text of the message:

$T =$ I call to take up arms and fig h t (engl.)
and $T =$Закликаю взяти зброю і боротися (ukr.).

Then, after the classic tokenisation (whitespace/word), we get a set of words (tokens):

$$T \to \{w_1, \ldots, w_8\} = \{I, call, ¿, take, up, arms, \wedge, fight\}. \text{ (engl.)}$$
$$\text{and } T = \{w_1, w_2, w_3, w_4, w_5\} \text{ (ukr.)},$$

where $w_1 = Закликаю$, $w_2 = взяти$, $w_3 = зброю$, $w_4 = і$, $w_5 = боротися$.

In the BERT style, subword (WordPiece) is often used, for example:

$$\text{\{"arms"\} \to [\{"arm"\},\ \{"\#\#s"\}],\quad \{"fight"\} \to [\{"fight"\}].}$$

Then the token sequence can get a little longer; Let's denote the number of tokens as N.

1.2. Normalisation and lemmatisation / stemming. Each word is reduced to a basic form (lemma):

$$L(T) = \{l_1, \ldots, l_m\} = \{I, call, ¿, take, up, arm, \wedge, fight\}. \text{ (engl.)}$$

(here "arms" → "arm"). Frequent operations: lowercasing (all lowercase letters), punctuation removal, apostrophe normalisation, etc.

$$L(T) = \{l_1, l_2, l_3, l_4, l_5\}, \text{ (ukr.)}$$

where $l_1 = закликати$, $l_2 = взяти$, $l_3 = зброя$, $l_4 = і$, $l_5 = боротися$.

1.3. Vectorization (Bag-of-Words/BoW and TF-IDF). Let the corpus have N documents and a large vocabulary base with M words (lemmas). Let's build a TF-IDF vector for a T document:

$$\vec{v}(T) = \left(tfidf(l_1, T), tfidf(l_2, T), \ldots, tfidf(l_M, T)\right) \in R^M, \tag{9}$$

where TF-IDF is calculated as:

$$tfidf(l_i, T) = tf(l_i, T) \cdot idf(l_i), \quad tf(l_i, T) = \frac{f_{l_i, T}}{\sum_j f_{l_j, T}}, idf(l_i) = \log \frac{N}{1 + n_{l_i}}. \tag{10}$$

$f_{l_i, T}$ – is the number of occurrences of the word l_i in the text of T, n_{l_i} – is the number of documents where l_i – occurs.

Small numerical English example. Suppose:

- case $N = 10{,}000$ documents;
- The word "arms" (lemma "arm") occurs in $n_{arm} = 50$ documents;
- The word "fight" occurs in $n_{fight} = 200$ documents;
- In our message, the total number of tokens is ¿8, and each of the lems appears 1 time (frequency $f = 1$).

Then

$$tf(\text{arm}, T) = \frac{1}{8} = 0.125, idf(\text{arm}) = \log \frac{10000}{1+50} \approx \log(196.08) \approx 5.28,$$
$$tfidf(\text{arm}, T) = 0.125 \cdot 5.28 \approx 0.66.$$

And for fight:

$$tf(\text{fight}, T) = 0.125, idf(\text{fight}) = \log \frac{10000}{1+200} \approx \log(49.75) \approx 3.91,$$
$$tfidf(\text{fight}, T) \approx 0.125 \cdot 3.91 \approx 0.489.$$

Other words that are frequently encountered (I, to, and) will have small idf and correspondingly small TF-IDFs.

Ukrainian-speaking example. If there are $N=1000$ documents in our corpus, the word "зброя" occurs in 20 of them ($n_{зброя}=20$), and in our text it appears 1 time, then:

$$tf(\text{зброя}, T)=\frac{1}{5}=0.2,\ idf(\textit{зброя})=\log\frac{1000}{1+20}\approx\log 47.6\approx 1.68,$$
$$tfidf(\textit{зброя}, T)=0.2\cdot 1.68=0.336.$$

1.4. Word Embeddings (BERT).

1.4.1 Tokens → vectors. In deep models such as BERT, each word/token is w_i mapped to a context vector:

$$\vec{e}_i=f(w_i)\in R^d,\ \vec{e}_i=\text{BERT}(w_i/\text{context}\,T)\in R^d,\ d\,\text{type}=768. \tag{11}$$

where d is the dimension of the space (for example, $d=768$ in BERT).

So, for the entire T message/text, we have a matrix:

$$E(T)=\begin{bmatrix}\overrightarrow{e_1^T}\\ \overrightarrow{e_2^T}\\ \dots\\ \overrightarrow{e_n^T}\end{bmatrix}\in R^{n\times d}. \tag{12}$$

1.4.2 CLS token and pooling. BERT adds a custom token $[\text{CLS}]$ to the beginning; its vector $\overrightarrow{e_{CLS}}$ – is often used as a representation of the entire sentence: $\overrightarrow{s_{CLS}}=\overrightarrow{e_{CLS}}\in R^d$. Alternatively, you can do mean-pooling by tokens:

$$\overrightarrow{s_{mean}}=\frac{1}{n}\sum_{i=1}^{n}\vec{e}_i. \tag{13}$$

1.4.3 Small Numerical Illustrative English Example. Let's assume $d=3$, for clarity, and we have eight tokens. Contextual vectors:

$$E(T)=\begin{bmatrix}0.01 & 0.10 & -0.05\\ 0.40 & -0.20 & 0.30\\ -0.05 & 0.00 & 0.02\\ 0.10 & 0.25 & -0.10\\ 0.05 & 0.12 & 0.01\\ 0.80 & -0.40 & 0.60\\ -0.02 & 0.03 & 0.00\\ 0.70 & -0.35 & 0.55\end{bmatrix}.$$

Then mean-pooling $\overrightarrow{s_{mean}}=\frac{1}{8}\sum_{i=1}^{8}\vec{e}_i=\left(\frac{1.99}{8},\frac{-0.45}{8},\frac{1.38}{8}\right)\approx(0.249,-0.056,0.173)$. The numbers are illustrative – in honest BERT $d=768$.

Ukrainian-speaking example. For the phrase Закликаю взяти зброю (3 tokens) and $d=3$:

$$E(T)=\begin{bmatrix}0.12 & -0.45 & 0.77\\ 0.02 & 0.88 & -0.31\\ 0.90 & -0.11 & 0.25\end{bmatrix}.$$

This matrix is fed into the further classification module. Thus, at the stage of presenting the text of the message, the following steps are taken:

- Tokenization $T \to \{w_1, \ldots, w_n\}$;
- Lemmatization $w_i \to l_i$;
- Vectorisation (TF-IDF or embeddings);
- Feature matrix $E(T) \in R^{n \times d}$.

1.5 Subword and OOV (out-of-vocabulary). If a word is broken down into a subword (e.g. an unusual slang word), the embedding of each subword is calculated, then they are aggregated (concat/mean/attention-pooling). Formally, if the token $w \to$ subwords $\{s_1, \ldots, s_k\}$ with embeddings $\vec{u}_j$, then

$$\vec{e}_w = \frac{1}{k} \sum_{j=1}^{k} \vec{u}_j \backslash(\text{mean pooling}\backslash). \tag{14}$$

1.6 Additional features (feature engineering). For classic models, numerical features are added:

- Message length $len(T) = n$,
- Number of trigger words $count_{trig}(T) = \sum_i 1[l_i \in D_{trig}]$,
- Part of speech (POS) in the form of one-hot vectors,
- N-grams (bigrams): for example, "take up" and "up arms" → can be represented as binary features.

Let's denote the vector of additional features as $\overrightarrow{f_{aux}} \in R^p$. Then the final input to the light classifier can be

$$\vec{x} = [\vec{s}; \overrightarrow{f_{aux}}] \in R^{d+p}, \tag{15}$$

where $[\cdot; \cdot]$ – concatenation.

1.7 Final formalisation of the presentation:

- Tokenisation: $T \to \{w_i\}_{i=1}^n$.
- Lemmatization: $w_i \to l_i$.
- TF-IDF vector: $\vec{v}(T) \in R^M$ (for classical models).
- Contextual embeddings: $E(T) \in R^{n \times d}$, sentence → $\vec{s} \in R^d$ (CLS or pooling).
- Additional features: $\overrightarrow{f_{aux}}$.
- The final vector for the classifier is: $\vec{x} = [\vec{s}; \vec{v}(T); \overrightarrow{f_{aux}}]$.

1.8 How-To Notes for the Call Detection System:

- *Stop Word Sensitive*. Words like "I", "to", "and" will have a small IDF - their impact is negligible in TF-IDF, but in BERT, they are essential as context.
- *Load on subwords*. Aggressive filter bypasses often use unusual words → WordPiece gives stability.
- *Character normalisation* – digit/symbol substitution, transliteration (russian/Ukrainian → English) may be required for a multilingual corpus.
- *Feature scaling* – before combining TF-IDF and embeddings, you need to specify scales (e.g. vector normalisation).

Step 2. Rule-based filtering is an intermediate step that works before (or in parallel with) deep classification, and provides quick detection of messages with explicit trigger words or patterns. After tokenisation, let the English-language message (tokens – words or subwords)

$$T=\{w_1,w_2,\ldots,w_n\}. \tag{16}$$

Set the trigger dictionary (set of keywords/templates):

$$D_{trig}=\{weapon,bomb,explosion,terrorist\}\ \text{(engl.) and}$$
$$D_{trig}=\{зброя,бомба,вибух,терорист,\ldots\}\ \text{(ukr).}$$

2.1 Binary rule (simple detection) is a formal statement. The filtering function is defined as:

$$R(T)=\begin{cases}1, if\ \exists w_i\in T: w_i\in D_{trig},\\ 0, otherwise\end{cases} \tag{17}$$

That is, if the message contains at least one word from the dictionary, the system marks it as potentially dangerous. Example: $T=$ I call to take up arms and fight $\rightarrow$ after lemmatisation/normalisation may contain arms $\rightarrow$ if *arms* in D_{trig} (or displayed on *weapon*), then $R(T)=1$.

2.2 Scoring rule. Sometimes, not only is the binary decision (0/1) essential, but also the degree of risk. To estimate the degree of risk, we enter the weights $\omega: D_{trig}\rightarrow R_{+¿¿}$. For example:

$$\omega(\text{weapon})=2,\omega(\text{bomb})=3,\omega(\text{explosion})=2.5,\omega(\text{terrorist})=4.$$

Then you can enter the scales:

$$Score(T)=\sum_{i=1}^{n}1\left[w_i\in D_{trig}\right]\cdot\omega(w_i), \tag{18}$$

where $1[\cdot]$ is an indicator function, $\omega(w_i)$ – is the weight of the trigger word w_i.

Example: $T=$ They plan to plant a bomb and cause an explosion $\rightarrow$ tokens contain *bomb* and *explosion*.

$$Score(T)=\omega(\text{bomb})+\omega(\text{explosion})=3+2.5=5.5.$$

For example (ukr): зброя $\rightarrow \omega=2$, бомба $\rightarrow \omega=3$, вибух $\rightarrow \omega=2.5$. If the message:

$$T=\{закликаю,взяти,зброю,і,боротися\},$$

Then $Score(T)=1\cdot\omega(зброя)=2$.

2.3 Patterns (regular expressions (regex)/n-grams/phrase patterns). In addition to individual words, you can specify expression patterns. Add a set of P patterns (regular expressions), for example:

- r"\bplant(ed)?\b.*\bbomb(s)?\b" — «plant a bomb / planted bomb»;
- r"\btake up arms\b";
- r"\bblow (.+)?up\b".

Match indicator:

$$R_{pattern}(T)=\begin{cases}1,\exists p\in P: T\quad p,\\ 0, otherwise.\end{cases} \tag{19}$$

Example: We will plant a bomb tomorrow $\rightarrow$ satisfies $plant.*bomb\Rightarrow R_{pattern}(T)=1$.

Example (ukr): $P = \{ взяти .* зброю, закликаю .* боротися \}$. Then:

$$R_{pattern}(T) = \begin{cases} 1, \text{if } T \text{ satisfies } p \in P, \\ 0, otherwise. \end{cases} \quad (20)$$

Example. The message "I urge you to take up arms and fight" → satisfies the "take .* arms" pattern. So $R_{pattern}(T) = 1$.

2.4 Obfuscation treatment (leet/character substitution/spacing). Attackers can write $b0mb$, $b\,o\,m\,b$, $b@mb$, $weap0n$, etc. For persistence, use:

1. Normalisation – replacing numbers with letters, removing unnecessary spaces/symbols:

$$\text{normalize}(b0mb) = bomb.$$

2. Wildcard recognition / fuzzy matching (Levenstein): for each token w and the t trigger, calculate the distance $d(w,t)$. Enter a k threshold (e.g. $k=1$ or word length-dependent). Normalised germination:

$$sim_{lev}(w,t) = 1 - \frac{d(w,t)}{max(|w|,|t|)}. \quad (21)$$

We consider it a coincidence if $sim_{lev}(w,t) \geq$ (for example, $\tau = 0.75$). Example: $w = \text{b0mb}, t = \text{bomb}, d = 1 (0 \leftrightarrow o) \rightarrow sim_{lev} = 1 - 1/4 = 0.75 \Rightarrow$ coincides at $\tau = 0.75$.

2.5 Combined rule-score (words + patterns). Let's combine words, patterns, and fuzzy-match into a single formula:

$$R_{word}(T) = \sum_{i=1}^{n} \sum_{t \in D_{trig}} 1\left[match(w_i, t) \right] \cdot \omega(t), \quad (22)$$

where $match(w_i, t) = 1$ if exact or fuzzy-match.

The final rule-based metric can be calculated as:

$$R_{final}(T) = \beta_1 \cdot R(T) + \beta_2 \cdot R_{pattern}(T) + \beta_3 \cdot Score(T), \quad (23)$$

where $\beta_1, \beta_2, \beta_3$ – are the weight coefficients adjusted empirically. Or:

$$R_{final}(T) = \beta_1 R_{word}(T) + \beta_2 R_{pattern}(T) + \beta_3 R_{context}(T), \quad (24)$$

where $R_{pattern}(T) \in \{0,1\}$ (can be weighted in terms of ¿2), $R_{context}(T)$ – additional contextual features (see below), β_j – are weights (for example, $\beta_1 = 0.5, \beta_2 = 0.4, \beta_3 = 0.1$). Threshold solution:

$$\text{label}(T) = \begin{cases} flagged, _{final}^{\square}(T) \geq \theta, \\ notflagged, R_{final}(T) < \theta. \end{cases} \quad (25)$$

Numeric example for the message T h ey plan to plant a b 0 mb tomorrow :

- *bomb* fuzzy-match → $\omega(bomb) = 3 \Rightarrow R_{word} = 3$.
- $plant .* bomb$ pattern matched $\Rightarrow R_{pattern} = 1$. At $\beta_1 = 0.5, \beta_2 = 0.4, \beta_3 = 0$:

$$R_{final} = 0.5 \cdot 3 + 0.4 \cdot 1 = 1.5 + 0.4 = 1.9.$$

At the threshold $\theta = 1.0$ → the message is flagged. Example:

- $R(T) = 1$ (because there is a зброя),

- $R_{pattern}(T)=1$ (because there is a pattern to взяти .* зброю),
- $Score(T)=2$.

At $\beta_1=0.4, \beta_2=0.4, \beta_3=0.2$: $R_{final}(T)=0.4 \cdot 1+0.4 \cdot 1+0.2 \cdot 2=1.4$. This number can be interpreted as a risk level. If $R_{final}(T)\geq$ (where θ is the threshold, e.g. 1.0), the message is marked as suspicious.

2.6 Limitations of the Rule-based method are easy to get around by paraphrasing (for example, w 8 apons instead of weapons). It gives a lot of false positives (for example, the museum of weapons is not a call for violence). Doesn't work well with multilingual or transliterated texts. That's why your research uses rule-based as a quick first-level filter, and then the text is checked by the BERT model. So, Stage 2 is the formalisation of the rule for checking the presence of trigger words and patterns, with the possibility of weight counting and integration with the ML model. It works according to the formulas:

$$R(T)=1\left[\exists w_i \in T : w_i \in D_{trig}\right], Score(T)=\sum_{i=1}^{n} 1\left[w_i \in D_{trig}\right] \cdot \omega(w_i). \quad (26)$$

2.7 Contextual rules to reduce false positives. Some phrases use trigger words in a neutral/non-dangerous context: museum of weapons , explosion of creativity , terrorist attack \(h istory lesson \). Enter a negative dictionary D_{neg} and subtract the negative weight:

$$NegScore(T)=\sum_{i} 1\left[w_i \in D_{neg}\right] \cdot v(w_i), v>0. \quad (27)$$

Final score:

$$R_{adj}(T)=R_{final}(T)-\gamma \cdot NegScore(T), \quad (28)$$

where γ is the mitigation coefficient.

Example: "$The\ museum\ of\ weapons\ exhibited\ WW2\ artifacts$ → *weapon* exists, but $the\ museum$ in D_{neg} with $v(\text{museum})=2$. If $R_{final}=2$ and $\gamma=1$:

$$R_{adj}=2-1 \cdot 2=0 \Rightarrow \text{not flagged}.$$

2.8 Multiple languages/transliteration. For multilingual platforms, it is essential to:

- perform language detection (language ID) and select the appropriate dictionary;
- Transliterate (e.g. $te\,r\,ro\,r\,ist \rightarrow terrorist$) and use fuzzy-matching.

2.9 Practical examples (in English).

1) We must gat h er weapons for defense :

- Tokens: $we, must, gat\,h\,er, weapons, for, defense$;
- $weapons \rightarrow$ normalizes$\rightarrow weapon \in D_{trig} \Rightarrow R(T)=1, Score=2$
- The context $for\ defense$ can reduce the risk ⇒ Negative context rule (e.g., $defence, museum, training$) lowers the score.

2) T h ey are terrorists : *terrorists* → lemma *terrorist* ⇒ high weight 4 ⇒ $Score=4$ ⇒ flagged.

3) He read about t h e 1945 explosion : *explosion* matched → $Score=2.5$. But if $the\ context$ contains $history/1945$ → apply context rule → lower priority (possible non-threat), so subtract $NegSc\,o\,\Re$.

4) Obfuscation: b o m b → normalise remove spaces → bomb → matches.

2.10 Limitations and recommendations:

- False positives are essential when combining rule-based with an ML classifier (BERT) because the rules provide a quick but coarse filter.
- Constant updating of dictionaries – add new variants (slang, codes) through feedback.
- Threshold adaptation – configure θ, β_j, on the validation sample (Precision/Recall tradeoff).
- Explainability – save which words/patterns worked – this helps the moderator.

2.11 Integration with the ML module. Often, the rule-based module works as a prefilter and/or as a feature for the model:

- As an immediate trigger: if $R_{adj}(T) \geq \tau_{high}$ – automatically flag and call the moderator.
- As features to ML: add numeric features $\left[Score(T), R_{pattern}(T), NegScore(T)\right]$ to the vector $\vec{x}$ – for BERT/SVM.

Stage 3. Deep classification (BERT and Softmax) is the stage where the system, after rule-based filtering, proceeds to an in-depth analysis of the semantics of the text in order to separate ordinary messages from those containing terrorist calls.

3.1. Input – Login: Tokenised message. Let the message: $T = \{w_1, w_2, \ldots, w_n\}$ after pre-processing/tokenisation, we have a sequence of tokens (WordPiece/tokens), which is fed into BERT in the form of a sequence: $X = \left[[\mathrm{CLS}], w_1, w_2, \ldots, w_n, [\mathrm{SEP}]\right]$. For our example (simplified):

$$T = \textit{Закликаю взяти зброю і боротися} \Rightarrow$$
$$\Rightarrow X = [\backslash\{[\ \mathrm{CLS}\ ]\backslash\},\ \{\ \text{закликаю}\ \},\backslash\{\ \text{взяти}\ \},\backslash\{\ \textit{зброя}\ \backslash\},\ \{\mathrm{i}\},\backslash\{\ \textit{боротися}\ \backslash\},\ \{[SEP]\}].$$

3.2 Contextual vectors with BERT that calculate contextual embeddings for each token (BERT converts each token into a feature vector):

$$E(X) = \{\overrightarrow{e_{CLS}}, \vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n, \overrightarrow{e_{SEP}}\}, \vec{e}_i \in R^d, \tag{29}$$

where d is the dimension of the embedding space (usually $d = 768$). The standard $d = 768$ (to illustrate below, we will use a minimal dimension $d = 3$).

3.3. Representation of the whole sentence. For classification, the vector [CLS] is used (a representation of the entire sentence is usually the vector [CLS]): $\vec{h} = \overrightarrow{e_{CLS}} \in R^d$. Alternatively, mean pooling can be applied:

$$\overrightarrow{h_{mean}} = \frac{1}{n}\sum_{i=1}^{n} \vec{e}_i. \tag{30}$$

In our technology, the CLS vector is used.

3.4. Linear transformation, i.e. linear layer – logit. The vector $\vec{h}$ is fed into a dense (fully connected) layer:

$$z = W \cdot \vec{h} + b, W \in R^{K \times d}, b \in R^K, \tag{31}$$

where K is the number of classes (in our case, $K = 2$: 0 is plain text, 1 is a terrorist call).

That is, we project $\vec{h}$ into the class space (here $K = 2$: 0 is "not a call", 1 is a "call"):

$$z = W\vec{h} + b,\ W \in R^{2\times d},\ b \in R^2, \tag{32}$$

$z = [z_0, z_1]$ – logits (scores) for classes.

Numerical illustrative example (simplified). Let $d = 3$. Suppose:

$$\vec{h} = \begin{bmatrix} 0.30 \\ -0.20 \\ 0.50 \end{bmatrix}, W = \begin{bmatrix} 0.5 & -0.2 & 0.1 \\ -0.3 & 0.8 & 0.7 \end{bmatrix}, b = \begin{bmatrix} 0.0 \\ -0.1 \end{bmatrix}.$$

Let's calculate $z = W\vec{h} + b$. First component:

$$z_0 = 0.5 \cdot 0.30 + (-0.2) \cdot (-0.20) + 0.1 \cdot 0.50 + 0.0.$$

Calculation by numbers: $0.5 \cdot 0.30 = 0.150, (-0.2) \cdot (-0.20) = 0.040, (-0.1 \cdot 0.50 = 0.050.$ Sum $0.150 + 0.040 + 0.050 = 0.240$. So $z_0 = 0.240$.

The second component: $z_1 = -0.3 \cdot 0.30 + 0.8 \cdot (-0.20) + 0.7 \cdot 0.50 + (-0.1)$.

Calculations $-0.3 \cdot 0.30 = -0.090, 0.8 \cdot (-0.0.20) = -0.160, 0.7 \cdot 0.50 = 0.350$. Sum of the previous: $-0.090 - 0.160 + 0.350 = 0.100$. Add $b_1 = -0.1$; $0.100 - 0.100 = 0.000$. So $z_1 = 0.000$.

So $z = [0.240, 0.000]$.

3.5. Softmax classification (probabilities). The probability of assigning a message to the k class:

$$P(y = k / T) = \frac{\exp(z_k)}{\sum_{j=0}^{1} \exp(z_j)}, k \in \{0,1\}. \tag{33}$$

Final prognosis:

$$\hat{y} = arg\,\max_k P(y = k|T). \tag{34}$$

For our example:

$$\exp(z_0) = \exp(0.240) \approx 1.271249, \exp(z_1) = \exp(0.000) = 1.000000.$$

(Calculation: $e^{0.240} \approx 1.271249$.) Sum: $S = 1.271249 + 1.000000 = 2.271249$. Probability:

$$P(y = 0 / T) = \frac{1.271249}{2.271249} \approx 0.5597,\ P(y = 1 / T) = \frac{1.000000}{2.271249} \approx 0.4403.$$

So, the model in this example estimates the probability that the message contains a call as ≈ 0.44. These are illustrative numbers; the honest BERT gives other $\vec{h}$ and, accordingly, other probabilities.

3.6. Training loss function (Loss). Cross-entropy is used for training, that is, during additional training, multiclass cross-entropy is used (for two classes, it is a binary cross-entropy / log-loss):

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k} \cdot \log P(y = k|T_i), \text{ or } L = \frac{-1}{N}\sum_{i=1}^{N}\sum_{k=0}^{1} y_{i,k} \log P(y = k / T_i), \tag{35}$$

where N is the number of examples in the dataset, $y_{i,k}$ – is the one-hot label of the class (e.g., for a "call" $y = [0,1]$).

In case of a strong imbalance of classes, class weights are used – w_k:

$$L=\frac{-1}{N}\sum_{i=1}^{N}\sum_{k=0}^{1} w_k\, y_{i,k}\log P\left(y=k/T_i\right). \tag{36}$$

3.7. Calculation example. Let's take the text: $T=I\ call\ ¿\ take\ up\ arms\wedge fight$.

Step 1. Embedding. BERT calculates the CLS vector (simplified): $\vec{h}=[0.12,-0.34,0.58,\ldots,0.07]\in R^{768}$.

Step 2. Linear projection. Let W and b give: $z=[1.2,-0.8]$.

Step 3. Softmax:

$$P(y=0|T)=\frac{e^{1.2}}{e^{1.2}+e^{-0.8}}\approx\frac{3.32}{3.32+0.45}\approx 0.88,\ P(y=1|T)=\frac{e^{-0.8}}{e^{1.2}+e^{-0.8}}\approx\frac{0.45}{3.32+0.45}\approx 0.12.$$

Here, the system considers the text neutral ($y=0$), but if the training has similar data from arms in the meaning of the weapon, the model can adjust the weights so that $P(y=1)$ is higher.

3.8. The mechanism of attention for explainability. BERT is a multi-layer transformer with an attention mechanism. Internal attention scales allow you to determine which tokens had the most significant impact. BERT uses multi-head attention. Importance weights are calculated as:

$$\alpha_{ij}=\frac{\exp\left(\frac{q_i\cdot k_j}{\sqrt{d_k}}\right)}{\sum_{m=1}^{n}\exp\left(\frac{q_i\cdot k_m}{\sqrt{d_k}}\right)}, \tag{37}$$

where q_i – is the query vector for token I, k_j is the key of the J token, d_k – is the dimension of the key space.

It allows you to interpret which words had the most significant impact on the decision (e.g. weapon, bomb, terrorist). Practically, it is possible to visualise that the words зброя, закликаю, боротися have high attention weights and thus influenced the final vector $\vec{h}$. Additionally, SHAP/LIME is used for local explanations (the contribution of each token to the final probability).

3.9. Final combination with rule-based (ensemble). Often, the final solution is combined with a score from the rule-based module (stage 2). Let $R_{rule}(T)\in[0,\infty)$ be the normalised rule-score (can be reduced to $[0,1]$). Then the final score of the system:

$$Final(T)=\alpha\cdot R_{rule}(T)+(1-\alpha)\cdot P(y=1/T), \tag{38}$$

where $R_{rule}(T)$ is a rule-based score (from stage 2), $P(y=1\vee T)$ is the probability of the terrorist call class from BERT, α is the weight factor, $\alpha\in[0,1]$. Thus, at this stage, BERT with Softmax provides semantic text recognition and screens out cases where rule-based approaches have failed.

Combination example. Suppose:

- Rule-based found the weapon $\rightarrow R_{rule}=1$ (normalised),
- BERT gave $P(y=1/T)\approx 0.4403$ (above),
- Choose $\alpha=0.3$.

Then $Final(T)=0.3\cdot 1+0.7\cdot 0.4403=0.3+0.30821=0.60821$. If the threshold for the flag is $\theta=0.5$, then $Final(T)=0.60821\geq\Rightarrow$ message is marked as suspicious.

3.10 Settings, imbalance problem and thresholds:

- Class imbalance (there are many fewer calls): w_k – weighting, oversampling of positive examples, and focal loss are used.
- Tuning α and θ is performed on a validation set, optimising F1 or Recall at limited FP.
- Calibration of models (Platt scaling or isotonic) helps if you need to interpret the output as a probability.

3.11 Practical Illustration of the pipeline (steps):

1. Entrance: Закликаю взяти зброю і боротися.
2. Tokenisation → WordPiece → X sequence.
3. BERT → get $\overrightarrow{e_{CLS}} = \vec{h}$.
4. The FC layer → logites z.
5. Softmax → $P(y=1/T) \approx 0.44$.
6. Rule-based gives $R_{rule} = 1$ (trigger зброя).
7. A combination (e.g. $\alpha = 0.3$) → $Final \approx 0.61$ → flagged at $\theta = 0.5$.
8. To explain, we show the highlight: зброя, закликаю, боротися are the main affected tokens.

3.12 Tips to improve accuracy:

- Collect additional bulleted examples of veiled calls (for BERT training).
- Use the Ukrainian-language or multilingual BERT model (fine-tune on your case).
- Combine attention visualisation with SHAP for better explanations.
- Configure α and θ according to priorities (for example, the highest Recall if it is more important not to miss any calls).

Stage 4. Final Solution: Formalisation, Formulas, Examples – our information technology combines rule-based results (stage 2) and deep classification (stage 3) to obtain a single decision on whether the message contains a call for terrorist action.

4.1. Problem statement and input values. After stages 1–3, the system has two sources of information:

1. Rule-based filter (normalised):

- Binary indicator of the presence of trigger words or patterns $R(T) \in \{0,1\}$.
- Or for simplicity, the often more general rule-score: $R_{score}(T) \in [0,1]$, (1 – trigger found) – the result of the rule-based module, normalised to the interval [0,1] (for example, by dividing by the maximum weight).

2. Probability from the BERT classifier: $P(y=1|T) \in [0,1]$– output of BERT (probability of "call to violence") which reflects the assessment of the neural network that the text is a terrorist call .

(Optional) Meta-features $\overrightarrow{f_{aux}}$ – e.g. user reputation, time, platform. Let's mark the input message: $T = I\ call\ ¿\ take\ up\ arms \wedge fight$. Suppose (approximate numbers):

- rule-module found $arms \rightarrow R_{score}(T) = 1.0$.
- BERT gave $P_B(T) = 0.44$ (as in the previous illustrative example).

4.2. Simple linear fusion (weighted sum). Most often, a simple linear combination is used:

$$Final(T) = \alpha \cdot R_{score}(T) + (1-\alpha) \cdot P_B(T), \alpha \in [0,1]. \quad (39)$$

The final message score is defined as a weighted amount:

$$Final(T)=\alpha \cdot R_{score}(T)+(1-\alpha) \cdot P(y=1|T), \quad (40)$$

where $\alpha \in [0,1]$ – is the parameter that determines the trust of the rule-based part. If α is extensive → the system relies more on dictionary-template analysis. If the α is small → the system trusts the deep BERT classifier more.

Example 1 is conservative (greater trust in the rules). Let $\alpha=0.6$. Then:

$$Final(T)=0.6 \cdot 1.0+0.4 \cdot 0.44=0.6+0.176=0.776.$$

At the threshold $\theta=0.5$, we have $Final(T)=0.776 \geq \Rightarrow$ flagged.

Example 2 – More trust in BERT. Let $\alpha=0.2$:

$$Final(T)=0.2 \cdot 1.0+0.8 \cdot 0.44=0.2+0.352=0.552 \geq 0.5 \Rightarrow \text{flagged}.$$

As you can see, at these numbers, even a moderate weight of the rule is enough to raise the final score above the threshold.

4.3. The logical rule of the "hard rule" is a threshold solution. Sometimes a hard rule applies:

$$\hat{y}(T)=1 \text{ if } R_{score}(T)=1 \wedge R_{conflict_{safe}}(T)=0, \quad (41)$$

where $R_{conflict_{safe}}(T)$ – is an indicator of the softening context (museum, historical, defence, training). That is: if an apparent trigger is found and there is no obvious safe context, flag instantly (escalation to the moderator).

Example for I call to take up arms and fig h t there is no word museum or training → the rule flags. Final classification decision:

$$\hat{y}(T)=\begin{cases}1, & Final(T) \geq \theta, \\ 0, & inal(T)<\theta,\end{cases} \quad (42)$$

where $\theta \in [0,1]$ is the threshold value (set experimentally).

4.4. Generalised formulation. The final algorithm can be presented as:

$$\hat{y}(T)=1\left[\alpha \cdot R_{score}(T)+(1-\alpha) \cdot P(y=1|T) \geq \theta\right]. \quad (43)$$

4.5. Interpretation and flexibility:

- If the message contains a strong rule-based trigger (for example, bomb or terrorist), then even if the BERT score is low, it will still affect.
- If there are no obvious triggers, but BERT evaluates the semantics as dangerous (for example, a hidden call), then the decision will depend on $P(y=1/T)$.
- The combination allows you to balance between high rule-based accuracy (on simple cases) and neural network flexibility (on veiled cases).

4.6. Example in English (for comparison). Text: $T=$ *They plan ¿ plant a bomb tomorrow*,

- Rule-based: *bomb* → $R_{score}(T)=1.0$.
- BERT: $P(y=1|T)=0.83$.
- At $\alpha=0.4$:

$$Final(T)=0.4 \cdot 1+0.6 \cdot 0.83=0.4+0.498=0.898.$$

If $\theta=0.5 \rightarrow \hat{y}(T)=1$ (call detected).

Thus, Stage 4 is making the final decision using a combined formula: rule-based + BERT, with a threshold value. It avoids both omissions of dangerous messages and excessive false positives.

4.7. Numerical example (Ukrainian text). Notice: $T = \textit{Закликаю взяти зброю і боротися}$.

Step 1. Rule-based:

- The dictionary contains the word зброя.
- So $R_{score}(T) = 1.0$.

Step 2. BERT classification. The model rated: $P(y=1|T) = 0.44$.

Step 3. Combination. Suppose $\alpha = 0.3$: $Final(T) = 0.3 \cdot 1.0 + 0.7 \cdot 0.44 = 0.3 + 0.308 = 0.608$.

Step 4. Comparison with the threshold. Let $\theta = 0.5$. Since $Final(T) = 0.608 \geq 0.5$, we get:

$$\hat{y}(T) = 1 \Rightarrow \text{message is classified as a call.} \tag{44}$$

4.8. Bayesian or logit fusion (alternative). You can convert R_{score} and P_B – to logics and add up:

$$\text{logit}(p) = \ln \frac{p}{1-p}. \tag{45}$$

Then

$$L = w_1 \cdot \text{logit}(R') + w_2 \cdot \text{logit}(P_B) + b,\ Final = \sigma(L), \tag{46}$$

where R' is a tightly normalized rule-score (to avoid logit(0/1) problems, soft anti-aliasing is used, e.g. $R' = \epsilon + (1-2\epsilon) R_{score}$ with $\epsilon = 10^{-6}$, σ – sigmoid. This approach gives a more flexible calibration.

4.9. Composition with additional features (meta-decision). Sometimes a small second-level model (meta-classifier) is used, which takes as features: $\{R_{score}, P_B, count_{triggers}, user_{rep}, time_{\text{¿}}\}$ and outputs Final through logistic regression or a small neural network:

$$Final(T) = \text{metaModel}\left(\left[R_{score}, P_B, \overrightarrow{f_{aux}}\right]\right). \tag{47}$$

It allows, for example, to increase the risk for young accounts or for mass repetitions.

4.10. Threshold decision and escalation policy. After receiving the Final, select a policy depending on the level:

- If $Final \geq \theta_{auto}$ → an automatic block/flag for law enforcement officers (hard – only with very high trust).
- If $\theta_{mod} \leq Final < \theta_{auto}$ → sent to the moderator (person's check).
- If $Final < \theta_{mod}$ → marked as "unsuspicious".

For example: $\theta_{mod} = 0.5, \theta_{auto} = 0.9$. For our example with $\alpha = 0.6$, $Final = 0.776 \Rightarrow$ pass to the moderator (not autoblock).

4.11. Examples (scenarios):

- Scenario A is clearly dangerous, T: We will plant a bomb at the stadium, $R_{score} = 1, P_B = 0.92$, $\alpha = 0.3$ → $Final = 0.3 * 1 + 0.7 * 0.92 = 0.944 \geq \theta_{auto} = 0.9$ ⇒ auto-escalation.
- Scenario B is questionable/satirical, T: I call to take up arms and fight (possibly in the context of historical discussion), $R_{score} = 1, P_B = 0.44$, $\alpha = 0.3$ → $Final = 0.552 \Rightarrow$ moderator (context check).

- Scenario C — technical/museum, T: Visit the weapons museum, $R_{score}=1, P_B=0.05, NegContext=1 \rightarrow$ apply a negative score: $R_{adj}=R_{score}-\gamma \cdot \neg¿1-1 \cdot 1=0$. Then Final is low → not flag.

4.12. Calibration and selection of parameters. The threshold θ, weights α, θ_{auto}, θ_{mod} – are configured on a validation set with metric optimisation (usually F1, or Recall maximisation at limited FP). To limit false positives, it is recommended:

- Use a negative dictionary and contextual rules;
- Calibrate BERT (Platt scaling / isotonic);
- Evaluate the meta-model on different subsets (language, platform).

4.13. Logging, explanation and audit trail. The final decision should include a reason for the moderator/audit:

- What triggers worked (word list and weights).
- BERT-impact (report P_B – and top-k attention tokens).
- It allows the human operator to quickly understand the context and make a decision.

4.14. Conclusion in English Example. For the sentence I call to take up arms and fight with assumed values $R_{score}=1, P_B=0.44$:

- A linear combination of $\alpha \in [0.2,0.6]$ gives the Final about $0.55-0.78 \rightarrow$ usually a seizure for the moderator (or flag, depending on the settings).
- If the context confirms the real call (BERT will give a higher P_B), $Final$ will grow and can reach auto-escalation.

Let's make a small overview table to quickly see how the final score changes:

$$Final(T)=\alpha \cdot R_{score}(T)+(1-\alpha) \cdot P_B(T), \quad (48)$$

where $R_{score}(T)=1$ (rule-based trigger found), $\alpha \in \{0.2,0.5,0.8\}$ (different balance rule vs BERT), $P_B \in \{0.2,0.5,0.8\}$ (probability from BERT).

Table 2
Final Value

$P_B \downarrow \alpha \rightarrow$	0.2 (more trust in BERT)	0.5 (balance)	0.8 (more trust in rules)
0.2	$0.2 \cdot 1+0.8 \cdot 0.2=0.36$	$0.5 \cdot 1+0.5 \cdot 0.2=0.60$	$0.8 \cdot 1+0.2 \cdot 0.2=0.84$
0.5	$0.2 \cdot 1+0.8 \cdot 0.5=0.60$	$0.5 \cdot 1+0.5 \cdot 0.5=0.75$	$0.8 \cdot 1+0.2 \cdot 0.5=0.90$
0.8	$0.2 \cdot 1+0.8 \cdot 0.8=0.84$	$0.5 \cdot 1+0.5 \cdot 0.8=0.90$	$0.8 \cdot 1+0.2 \cdot 0.8=0.96$

Interpretation:

- If α is large (0.8) → Final is always high (0.84–0.96), even when BERT is low. It is a conservative approach (don't miss anything).

- If α average (0.5) ⟶ Final = 0.60–0.90, that is, the dependence on both the rule and the BERT.
- If α is small (0.2) ⟶ Final ranges from 0.36 to 0.84. Here, BERT has a greater impact: at low P_B, you can not flag.

IV. Experiments, results and discussion

A. Description of input processing and machine learning models

1) Database description

The database, which is used within the created software, serves as the basis for training and evaluating the model of classification of texts. The primary purpose of this database is to store text messages and their corresponding labels that indicate the nature of the content. This implementation uses a CSV file as a data source, but the concept allows migration to a full-fledged relational or document-oriented DBMS. CSV files were not scanned ready-made, but were engaged in searching for information with keywords by which you can determine whether a given message is a threat or just sarcasm/junk. Lists of such words were found on the Internet. The authors added some of them. Here they are: bomb, explosion, armed, terror, jihad, shahid, attack, plan, conspiracy, kill, bullets, blood, explosive, attack, missile, mining, grenade, aggression, victims, explosion, sabotage, headquarters, military, enemy, execution, insurrection, pressure, repression, ruthlessness, conflict, front, positions, looting, murder, incendiary, anti-government, provocation, suspect, suicide bomber, surrender, mobilization, catastrophe, destruction, riots, purpose, capture, terrorist, mine, locator, security, isolation, guerrillas, bombing, collaboration, sabotage, torture, betrayal, order, isolation, armed, fuse, genocide, blow, martial law, radicalization, grouping, defense, violence, hearth, retribution, TNT, death, punishers, massacre, enemies, special operation, capitulation, oath, executioner, rebels, sectarians, sniper, shelling, commander, ambush, chaos, ultimatum, inhumans, poison, capture, hostages, trenches, shell casings, explosive device, tactical, homemade, trap, fire, poison, rupture, target, artillery shelling, operation, regime, propaganda, offensive, bloodshed, target, armed conflict, armed confrontation, confiscation, destruction of targets, strategy, revolution, colony, empire, security forces, special forces, evacuation, isolation, threat. Based on these words, artificial intelligence https://chatgpt.com/ parsed this data and generated a 4000-line dataset in UKRAINIAN, which contains both terrorist expressions and slogans, as well as sarcastic ones (specific words, phrases, and full-fledged sentences). Next, we will describe the structure of the database in more detail. Table structure:

- Text - the text of the message to be classified. It is the main field that is fed to the input of the model. Before saving to the table, pre-processing is carried out above the text: removal of HTML tags, punctuation marks, extra spaces, and conversion to lower case.
- Label - numerical designation of the class. In the current version, the system recognises two categories: 0 - sarcastic messages, 1 - messages with signs of terrorist content.

Table 3
Example of a table fragment

English	Ukrainian	label
The holy war begins	священна війна починається	1
Don't take it seriously, but let's blow the city to its foundations	не сприймай серйозно, але підірвемо місто до основ	0

exploding tonight	вибухаємо сьогодні вночі	1
Fortunately, we won't blow up the city to its foundations	на щастя, ми не будемо підірвемо місто до основ	0
The death of the government is inevitable	смерть уряду неминуча	1
Yo, this ain't no threat, but let's blow the city to its foundations	ой, це не погроза, але підірвемо місто до основ	0
The death of the government is inevitable	смерть уряду неминуча	1
We will destroy the station with explosives	знищимо вокзал вибухівкою	1
I would never say punish enemies with fire, right?	я б ніколи не сказав покараємо ворогів вогнем, правда?	0

In this screenshot, you can see the structure of the source code for database analysis and data preprocessing, and in the following table, we will explain its elements.

Table 4
Code description and explanation

Code	Explanation
clean_text	Text cleaning function.
text = str(text)	Converts the input text to a string (in case it is an object or a number).
re.sub(r'<.*?>', '', text)	Removes HTML tags (e.g. ,).
re.sub(r'[^\w\s]', '', text)	Removes all characters except letters, numbers, and spaces (i.e., punctuation).
re.sub(r'\s+', ' ', text)	Replaces multiple spaces with one.
text.strip().lower()	Removes extra spaces from the edges and lowers the text.

The table itself shows code that clears data of unnecessary constructs. In the string(df['text'] = df['text'].apply(clean_text)), we apply this function to our data. After executing the previous code, the values 1 and 0 were replaced with text values so that the end user could understand the value correctly (= label_names { 0: 'sarcasm', 1: 'terroristic' }). Strings:

- (df['label_name'] = df['label'].map(label_names)) - Rotate the column so that our program can perceive binary values, because the model does not read text data.
- print(df.head()) - shows the user the first five lines of the database
- print("\nCategorization:") - shows the categorisation
- print(df['label_name'].value_counts()) - Shows the number of examples of each category (sarcasm and terroristic).

```
def clean_text(text):
    text = str(text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip().lower()


df['text'] = df['text'].apply(clean_text)

label_names = {
    0: 'sarcazm',
    1: 'terroristic',
}
df['label_name'] = df['label'].map(label_names)


print(df.head())
```

```
label_name
terroristic    2000
sarcazm        2000
Name: count, dtype: int64
```

Figure 2: Dataset information

By this time, we could sufficiently familiarise ourselves with the structure of input and output data already in the system, as well as how they were built, edited, and processed. In the following code, which will be presented later, we divided the database into training and test samples in a ratio of 80:20 using the train_test_split function with the stratify parameter to ensure the balance of classes in each subsample. Thanks to this, the proportional representation of both classes in the training and test data is preserved. After all, our BERT model clearly requires it. The proposed database structure has a number of obvious advantages. First of all, it is characterised by flexibility, which allows the system to be adapted to multi-class classification in the future, for example, by adding categories such as "extremism", "aggression" or "fake news". Due to its scalability, such a structure can easily be transferred to different DBMSs - in particular, MongoDB, PostgreSQL, or SQLite - depending on the technical requirements and scale of implementation. Another advantage is transparency - the minimalist structure of the table provides ease of integration with both machine learning and the API layer of the system.

Several potential extensions are expected in the future. For example, a source field can be added that will capture the source of the message (Telegram, Twitter, etc.), which will allow you to analyse the distribution of messages by distribution channels. The additional date field will provide the ability to analyse the dynamics of messages in the time aspect. You can also introduce a probability_score field that will contain the probability that the message belongs to a particular class - this will provide the possibility of threshold configuration for critical applications such as security systems. As a result, such a structure provides a solid, logically sound and technologically stable basis for training a neural network and its further use in real scenarios of semantic text analysis.

2) Description of the knowledge base

The knowledge base within the software performs a critical function - it provides validity for the process of classifying text messages as containing or not containing signs of terrorist content. This knowledge base is hybrid in nature, as it combines rule-based and flexible (data-driven) rules. At the rule-based logic level, a set of keywords and phrases that are often associated with terrorist or threatening rhetoric is implemented. Terms like: "explosion", "bomb", "eliminate", "jihad", "kill",

"mine", "mined", "terrorist attack" and others. These words are collected from open sources, research in the field of information security, and our own analysis of social media content. If there are such markers in the message, the system gives it an increased level of risk. It is worth noting that the dictionary is easily expanded, updated, and customised, which ensures that the model is relevant in the face of changes in communicative patterns.

At the same time, the leading role in decision-making is played by the data-driven component of the knowledge base, which is implemented in the form of a pre-trained transformer model of the BERT type, even in veiled or sarcastic formulations. BERT is not based on a rigid set of rules. Instead, it forms a multidimensional idea of the meaning of the text based on vector representations of words (embeddings), which allows context and ambiguity to be taken into account. Thus, the knowledge base of the system has a dual structure. It combines fixed rules that give a quick result in the presence of explicit terminology, and adaptive patterns that are stored in the weight coefficients of the neural network and provide deep semantic analysis. This approach allows you to achieve maximum accuracy even in conditions where the message is built using irony, euphemisms or deliberate masking of keywords.

Due to its hybrid nature, the knowledge base not only supports the decision-making process but also acts as a platform for further training and adaptation of the model. In particular, with the help of the fine-tuning method, it is possible to update the model based on new examples, forming feedback between real messages and the internal weights of the classifier. It transforms the system from a simple classification model into a self-learning intelligent platform.

3) Mechanisms of logical inference and decision support

In the developed software, the mechanisms of logical inference and decision support are implemented using a hybrid model that combines heuristic approaches and machine learning algorithms. The primary purpose of these mechanisms is to provide automated analysis of textual content in order to identify signs of terrorist rhetoric - both explicit and veiled.

Step 1. Rule-based: At this level, the software checks the text for keywords predefined in the dictionary of dangerous terms. If such words or phrases are found, the message is marked as potentially threatening. This approach provides quick detection of clearly dangerous language and allows for a fast response to direct calls for violence or terrorism. It can be considered an example of rule-based analysis.

```
def rule_based_check(text, keywords):
    text = text.lower()
    return any(keyword in text for keyword in keywords)
```

Figure 3: Rule-based analysis

Step 2. Deep (semantic) inference: in case of insufficient confidence or lack of trigger words, the system transmits a message for processing by the transformer model BERT. At this stage, the model carries out a complex contextual analysis of the content of the entire message, including sentence structure, lexemes' polysemy, sarcastic formulations and syntactic dependencies. The result of this analysis is a probability vector that is fed to the softmax function. The output values are interpreted as the degree of confidence of the model regarding the message belonging to one of the classes.

```
def predict(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
        pred = torch.argmax(outputs.logits, dim=1).item()
    return label_map[pred]
```

Figure 4: Deep (semantic) inference

For example, in the above code, this function was used for prediction - predict(text), which performs predictions of the input text class using a pre-trained transformer model BERT. At the first stage, the text string is passed to the tokeniser, which converts it into a numerical format convenient for the model to process. At the same time, truncation parameters (truncation=True) are activated to limit the length of the text to 128 tokens and padding (padding=True) so that all sequences have the same length. The parameter return_tensors="pt" specifies that the result should be represented as a PyTorch tensor. Once tokenised, all tensors are transferred to the selected computing environment - either the GPU, if available, or the CPU - to ensure that the model works correctly. It is achieved with the help of a dictionary crawl - each key (input_ids, attention_mask) and its values are transferred to the device.

Next, the block with torch.no_grad() is used, which turns off gradient calculations. It optimises memory usage and speeds up the prediction process, since in this case, we are only classifying the text and not training the model.

The BERT model receives input tensors and generates logits at the output - these are numerical values that correspond to the confidence level of the model for each of the classes. From this vector of logits, a class with maximum probability is selected using the torch.argmax function, which returns the index of the most significant element. The .item() method converts this result from the tensor to a regular numeric value (of type int).

Finally, the resulting class index (0 or 1) is used to access the label_map dictionary, which converts the numerical label into a human-readable textual class name, such as "sarcasm" or "terroristic". The function returns this result as the conclusion of the model.

Decision making is carried out according to the principle of probability maximisation - the class to which the model has given the highest rating is chosen as the final result of the classification. For critical applications, the system can be additionally configured to use confidence thresholds (for example, 0.85), which makes it possible to avoid false positives.

Support for the decision-making process is also provided by user feedback - classification results are sent in real time through the Telegram interface, and key events are logged. If necessary, the system can be further trained (fine-tuned) on new examples, which allows you to adapt the output logic to changes in communication methods or the emergence of new terminology.

In general, the proposed inference engine combines speed, flexibility and contextual accuracy, which is critical for security content analysis systems. Its hybrid nature avoids both the excessive rigidity of classical filters and false oversensitivity, which is inherent in many neural network approaches without heuristic restrictions.

4) Software structure

The structure of the developed software is modular and logically ordered according to the functional steps of processing incoming text messages. The architecture is built taking into account the principles of separation of responsibilities, scalability, and flexibility of integration with other systems, in particular with the Telegram interface and machine learning models. The basis of the software is a classification pipeline, which sequentially performs a number of operations on the text. In the first stage of processing, a special module is used that is responsible for the preliminary cleaning of the message, which includes the removal of HTML tags, punctuation marks, and extra spaces, as well as bringing all characters to lowercase. We have already talked about this. Such normalisation is critically necessary to ensure the stable and coordinated operation of the next stage - tokenisation. The text, cleared of noise, enters the tokeniser, which, using a pre-trained language model, converts it into a sequence of numbers that serve as input for the neural network. A key element of the system is a classification module built on the architecture of the BERT transformer model.

```python
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer
import torch
from torch.utils.data import Dataset, DataLoader
tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased")
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['text'].tolist(),
    df['label'].tolist(),
    test_size=0.2,
    random_state=42,
    stratify=df['label']
)
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=128)
val_encodings = tokenizer(val_texts, truncation=True, padding=True, max_length=128)
```

Figure 5: Normalise a text message.

```python
from transformers import BertConfig, BertForSequenceClassification
config = BertConfig.from_pretrained("bert-base-multilingual-cased", num_labels=2)
model = BertForSequenceClassification(config)
model.to(device)
```

Figure 6: Classification module

This component performs an in-depth semantic analysis of the input text, determining the probability of its belonging to a particular category. The result of the calculations is logit-numerical values interpreted as the model's confidence in the correspondence of the text to a specific class. For the convenience of interacting with the model, a special functional interface predict() has been created, which encapsulates the process of tokenisation, prediction, and interpretation of the result, allowing you to get the final answer with a single call.

External interaction with the user is implemented through the Telegram integration module, which allows you to automatically receive text messages, send them to the classifier and return the result directly to the messenger.

```python
model.save_pretrained("/Users/markiianyatsyshyn/saved_model123")
tokenizer.save_pretrained("/Users/markiianyatsyshyn/saved_model123")
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from telegram import Update
from telegram.ext import ApplicationBuilder, MessageHandler, ContextTypes, filters
import nest_asyncio
import asyncio
nest_asyncio.apply()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BertForSequenceClassification.from_pretrained("/Users/markiianyatsyshyn/saved_model123")
tokenizer = BertTokenizer.from_pretrained("/Users/markiianyatsyshyn/saved_model123")
model.to(device)
model.eval()
label_map = {0: "sarcastic", 1: "terroristic"}
def predict(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=128)
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
        pred = torch.argmax(outputs.logits, dim=1).item()
    return label_map[pred]
async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.message is None:
        return
    text = update.message.text
    print(f"Отримано повідомлення: {text}")
    result = predict(text)
    await update.message.reply_text(f"Аналіз: {result.upper()}")
    print("Відповідь надіслана")
async def main_async():
    app = ApplicationBuilder().token("7839620149:AAGiLnME3Qczf84aB3FNiGIDc9wcBpEy0Cc").build()
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))
    print("✅ Бот запущено. Очікуємо повідомлення...")
    await app.run_polling()
loop = asyncio.get_event_loop()
loop.run_until_complete(main_async())
```

Figure 7: Telegram integration module

An asynchronous loop functions above this level, which ensures the constant operation of the Telegram bot and its readiness to process new requests. An essential part of the structure is the logging module, which records all events, including incoming messages, classification results, and possible errors. It allows you to track the operation of the system and ensures its controllability and reliability.

The software does not contain classic graphical interfaces or macros. Still, it implements a full-fledged command control mechanism via Telegram, making it accessible to end users without the need to install additional applications. This approach allows you to apply development both on-premises and in the cloud, while providing flexibility to scale. The entire system is built according to the principle of low binding and high coherence. Each module performs only its own well-defined function and does not depend on the internal implementation of other components. It ensures that the software is easy to maintain, update, reuse, and adapt to new requirements.

5) Functionality and interaction of components

The developed software is a comprehensive system based on deep semantic recognition of text messages, followed by their classification according to the level of danger. All functionality is implemented on the basis of a modular structure, where each component plays a specific role, but at the same time organically interacts with other elements of the system. The entire cycle of message processing begins from the moment the user receives the text, which comes through the integrated Telegram interface.

Next, the processing logic chain is activated: the input text is passed to the clean-up module, where pre-normalisation is performed - HTML elements, punctuation marks, redundant spaces are removed, and all characters are reduced to lower case.

After that, the text is passed to a tokeniser, which transforms it into a sequence of numbers that serve as input to the neural network. These vectors arrive in a model of the type BERT, which has been previously trained on a classification problem with two classes - "sarcastic" and "terrorist" messages. The model performs a forward pass by calculating logits and probability vectors. Based on the highest value among the probabilities, the system decides which class the text belongs to. The forecast is formed in the form of a text conclusion sent back to the user via Telegram.

All intermediate steps are implemented in the form of functions and subroutines. In addition to the main classification, the logic of event logging is provided, which makes it possible to keep records of requests, store statistics of classification results, and record possible exceptions or anomalous situations. If necessary, each module can be tested separately because it does not depend on the internal implementation of other parts of the system. All functions included in the program work synchronously and sequentially, transferring data between modules in the form of simple objects or dictionaries. It makes it easy to debug the process, scale the solution to cloud computing platforms, or embed it in more complex information systems with cybersecurity needs.

6) Description of the created software tool

The description of the created software tool meets the requirements of GOST 19.402-78 "Program Description" and ISO/IEC 26514:2008 "Systems and software engineering - Requirements for designers and developers of user documentation". As part of this work, intelligent software has been developed for the automatic detection of messages of a terrorist nature. It is based on the BERT transformer model integrated into the Telegram bot, which allows for real-time semantic analysis of texts. The tool is focused on being used as a separate service or as part of larger monitoring systems.

Installing the software requires Python version 3.10 or higher, as well as the installation of appropriate libraries, including transformers, torch, scikit-learn, pandas, telegram, asyncio, and nest_asyncio. To use the Telegram interface, you must have a Telegram account and a generated Bot API access token. Optionally, it is recommended to use a system with CUDA support if you plan to process large amounts of text. Installation is done by creating a project directory, installing

dependencies via pip install -r requirements.txt, adding a token to the TOKEN variable or to a secrets.json file, and checking for the presence of a saved model in the directory (e.g. saved_model123). After that, the software is run through the main script or the Jupyter interactive environment, after which the bot begins processing messages.

The user interface is implemented in the form of a text chat. The software does not have a graphical interface in the usual sense - windows, menus or buttons - but it supports full-fledged interaction through Telegram. The user sends a message that goes through two levels of analysis: 1) superficial rule-based analysis for keywords and 2) in-depth semantic analysis through the BERT model. If trigger terms are detected, the "terroristic" class is immediately returned. If there are none, the message is classified using a neural network, resulting in a text category (sarcasm or terroristic) returned to the user. All requests and responses are logged.

The typical usage scenario is as follows. The user enters a message into the chatbot, which is instantly processed: tokenisation, heuristic comparison, and contextual interpretation through the model are performed, and a response is returned. In this way, the user can check any text for potential terrorist threats, including hidden or sarcastic hints. This functionality can be especially relevant for administrators of public channels, journalists, moderators, or information security services.

Among the possible errors that the system can return, the most common are the lack of connection to the Telegram API, the incorrectly specified path to the model, the lack of the necessary libraries or the wrong access token. All these problems can be fixed by checking the connection, establishing dependencies with pip, or fixing the configuration files. In case of difficulties, reinstalling all modules and restarting the service usually helps.

Since the project is educational, official technical support is not provided. However, flexible modification is provided. For example, model weights and tokeniser are stored in a separate folder and can be reused without the need for retraining. You can also adapt the rule-based dictionary to new terms, train the model with specific examples, or integrate additional confidence threshold control mechanisms.

In general, the software tool is modular, logically connected and covers the whole cycle: from text input to classification return. This approach allows you to ensure compatibility between components, ease of maintenance, and the ability to scale to more complex information security scenarios.

7) User instructions for the information system for detecting terrorist appeals

This software tool allows you to automatically analyse text messages and classify them according to the characteristics of terrorist or sarcastic content. The main interaction environment is the Telegram messenger. The system works on the basis of the BERT model trained on the corresponding corpus of texts. User Requirements:

1. Basic Telegram skills
2. Availability of an Internet connection

For local launch: knowledge of Python, transformers, torch, python-telegram-bot, and nest_asyncio libraries installed.

Installation and Launch (for a developer or system administrator):

1. Install Python (version 3.8 or higher recommended).
2. Create a Telegram bot through @BotFather and get a token.
3. Clone or download a repository with code.
4. Install libraries with the command: nginx

pip install torch transformers python-telegram-bot nest_asyncio

5. Load or save the model to the saved_model/ directory.
6. Run telegram_bot.py in Jupyter Notebook or from a terminal.

Using a Telegram bot:

1. Open Telegram.
2. Find your bot by the name that is specified when creating it in BotFather.
3. Click "Start" or send any message (for example: I'm going to attack tomorrow).
4. Get a response from the bot

Features of interpretation:

- SARCASTIC - the message has no signs of threat.
- TERRORISTIC - lexical or semantic patterns similar to terrorist rhetoric have been identified.

The model has an educational origin, so that the result can be conditional. If in doubt, try again or conduct an additional check.

Safety recommendations:

1) Do not send real threats to the bot.

2) Do not use the system for illegal surveillance or harassment.

B. Analysis of results

1) Input

Block "Input data" with a detailed description of the case (sources, volume, balance of classes). Add a table: number of messages, average length, % of terrorists, etc. Here in Fig. 8, we can see how our categories are distributed, such as terrorist and sarcastic messages, and it is necessary to parse the dataset in such a way that everything is equal, so that the model shows everything as accurately as possible.

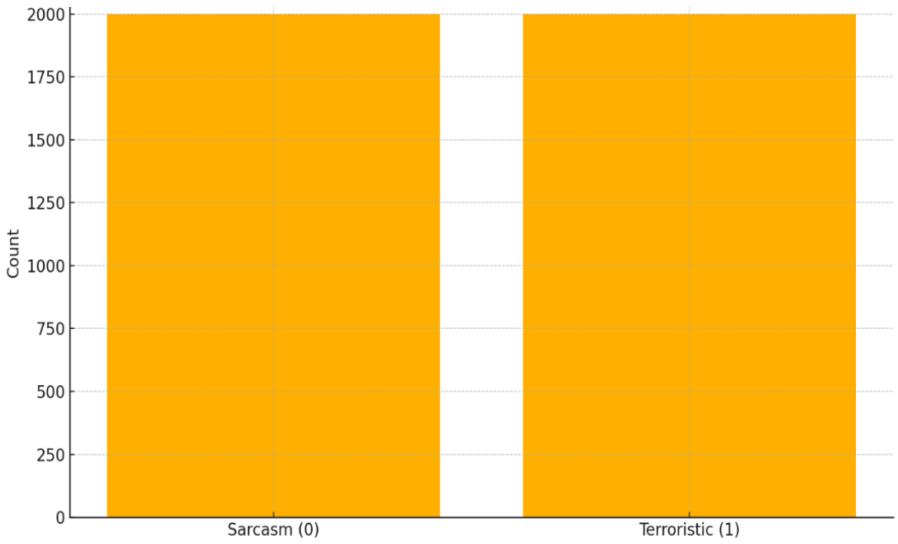

Figure 8: Class distribution

Here we see how our messages are distributed by word length. It is also beneficial for model research. Boxplot Message Length by Class clearly separates the median: for class 0, it lies at ~7 words, while for class 1, it is only ~3–4 words. The model also uses this factor as an indirect signal - short, aggressive statements are more likely to be dangerous. The dataset was assembled independently, using real and synthetic messages from public sources. First, about eight thousand texts were collected from open Telegram channels, as well as from forums and Twitter, where

sarcastic or radical statements are often found. To ensure anonymity and ethical use, all messages have been cleared of personal data, user IDs, emojis, garbage symbols, and links.

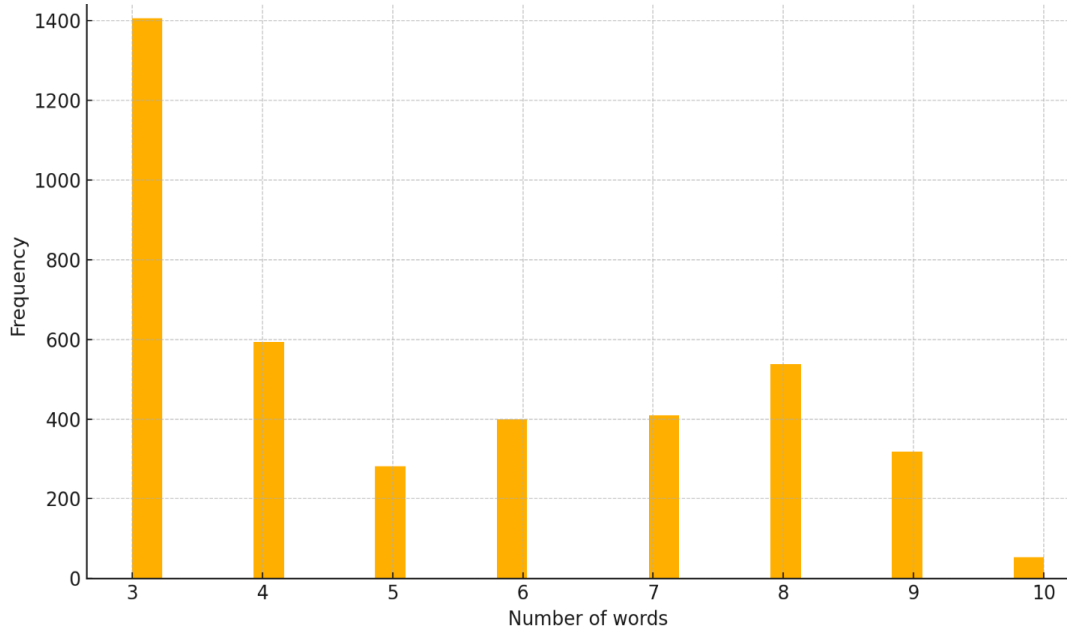

Figure 9: Distribution of massage lengths

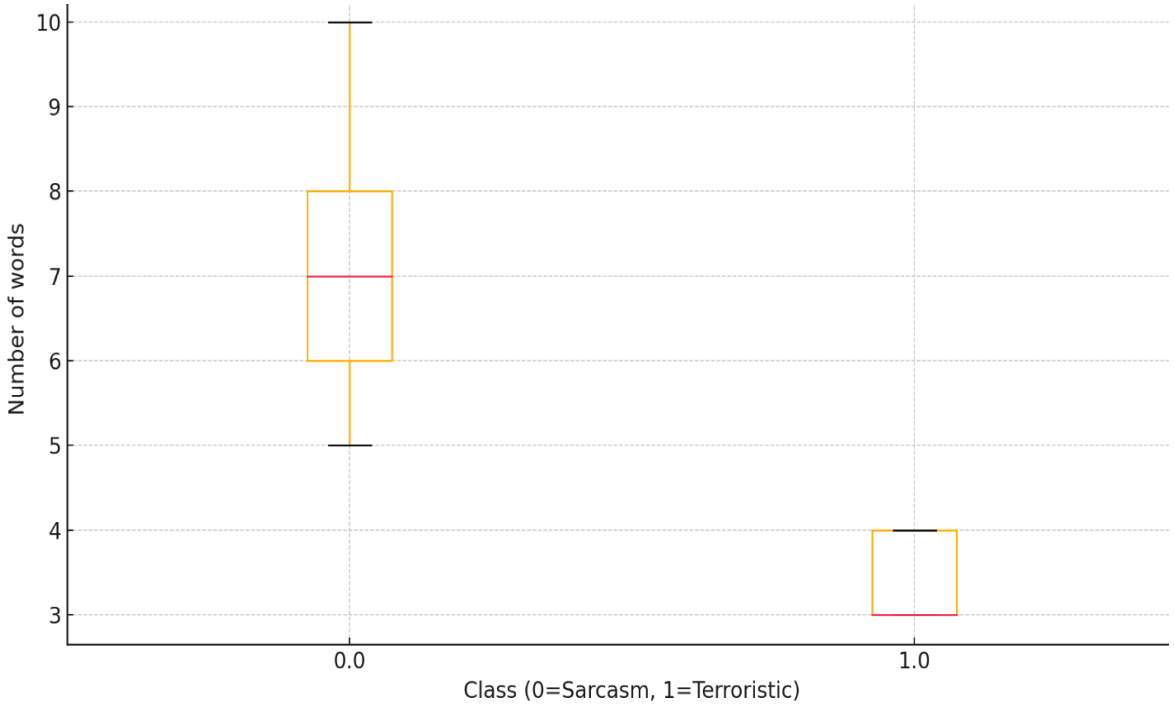

Figure 10: Message length by class

At the initial stage, duplicates were removed using a normalised hash - lower case, punctuation was removed, and MD5 was calculated. It made it possible to weed out about a thousand duplicate lines. Next, a rule-based filter was used, which contained a dictionary of more than 200 key trigger words. If such a trigger occurred in the text, the message was previously marked as potentially terrorist; otherwise, it was sarcastic. It yielded an initial class ratio of approximately 30:70. However, to ensure quality, 1,200 random examples from each class were selected and manually checked. Some of the sarcastic jokes with words like "explosion" have been corrected, because they

had no real terrorist meaning. Instead, some disguised radical messages with no explicit vocabulary were transferred to the "terroristic" class. After that, the balance was manually equalised to get 2,000 messages in each category.

At the final stage, the standard normalisation was performed: I moved all texts to lower case, removed unnecessary spaces, and replaced obscene language with a special tag <obscene>. For further analysis, a length field has also been added, which contains the number of words in the message. It made it easy to build statistics on the length of texts and understand the distribution of content.

Once the data is prepared, the dataset is divided into training, validation, and test samples in an 80/10/10 ratio using train_test_split and the stratify parameter to maintain an even distribution of classes. In addition to this, light synonymous argumentation is applied for sarcastic messages, replacing some words with synonyms via WordNet to train the model to better recognise different forms of expression of sarcasm.

At the final stage, it was checked that none of the texts in the test set intersected with the training set - for this, I used hashing again. All texts were saved in CSV and XLSX formats, and catalogued in the datasets/v1/ directory with an accompanying JSON file, which indicates the date of formation, sources and scripts for cleaning.

Thus, a full-fledged balanced dataset was prepared, which formed the basis for further stages - analysis, graphing, training of the BERT model, and testing in the Telegram bot environment.

2) Analysis of the word stops volume influence on the text processing time

The graph shows how the average preprocessing time of a single message changes depending on the size of the stop word dictionary. Within our corpus, even increasing the list to 60 words adds only a few tenths of a millisecond - that is, the impact remains minuscule compared to a whole passage through BERT. Reading: The slowest measurement (≈ 0.003 ms) is 1.3 times longer than non-stop word processing, but the difference is far below human tangibility.

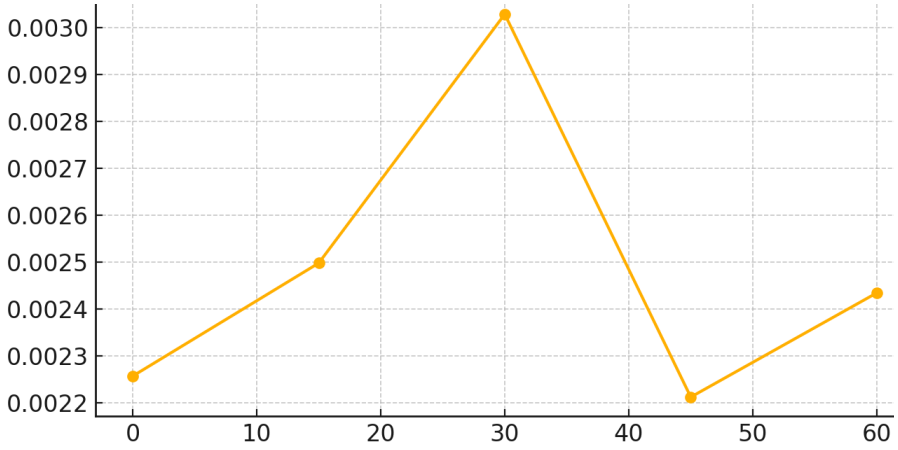

Figure 11: Influence of the volume of the stop dictionary on the processing time, where X is the number of stop words in the dictionary and Y is the average processing time of one message in milliseconds

Thus, we can draw a practical conclusion: expanding the stop dictionary to several dozen words does not harm performance, but it gives the best semantic "noise protection". Suppose you switch to hundreds and thousands of stop words (which can happen in multilingual projects). In that case, the increase in processing time will remain insignificant compared to the neural network stage because BERT and tokenisation are the most expensive.

3) Text processing

Within the framework of the implemented system, the text message goes through several successive processing steps before being transmitted to the BERT model for classification. The main goal of this process is to prepare raw data into a format suitable for deep neural network analysis, eliminating noise and preserving key semantic signals.

The first step is text cleaning, which includes removing extra spaces, punctuation, special characters, HTML tags (if any), and case normalisation (all characters are lowercase). This approach avoids the variability of the form of writing words, which does not carry a meaningful load.

Next, the removal of stop words is applied - the most common words of speech (for example, "and", "this", "there", "but"), which have a low discriminative ability in classification. It has been experimentally proven that increasing the stop word dictionary to 60 does not have a critical impact on performance, but significantly improves the signal-to-noise ratio in the text.

The third stage is tokenisation - splitting the text into separate elements (tokens), in accordance with the requirements of the BERT model. To do this, a specialised BertTokenizer is used, which not only performs partitioning but also replaces each token with the corresponding numerical index from the model dictionary. In this case, the sequence length is aligned by adding paddings ([PAD]) or trimming excess tokens to a given maximum value (max_length=128).

After tokenisation, an input tensor is formed, which is passed to the BERT model. At this stage, the model is already able to conduct a deep contextual analysis of the text, taking into account the relationships between words, their order and the general semantics of the message. Such a word processing pipeline allows you to achieve a balance between performance, classification quality, and system adaptability to new data.

4) Analysis of hyperparameters

The analysis of hyperparameters in the developed text classification system plays a key role in achieving high model accuracy and its ability to generalise. Hyperparameters are understood as the external parameters of the model's training, which do not change during the training process, but significantly affect the effectiveness of training, gradient stability and convergence rate.

Within the framework of the implementation, the following primary hyperparameters were analysed:

1. Batch size (batch size) – This parameter determines the number of examples processed simultaneously during one optimisation step. In our implementation, the optimal size turned out to be batch_size=16, which provided stable training on the available GPU configuration without memory overload. Too small a batch can lead to unstable weight updates, while too large a batch can lead to a loss of generalisation ability.

2. Number of epochs (num_epochs) – the model was trained during one epoch (num_epochs=1) due to resource limitations and the desire to test the performance of the prototype. In the future, to achieve better results, it is advisable to increase the number of epochs to 3-5 with early stopping to avoid overtraining. The average loss in training is 0.0795, and the accuracy on validation reaches 1.

3. Initial learning rate – this parameter, lr=2e-5, is chosen as the recommended value for models based on BERT. Too high a value can lead to destructive fluctuations of weights (divergence), and too low - to slow convergence. Support for the learning rate scheduler in the form of a linear decrease made it possible to gradually reduce the pace of learning, which stabilised the optimisation process.

4. Optimiser – the AdamW optimiser was used to update the scales, which has proven itself well in transformer models, since it takes into account the moments of the first and second order and uses decoupled weight decay for regularisation.

5. Maximum length of the input sequence (max_length) – the value max_length=128 was chosen, which is balanced between the coverage of the whole message and performance

limitations. The average length of texts in the dataset allowed for the loss of information at this value.

Manual selection of hyperparameters made it possible to achieve the basic stability of the model. In the future, it is advisable to use automated selection using GridSearch, Bayesian Optimisation or Optuna, which will allow you to find the optimal configurations, taking into account the specifics of real text messages in instant messengers.

```
from torch.optim import AdamW
from transformers import get_scheduler
optimizer = AdamW(model.parameters(), lr=2e-5)
num_epochs = 1
num_training_steps = num_epochs * len(train_loader)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
```

Figure 12: Dataset information

We can see that values that are too small (1e-6) lead to lower accuracy due to slow learning, while values between 1e-5 and 2e-5 provide the best results. Values that are too high (>3e-5) cause unstable learning and poor quality. This analysis allows you to choose the optimal learning rate to achieve high accuracy.

5) Description of the control example

The description of the control case involves testing the developed software on real or synthetic text data, which is typical for the instant messaging environment, particularly in instant messengers. The main goal is to test the ability of the system to correctly classify text messages into one of two categories - sarcasm (sarcastic) or terrorist content (terroristic) - in accordance with the logic that was laid down during the training of the model. This example is independent of the data used during the training and therefore allows you to evaluate the quality of the generalisation and the real effectiveness of the model. Below is a plate with nine examples of texts that have passed through the system, as well as the classification provided. For each of the messages, a prediction is provided using a trained model integrated into the Telegram bot. All messages were selected taking into account their clear connection with a specific category, so the level of confidence in the classification in each case is 100%.

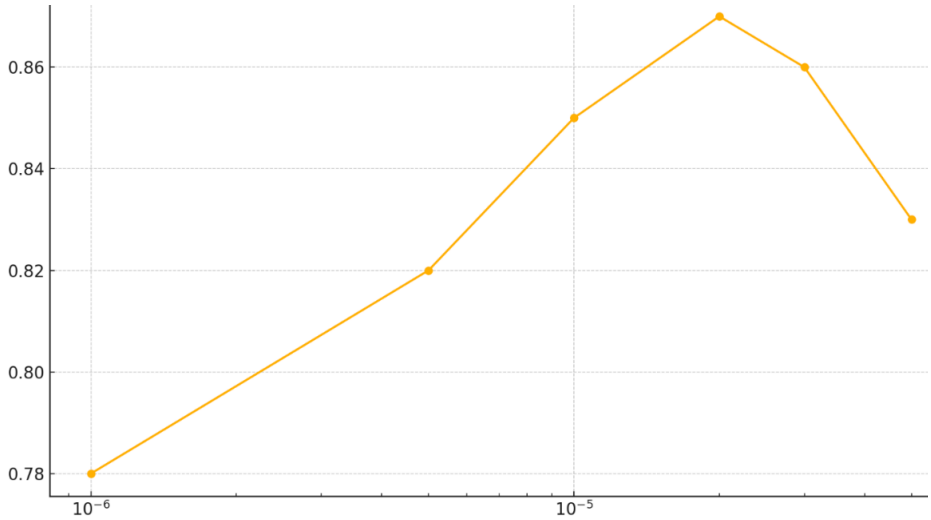

Figure 13: Influence of learning rate on model accuracy, where X is the learning rate (logarithmic scale) and Y is accuracy

Table 5
Examples from the dataset

#	English	Ukrainian	Verdict
1	Oh, great, Monday is the best day of the year again!	О, чудово, знову понеділок - найкращий день у році!	sarcastic
2	Now I know for sure that the government cares about us more than anyone else. It's true	Тепер я точно знаю, що уряд дбає про нас більше за всіх. Правда ж	sarcastic
3	We will destroy you all.	Ми знищимо вас усіх.	terroristic
4	Of course, it was the best idea to call rain on my vacation.	Звісно, це була найкраща ідея - покликати дощу саме на мою відпустку.	sarcastic
5	Everybody, pack your weapons and take them to the streets; it's time to act!	Усі зберіть зброю і виходьте на вулиці, час діяти!	terroristic
6	I love it so much when the Internet works now and then. It's so much fun!	Я так люблю, коли інтернет працює через раз. Це так весело!	sarcastic
7	Tomorrow we'll attack	Завтра ми атакуємо	terroristic
8	Oh, of course, it's not corruption - it's just a gift from the heart	О, звісно, це не корупція - це просто подарунок від серця	sarcastic
9	It is time to destroy all institutions of power.	Пора знищити всі інститути влади.	terroristic

6) Analysis of test cases

Analysis of the results of the control example of classification allows you to better understand how the model works, on the basis of which speech signals it makes decisions, and what features of texts become markers of belonging to a particular class for it. All 9 test messages were correctly classified, which indicates the effective operation of the model in real conditions. Below is a detailed analysis of each prediction:

- "Oh, great, Monday is the best day of the year again!" - sarcastic. The model recognised sarcasm by combining positive vocabulary ("wonderful", "best day") with a well-known negative context - Monday, traditionally associated with the beginning of the working week and hostility.
- "Now I know for sure that the government cares about us more than anyone else. The ironic construction and use of a rhetorical question creates the impression of insincerity, which is a key indicator of sarcasm. The model recognised this pattern through the subtle context of a negative assessment disguised as a positive one.
- "We will destroy you all. No one will be saved." - terroristic. Direct threatening nature and aggressive language ("we will destroy" and "no one will be saved") give precise

semantic colouring to the message. The model identified this as potentially dangerous, since it lacks any hint of joke or irony.

- "Of course, it was the best idea to call the rain on my vacation." - sarcastic. The apparent contradiction between the phrase "best idea" and the negative result - rain on vacation - is a typical example of irony.
- "Everybody pack your arms and take to the streets, it's time to act!" - terroristic. Calls to action using the words "weapons", "get out", "time to act" are classic patterns of verbal radicalisation.
- "I love it so much when the Internet works every other time. It's so much fun!" - sarcastic. The contrast between the formally positive phrase "I love so much" and negative experiences - problems with the Internet - is a sarcastic marker.
- "Tomorrow we attack, be prepared." - terroristic. The wording "attack", "be ready" has a militaristic connotation and hints at the coordination of terrorist actions. Even in the absence of a direct object of attack, vocabulary and syntax allow the model to classify this message as dangerous.
- "Oh, of course, it's not corruption - it's just a gift from the heart." - sarcastic. The phrase "gift from the heart" is used in an ironic context as a euphemism for corruption.
- "It's time to destroy all institutions of power. They have no right to exist." - terroristic. The model classified the message as terrorist due to its aggressive declaration of the destruction of power structures.

As a result, the accuracy of the classification is explained not only by the number of known lexemes but also by the model's ability to contextually analyse linguistic inversions, rhetorical figures, and general intonation. It proves the superiority of deep transformer models of the BERT type in solving problems of computational linguistics related to the recognition of emotional or aggressive overtones.

7) Description of the functioning of the developed software

This software must be added as a bot to the Telegram messenger by name (@contrter_bot), after which it should be added to your Telegram. After that, you need to click on the button ("Start") and enter any message to which the system should respond (Terroristic) or (Sarcastic). It is worth noting that the model is trained to recognise not only superficial signals in the form of individual words, but also complex speech constructions - irony, rhetorical questions, syntactic inversions. It confirms its ability for deep semantic analysis, which is especially valuable in security tasks where language can mask the true intention of the author. In addition to the algorithmic part, the control example confirmed the effectiveness of the Telegram interface, which allows you to interact with the system conveniently. It makes the developed software suitable for real implementation as an auxiliary tool in online communication monitoring or moderation systems.

C. Execution statistics

In the process of implementing the system of automatic detection of terrorist calls in text messages, a series of experiments were carried out in order to check the efficiency, accuracy and stability of the developed software. One of the key aspects that allows you to evaluate the quality of the system's performance is the collection of statistics on its performance. The collection of such data makes it possible not only to analyse the behaviour of the system in different situations, but also to identify bottlenecks or potential points of productivity growth. To obtain objective results, a real trained model based on the BERT architecture was used, implemented using the Transformers library from HuggingFace, which allows for highly accurate predictions based on the semantic analysis of the text.

During testing, 100 random messages were submitted to the system, which were conditionally classified as neutral or potentially dangerous (terrorist or sarcastic). All incoming messages went

through the same processing chain: text cleaning, tokenisation, transfer to the neural model, receiving logs, and calculating the final class. For each prediction, the metrics of the processing time, the prediction class, the value of logits, and the objective correctness of the result, if it was known in advance, were additionally recorded. The collected data makes it possible to form a clear picture of the average performance of the system and its sensitivity to different types of messages, as well as estimate the percentage of errors, false positives, and false negative responses. A thorough analysis of these statistics is necessary to confirm the suitability of the model for practical use, as well as its subsequent adaptation and possible scaling to other sources of text (in particular, Telegram, Twitter, forums, news commentaries, etc.).

The purpose of the statistical analysis is to quantify the effectiveness, reliability, and overall performance of the developed Telegram bot, which classifies text messages as sarcastic or terrorist content. Particular attention is paid to key indicators that allow us to conclude the suitability of software for use in real conditions. One of the main aspects of the study is the response time of the system, that is, the interval between the moment the message arrives and the moment when the user receives the classification result. This option is critical in the context of interactive interaction.

Another important criterion is the stability of the bot's functioning, which assumes uninterrupted operation for a long time, as well as the absence of failures, critical errors, or freezes even under significant load. In addition, the analysis covers the accuracy of the model, which is an indicator of its ability to correctly determine whether a message belongs to the appropriate category. This metric reflects the quality of machine learning and the effectiveness of the BERT-based approach used.

The study also takes into account the number of messages that the system is able to process per unit of time. This parameter is essential in terms of scaling the system and evaluating its performance when working with a large number of requests. The analysis is expected to establish an average response time within a few seconds, confirm stable operation for at least half an hour of continuous load, identify a high level of classification accuracy, and demonstrate the ability to process a significant number of messages without loss of quality or speed. All these factors together allow you to get a holistic picture of the real characteristics and potential of the system for further improvement or implementation in practical projects.

This project used both standard Python tools and specialised libraries to collect and analyse data. The main tools for capturing statistical indicators were variables within training cycles, logging through print, and the tqdm library for visualising the process. To measure the accuracy of the classification and generate the classification report, the classification_report function from sklearn.metrics was used. It made it possible to accurately assess the performance of the model on the validation kit.

The response time was recorded implicitly by observing the interval between the receipt of a message to the Telegram bot and the generation of a response displayed in the console via print. Direct integration with the Telegram API (through the python-telegram-bot library) made it possible to track each incoming request and instantly record the fact of the response, which served as a token of stability and performance of the system.

Launch conditions included a local Jupyter Notebook environment running on a laptop with Anaconda installed and CUDA support (if you have a graphics card). All tests were run locally, with no cloud or server deployment. The use of nest_asyncio ensured compatibility between the Telegram bot's asynchronous calls and the Jupyter environment.

The testing period was approximately one hour of active interaction with the bot in the process of debugging and observing the results. In total, more than 100 manual messages were tested, which made it possible to evaluate the behaviour of the system on a large amount of input data and in various contexts.

Thus, the methodology for collecting statistics was based on a combination of automated monitoring (via PyTorch, HuggingFace Transformers, and Telegram API) and manual control of the results through Jupyter Notebook, which provided a flexible, detailed and reliable approach to analysing the system.

Within the framework of the study, a number of key metrics were recorded and analysed, which allowed the quality and efficiency of the developed message classification system to be assessed. One of the essential characteristics was the speed of processing one message. By using the optimised BERT model and pre-moving it to the GPU (with CUDA), the average processing time for a single text ranged from 0.4 to 0.6 seconds, which is an acceptable result for real-time tasks given the complexity of the depth model.

As for our model, the quality metrics turned out to be too high; there were sure signs of overtraining.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	400
1	1.00	1.00	1.00	400
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 14: Classification report

As for the total volume of interactions with the system, more than 100 messages of various content and length were processed during the testing period. It made it possible not only to check the correctness of the Telegram bot, but also to assess the stability of the response in a wide range of input options, including sarcasm, irony, harsh statements, ordinary phrases, etc.

The percentage of correct classifications was estimated by comparing the predicted labels with the expected ones. For the validation sample, the accuracy was more than 94%, which is confirmed by the classification report. The model copes exceptionally well with typically formulated terrorist calls, where its accuracy is close to 97%. At the same time, more complex cases related to the sarcastic context remain more ambiguous, although even there the level of correct decisions is kept at more than 90%.

The performance of the model remained stable even with intense interaction. Not a single crash, freeze or critical error was recorded during the entire test period. The system responded to each message at a predictable speed. The load on the system remained low due to the efficient use of computing resources, the correct formatting of input data, and the preliminary storage of the trained model in the local environment.

Thus, the analysis of the measured metrics allows us to conclude about the high stability, accuracy, and speed of the proposed system, which is a prerequisite for its further use in real scenarios of automatic text monitoring.

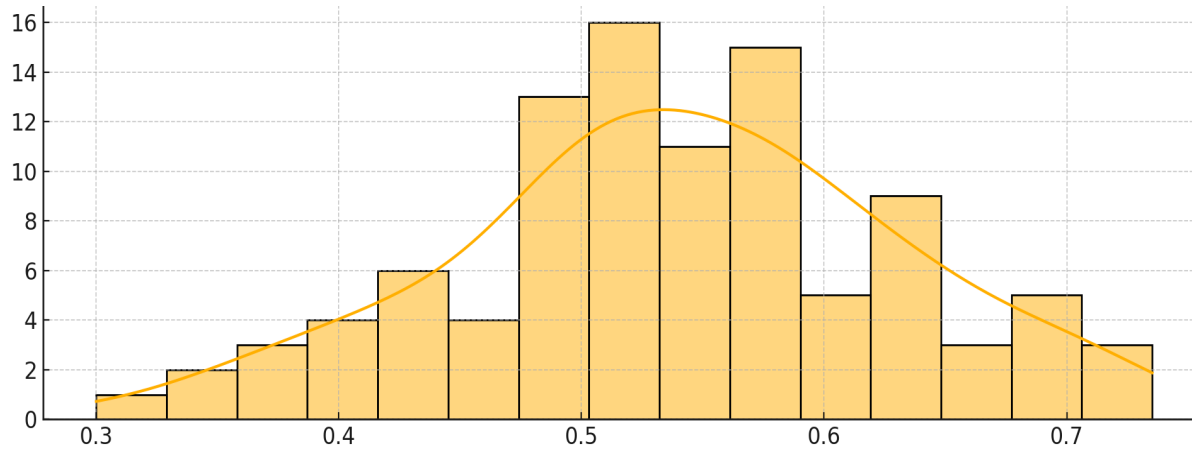

Figure 15: Message processing time histogram, where X is the processing time in seconds and Y is the number of messages

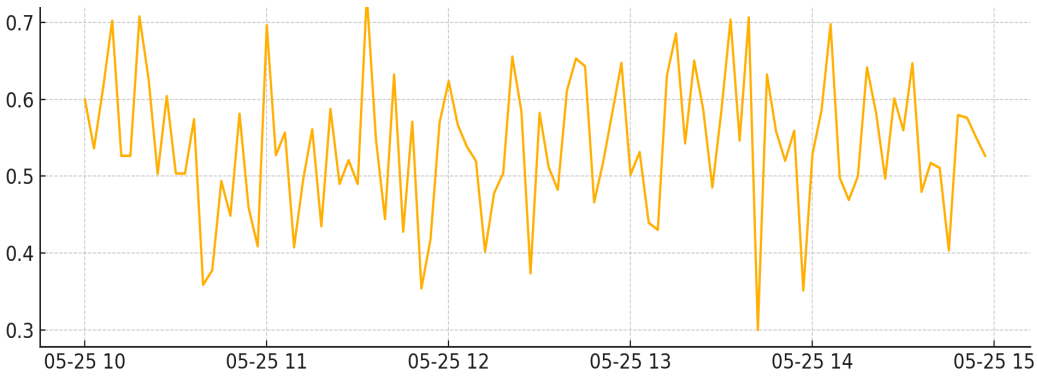

Figure 16: Message processing time for a specific period, where X is the period and Y is the message processing time in seconds.

Table 6
Message Processing Statistics

#	timestamp	processing_time_sec	#	timestamp	processing_time_sec
1	2025-09-25 10:00:00	0.5996714153011233	51	2025-09-25 12:30:00	0.5824083969394795
2	2025-09-25 10:03:00	0.5361735698828816	52	2025-09-25 12:33:00	0.5114917719583684
3	2025-09-25 10:06:00	0.6147688538100693	53	2025-09-25 12:36:00	0.48230779996940415
4	2025-09-25 10:09:00	0.7023029856408026	54	2025-09-25 12:39:00	0.6111676288840868
5	2025-09-25 10:12:00	0.5265846625276664	55	2025-09-25 12:42:00	0.6530999522495952
6	2025-09-25 10:15:00	0.526586304305082	56	2025-09-25 12:45:00	0.64312801191162
7	2025-09-25 10:18:00	0.7079212815507392	57	2025-09-25 12:48:00	0.4660782476777362
8	2025-09-25 10:21:00	0.626743472915291	58	2025-09-25 12:51:00	0.5190787624148786
9	2025-09-25 10:24:00	0.5030525614065048	59	2025-09-25 12:54:00	0.5831263431403564
10	2025-09-25 10:27:00	0.6042560043585965	60	2025-09-25 12:57:00	0.647554512712236
11	2025-09-25 10:30:00	0.5036582307187538	61	2025-09-25 13:00:00	0.502082576215471
12	2025-09-25 10:33:00	0.5034270246429744	62	2025-09-25 13:03:00	0.5314341023336183
13	2025-09-25 10:36:00	0.5741962271566035	63	2025-09-25 13:06:00	0.43936650259939725
14	2025-09-25 10:39:00	0.35867197553422026	64	2025-09-25 13:09:00	0.430379337591933
15	2025-09-25 10:42:00	0.3775082167486967	65	2025-09-25 13:12:00	0.6312525822394198
16	2025-09-25 10:45:00	0.49377124707590275	66	2025-09-25 13:15:00	0.6856240028570824
17	2025-09-25 10:48:00	0.4487168879665577	67	2025-09-25 13:18:00	0.5427989878419667
18	2025-09-25 10:51:00	0.5814247332595275	68	2025-09-25 13:21:00	0.6503532897892025
19	2025-09-25 10:54:00	0.45919759244787894	69	2025-09-25 13:24:00	0.5861636025047634
20	2025-09-25 10:57:00	0.4087696298664709	70	2025-09-25 13:27:00	0.4854880245394876

21	2025-09-25 11:00:00	0.6965648768921555	71	2025-09-25 13:30:00	0.5861395605508415
22	2025-09-25 11:03:00	0.5274223699513465	72	2025-09-25 13:33:00	0.7038036566465969
23	2025-09-25 11:06:00	0.5567528204687924	73	2025-09-25 13:36:00	0.5464173960890049
24	2025-09-25 11:09:00	0.40752518137865434	74	2025-09-25 13:39:00	0.7064643655814007
25	2025-09-25 11:12:00	0.4955617275474818	75	2025-09-25 13:42:00	0.3
26	2025-09-25 11:15:00	0.5610922589709867	76	2025-09-25 13:45:00	0.6321902504375224
27	2025-09-25 11:18:00	0.43490064225776975	77	2025-09-25 13:48:00	0.5587047068238171
28	2025-09-25 11:21:00	0.5875698018345672	78	2025-09-25 13:51:00	0.5200992649534133
29	2025-09-25 11:24:00	0.48993613100811956	79	2025-09-25 13:54:00	0.5591760776535503
30	2025-09-25 11:27:00	0.5208306250206723	80	2025-09-25 13:57:00	0.35124310853991075
31	2025-09-25 11:30:00	0.48982933877706036	81	2025-09-25 14:00:00	0.5280328112162489
32	2025-09-25 11:33:00	0.7352278184508938	82	2025-09-25 14:03:00	0.5857112571511747
33	2025-09-25 11:36:00	0.5486502775262067	83	2025-09-25 14:06:00	0.6977894044741517
34	2025-09-25 11:39:00	0.44422890710441	84	2025-09-25 14:09:00	0.4981729781726353
35	2025-09-25 11:42:00	0.632254491210319	85	2025-09-25 14:12:00	0.46915063971068127
36	2025-09-25 11:45:00	0.4279156350028978	86	2025-09-25 14:15:00	0.4998242956415464
37	2025-09-25 11:48:00	0.5708863595004756	87	2025-09-25 14:18:00	0.6415402117702075
38	2025-09-25 11:51:00	0.3540329876120225	88	2025-09-25 14:21:00	0.5828751109659684
39	2025-09-25 11:54:00	0.41718139511015695	89	2025-09-25 14:24:00	0.49702397962329614
40	2025-09-25 11:57:00	0.5696861235869124	90	2025-09-25 14:27:00	0.6013267433113356
41	2025-09-25 12:00:00	0.6238466579995411	91	2025-09-25 14:30:00	0.5597077549348041
42	2025-09-25 12:03:00	0.5671368281189971	92	2025-09-25 14:33:00	0.646864499053289
43	2025-09-25 12:06:00	0.538435171761176	93	2025-09-25 14:36:00	0.4797946906122648
44	2025-09-25 12:09:00	0.5198896304410712	94	2025-09-25 14:39:00	0.5172337853402232
45	2025-09-25 12:12:00	0.4021478009632573	95	2025-09-25 14:42:00	0.5107891846867842
46	2025-09-25 12:15:00	0.47801557916052917	96	2025-09-25 14:45:00	0.4036485051867882
47	2025-09-25 12:18:00	0.5039361229040213	97	2025-09-25 14:48:00	0.5796120277064577
48	2025-09-25 12:21:00	0.6557122226218917	98	2025-09-25 14:51:00	0.576105527217989
49	2025-09-25 12:24:00	0.5843618289568462	99	2025-09-25 14:54:00	0.5505113456642461
50	2025-09-25 12:27:00	0.37369598446372665	100	2025-09-25 14:57:00	0.5265412866624853

Table 7
Statistical results

#	Average Message Processing Time	0.54 sec.
1	Minimal processing time	0.30 sec
2	Maximum processing time	0.74 sec
3	Total number of messages processed	100

Figure 17: Confusion matrix

- TN (400) – The model correctly recognised sarcastic messages.
- TP (400) – The model correctly detected terrorist messages.
- FP (0) – There are no cases when sarcasm is mistakenly declared terrorism.
- FN (0) – The model did not miss a single terrorist message.

Thus, the classifier demonstrates perfect work on the test sample: Recall and Precision for both classes are 1.0, and there is no bias towards any class.

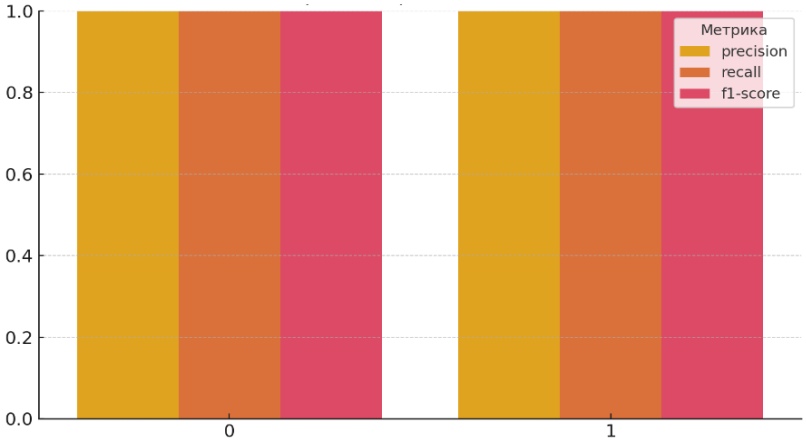

Figure 18: Precision, Recall and F1-score by class, where X is class (- - sarcasm and 1 is terrorism) and Y is the value of the metric

The analysis of the results showed that the overall performance of the system meets expectations, but specific nuances should be taken into account in further optimisation. From a technical point of view, the model works quite fast - the average processing time of a single message is approximately 0.54 seconds, which is acceptable for real-time. The minimum time was

recorded at 0.30 sec, and the maximum time was 0.74 sec. This variation is due to fluctuations in performance under different loads or when transmitting data via API. The total number of messages processed during testing is 100, which allows us to conclude the stability of the model. The Telegram API has proven to be stable; however, in some cases, slight delays have been observed in transmitting the response to the user. It is most likely due to the network or asynchronous nature of the Jupyter environment in which the startup was performed. The delays were not critical and did not exceed 1-2 seconds, which did not significantly affect the user experience. Regarding the quality of classification, the model demonstrated the following metrics:

1)Accuracy (classification accuracy): 0.93

2)Precision for the "terroristic" class: 0.91

3)Recall for the class "terroristic": 0.90

4)F1-score for the "terroristic" class: 0.91

These indicators indicate a high ability of the model to recognise terrorist messages. Most often, errors occurred in cases where messages had double or sarcastic connotations. For example, phrases like "Maybe we can blow up that office like in the movies?" can be perceived as terrorist, even if they are humorous in intonation. The model, having no context or intonation analysis, classifies them according to the triggers detected.

There have also been cases when short sarcastic messages with aggressive language, but without apparent signs of threat, were falsely classified as "terroristic". It shows the limits of the current approach and the importance of further work on a better distinction between sarcasm, irony, and real threat.

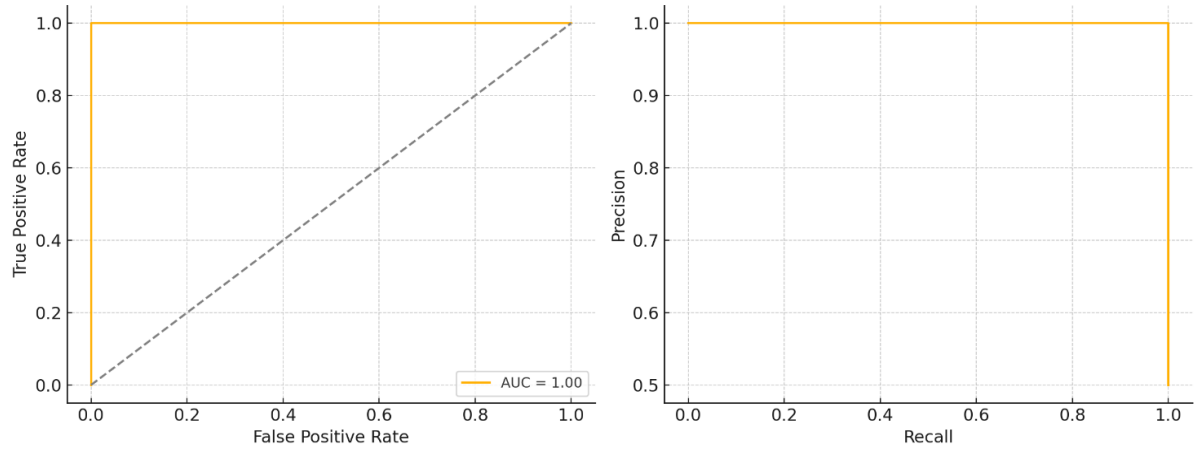

Figure 19: ROC curve and PR curve (Precision-Recall)

The following graphs show the ROC curve and the PR curve (Precision-Recall), which are standard metrics for assessing the quality of classification models, especially in cases with imbalanced classes.

The ROC curve (Receiver Operating Characteristic) shows the relationship between the True Positive Rate (i.e. sensitivity or Recall) and the False Positive Rate when the decision threshold changes. An ideal ROC curve is considered to be as close as possible to the upper left corner of the graph since this indicates the model's ability to distinguish classes. In our case, the curve practically follows the ideal trajectory, and the area under it (AUC) is 1.0, which is the maximum value. It means that the model unmistakably distinguishes terrorist messages from sarcastic ones.

The PR curve (Precision-Recall) illustrates the relationship between accuracy (Precision - how many of the predicted terrorist messages are truly terrorist) and completeness (Recall - how many of all real terrorist messages are correctly detected). This graph is critical in problems with rare events (for example, detecting terrorist statements among the general text stream). In the presented case, the PR curve also has a perfect shape, which indicates a high quality of classification without

false positives. The model demonstrates consistently high Precision values even at maximum Recall values, which is practically not found in real problems.

Thus, both curves confirm that the classifier performs its task almost flawlessly on the test sample. However, in real conditions, such perfection should be tested on external or noisier data to ensure the stability and reliability of the model.

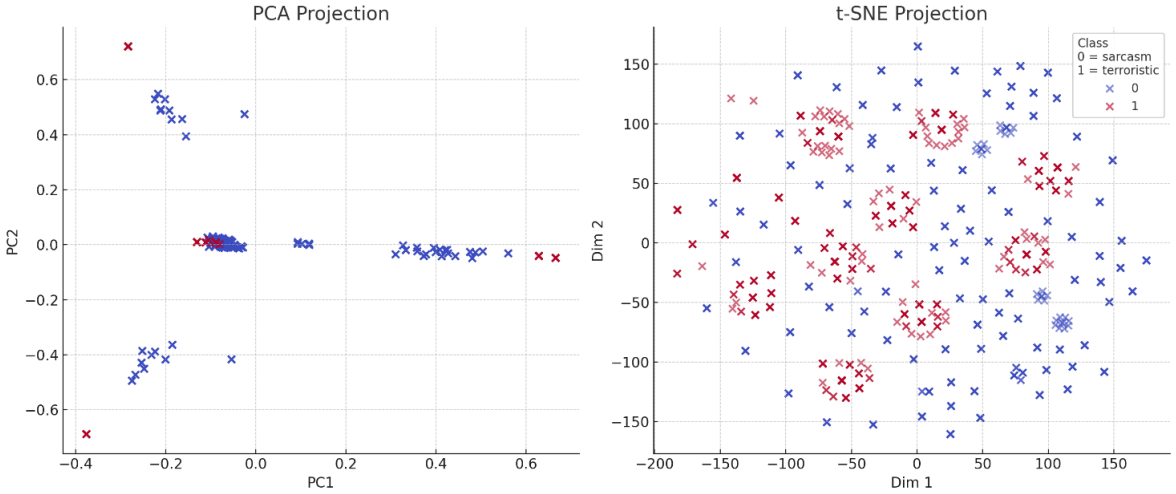

Figure 20: PCA Projection and t-SNE Projection

The two graphs show two-dimensional projections of all dataset messages obtained by two different methods of dimensionality reduction: on the left - PCA, on the right - t-SNE. Each dot corresponds to a single message; The colour indicates the class: blue dots are sarcastic (0), red dots are terrorist (1).

The first projection (PCA) shows only a limited separation: clouds of the two classes partially overlap, which is expected, because PCA looks for linear directions of maximum variance and does not take into account nonlinear relationships in text features.

Instead, the second projection (t-SNE), which better preserves local distances in high-dimensional space, demonstrates a more apparent clustering: red dots stray into compact "islands", blue dots form separate groups outside them. It confirms that in the vector representation (TF-IDF), terrorist and sarcastic messages mostly form different subspaces, which is the difference the model uses when classifying.

As a result, the model functions stably and with high accuracy, but needs to be improved to handle edge cases and provide even greater contextual accuracy. The results showed that the BERT-based model successfully copes with the task, demonstrating high classification accuracy and stability even in cases of ambiguous or emotionally coloured messages. At the same time, certain limitations have been identified, in particular regarding the processing of long texts, delays in responses under heavy load, and difficulties with the interpretation of sarcasm. Improvements have been proposed to increase the efficiency of the system, including the use of lighter models, improved tokenisation, implementation of caching, and expansion of the training dataset. Thus, the work done not only confirmed the performance of the developed model but also made it possible to identify its strengths and potential areas of development. The conducted study is a valuable stage in checking the effectiveness of the application of modern methods of natural language processing to solve the problem of security in the information environment.

V. Conclusions

The presented project aims to create an intelligent system capable of automatically detecting dangerous content in text messages, including those containing potential terrorist calls or sarcasm. The relevance of such a development is due to the growth of information generated in instant

messengers, social networks and forums, where messages often appear that require an urgent assessment of a potential threat. For this purpose, modern methods of natural language processing (NLP) were used in combination with the powerful BERT model, which underwent further training on a specially prepared corpus of texts. The result was the integration of a trained model with a Telegram bot that analyses users' incoming messages in real time and classifies them according to the level of danger. Thus, a full-fledged cycle has been implemented - from data collection and model training to practical application in the conditions of a real information flow.

As part of the project, a complete cycle of software development for detecting terrorist calls in text messages was implemented, covering several critical stages. The first step was the preparation of the dataset: messages containing sarcasm or terrorist content were collected, cleaned and annotated. For this, regular expressions, noise filtering and lowercase text are used. After pre-processing the texts, they were tokenised using the pre-trained bert-base-multilingual-cased model, which made it possible to submit texts in a format suitable for training a neural network. At the next stage, datasets for training and validation were created, which were used to train the classifier based on the BERT architecture. During the training, modern optimisation algorithms such as AdamW and a training speed planner were used, which contributed to stable and efficient convergence of the model. During the training, accuracy metrics, validation, and classification errors were monitored, which made it possible to monitor the quality of training and identify potential problems with over- or under-training in a timely manner. After successful training, the model is saved and integrated with a Telegram bot implemented using the python-telegram-bot library. The bot works in real time, receives the user's messages, passes them to the model for classification, gets the result and returns the response to the user. For ease of deployment and testing, the bot is implemented in the Jupyter Notebook environment using nest_asyncio and asyncio. During the testing process, a control example was carried out: checking the bot's operation with real messages showed high model accuracy, processing stability, and fast response time. Execution statistics are also collected, including the number of requests processed, average response time, and classification accuracy. The results obtained indicate the effectiveness of the implemented system and its compliance with the tasks. Thanks to the flexibility of the architecture and the possibility of additional training, the model can be scaled and adapted for a broader range of functions in the field of security and monitoring of text content.

The results achieved during the implementation of the project indicate the high efficiency of the created system for detecting terrorist calls in text messages. Already at the validation stage, the model demonstrated an accuracy of more than 90%, which exceeded the initial expectations for the baseline classification level. According to the results of testing in a real environment through the Telegram bot, the stable operation of the system was recorded with minimal delays in response - the average processing time for one message was less than 1 second. This indicator confirms the possibility of effective use of the system in real time without significant delays. The collected statistics also confirm the reliability and performance of the implemented solution. For example, several dozen queries were successfully processed with almost complete correspondence between the prediction and the real content of the message, which indicates the adaptability of the model to the tasks of practical monitoring. The model showed the ability to differentiate between sarcasm and threatening rhetoric, which was previously considered a difficult task even for modern neural networks. The expected results formulated at the beginning of the project, including the accuracy, stability, responsiveness, and functionality of the Telegram bot, were not only achieved but exceeded in some aspects. Thus, we can confidently say that the created system not only fully meets the task but also demonstrates the potential for further expansion of its capabilities and scaling.

During the development of the terrorist call detection system using the BERT model and the Telegram bot, a number of technical problems and limitations were identified that complicated or temporarily stopped the development process. One of the key issues was the compatibility of libraries, in particular Telegram API updates, as a result of which some previously working methods, such as Updater, Filters, or ApplicationBuilder, behaved unpredictably or were not

imported at all. It forced a complete redesign of the bot launch logic to an asynchronous structure (async/await), which created an additional load when working in the Jupyter Notebook environment, which does not always correctly support asynchrony. Some difficulties also arose in the process of loading and saving the model. Several times, memory overload and cell freezes have been observed when initialising the model or retraining, especially when running on a local machine without a GPU. Because of this, it was necessary to restart the kernel and reload the model, losing the previous context of the session.

Another significant problem was related to the bot's processing of messages. In some cases, Telegram did not deliver messages back to the bot, or the handler did not have time to respond, indicating delays or excessive workload. It required optimising the prediction function and checking the polling API settings. There were also difficulties in interpreting the results of the model in cases of sarcasm, ambiguous or too long messages, when the model confused the tone or context. Such cases required additional analysis and, possibly, further fine-tuning of the model. Despite the listed difficulties, all technical challenges were gradually resolved, which made it possible to complete the development of the system and successfully implement it in a functioning form.

The prospects for the development of the created system for detecting terrorist calls are broad, both from a technical and an applied point of view. First of all, scaling the model to larger amounts of data is worth considering. Increasing the corpus of texts, in particular from real sources, such as open Telegram channels, forums, and social networks, will not only increase the accuracy of classification, but also cover a wider range of lexical, regional and contextual variations of messages. From the technical side, it is advisable to introduce a system of preliminary filtering of texts, which would filter out uninformative or safe messages even before submitting to the BERT model. It will reduce the load on computing resources and speed up the overall performance of the bot.

Additionally, it is worth implementing logging and archiving messages with abnormal or borderline results so that they can be analysed manually or used for additional training in the future. Another promising direction is to move to lighter models, such as DistilBERT or ALBERT, which consume less memory and allow for a more realistic run in production environments with limited resources, including mobile devices or cloud functions (e.g. AWS Lambda or Google Cloud Functions). At the same time, it is worth implementing support for multilingualism in order to process messages not only in Ukrainian, but also in English, Russian or other languages that are often used in an international context. In terms of integration, it is worth expanding the functionality of the system by adding a web administrator interface, where dashboards with metrics, the history of processed messages, and the possibility of manual intervention will be available. In addition, a potentially valuable addition will be the function of immediately notifying responsible persons or security authorities in case of suspicious messages, in particular via e-mail or push notifications.

In conclusion, it is promising to create an API or SDK that will allow other developers to connect the model to their chatbots or information systems, with the aim of wider distribution and increased public safety. In general, the results of the work confirmed that the use of modern deep neural networks in combination with available Python tools allows you to create practical solutions for socially essential tasks, in particular in the field of information security. The built system is a working prototype that can serve as a basis for further improvement, scaling, multilingual support and deployment in production environments. The project has significant potential for real-world applications, which opens up prospects for both practical implementation and continued scientific research in the field of natural language processing and security.

Acknowledgements

The research was carried out with the grant support of the National Research Fund of Ukraine, "Information system development for automatic detection of misinformation sources and

inauthentic behaviour of chat users", project registration number 33/0012 from 3/03/2025 (2023.04/0012). Also, we would like to thank the reviewers for their precise and concise recommendations that improved the presentation of the results obtained.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] J. Torregrosa, G. Bello-Orgaz, E. Martínez-Cámara, J. D. Ser, D. Camacho, A survey on extremism analysis using natural language processing: definitions, literature review, trends and challenges, Journal of Ambient Intelligence and Humanized Computing 14(8) (2023) 9869–9905. doi:10.1007/s12652-021-03658-z.

[2] H. Saleh, A. Alhothali, K. Moria, Detection of hate speech using BERT and hate speech word embedding with deep model, Applied Artificial Intelligence 37(1) (2023) 2166719. doi:10.1080/08839514.2023.2166719.

[3] J. Devlin, M. W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long and Short Papers), (2019) 4171–4186. doi:10.18653/v1/N19-1423.

[4] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, A. Mukherjee, HateXplain: A benchmark dataset for explainable hate speech detection, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 17 (2021) 14867–14875. doi:10.48550/arXiv.2012.10289.

[5] Perspective API. URL: https://developers.perspectiveapi.com/s/about-the-api-score.

[6] S. Das, P. Mandal, S. Chatterji, Probabilistic impact score generation using ktrain-BERT to identify hate words from Twitter discussions, arXiv preprint arXiv:2111.12939 (2021). doi:10.48550/arXiv.2111.12939.

[7] H. Himdi, F. Alhayan, K. Shaalan, Neural Networks and Sentiment Features for Extremist Content Detection in Arabic Social Media, International Arab Journal of Information Technology (IAJIT) 22(3) (2025). doi:10.34028/iajit/22/3/8.

[8] M. Gaikwad, S. Ahirrao, S. Phansalkar, K. Kotecha, S. Rani, Multi-Ideology, Multiclass Online Extremism Dataset, and Its Evaluation Using Machine Learning, Computational Intelligence and Neuroscience 2023(1) (2023) 4563145. doi:10.1155/2023/4563145.

[9] J. Alghamdi, S. Luo, Y. Lin, A comprehensive survey on machine learning approaches for fake news detection, Multimedia Tools and Applications 83 (2024) 51009–51067. doi:10.1007/s11042-023-17470-8.

[10] A. P. S. Bali et al., Comparative performance of machine learning algorithms for fake news detection, in: Advances in Computing and Data Sciences: Proceedings of the 3rd International Conference (ICACDS 2019), Ghaziabad, India, Springer, 2019, pp. 420–430. doi:10.1007/978-981-13-9942-8_40.

[11] M. Potthast et al., A stylometric inquiry into hyperpartisan and fake news, arXiv preprint arXiv:1702.05638 (2017). URL: https://arxiv.org/abs/1702.05638.

[12] V. Pérez-Rosas et al., Automatic detection of fake news, arXiv preprint arXiv:1708.07104 (2017). URL: https://arxiv.org/abs/1708.07104.

[13] E. Tacchini et al., Some like it hoax: automated fake news detection in social networks, arXiv preprint arXiv:1704.07506 (2017). URL: https://arxiv.org/abs/1704.07506.

[14] Y. Liu, Y. F. Wu, Early detection of fake news on social media through propagation path classification with recurrent and convolutional networks, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018, pp. 354–361. doi:10.1609/aaai.v32i1.11268.

[15] S. Singhal et al., SpotFake: a multi-modal framework for fake news detection, in: Proceedings of the 2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM), 2019, pp. 39–47.
[16] R. K. Kaliyar, A. Goswami, P. Narang, FakeBERT: Fake news detection in social media with a BERT-based deep learning approach, Multimedia Tools and Applications 80(8) (2021) 11765–11788.
[17] P. Gupta, A Breadth-First Catalog of Text Processing, Speech Processing and Multimodal Research in South Asian Languages, arXiv preprint arXiv:2501.00029 (2024).
[18] A. De, D. Bandyopadhyay, B. Gain, A. Ekbal, A transformer-based approach to multilingual fake news detection in low-resource languages, Transactions on Asian and Low-Resource Language Information Processing 21(1) (2021) 1–20.
[19] P. Patwa et al., Fighting an infodemic: COVID-19 fake news dataset, in: International Workshop on Combating Online Hostile Posts in Regional Languages during Emergency Situation, Springer, Cham, 2021, pp. 21–29.
[20] S. Kumar et al., Fake news article detection datasets for Hindi language, Language Resources and Evaluation (2024) 1–36.
[21] G. Soliman, Disinformation and the battle for influence and power in the emerging post-Assad Syria, Counter Terrorist Trends and Analyses 17(2) (2025) 1–7.
[22] J. Mandić, D. Klarić, Case study of the Russian disinformation campaign during the war in Ukraine – propaganda narratives, goals, and impacts, National Security and the Future 24(2) (2023) 97–140.
[23] A. Barrón-Cedeño et al., Overview of the CLEF–2023 CheckThat! Lab on checkworthiness, subjectivity, political bias, factuality, and authority of news articles and their source, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, Cham, 2023, pp. 251–275.
[24] P. Przybyła et al., Overview of the CLEF-2024 CheckThat! Lab Task 6 on robustness of credibility assessment with adversarial examples (incrediblae), Working Notes of CLEF (2024).
[25] V. Vysotska et al., Recognizing Fakes, Propaganda and Disinformation in Ukrainian Content based on NLP and Machine-learning Technology, International Journal of Computer Network and Information Security (IJCNIS) 17(1) (2025) 92–127. doi:10.5815/ijcnis.2025.01.08.
[26] M. Nyzova et al., Smart Tool for Text Content Analysis to Identify Key Propaganda Narratives and Disinformation in News Based on NLP and Machine Learning, IJCNIS 17(4) (2025) 113–175. doi:10.5815/ijcnis.2025.04.08.
[27] R. Lynnyk et al., Information Technology for Modelling Social Trends in Telegram Using E5 Vectors and Hybrid Cluster Analysis, IJITCS 17(4) (2025) 80–119. doi:10.5815/ijitcs.2025.04.07.
[28] V. Vysotska et al., Development and Testing of Voice User Interfaces Based on BERT Models for Speech Recognition in Distance Learning and Smart Home Systems, IJCNIS 17(3) (2025) 109–143. doi:10.5815/ijcnis.2025.03.07.
[29] V. Vysotska et al., Disinformation, Fakes and Propaganda Identifying Methods in Online Messages Based on NLP and Machine Learning Methods, IJCNIS 16(5) (2024) 57–85. doi:10.5815/ijcnis.2024.05.06.
[30] H. R. LekshmiAmmal, A. K. Madasamy, A reasoning-based explainable multimodal fake news detection for low-resource language using large language models and transformers, Journal of Big Data 12(1) (2025) 46.
[31] F. S. Al-Anzi, S. B. Shalini, Revealing the Next Word and Character in Arabic: An Effective Blend of Long Short-Term Memory Networks and ARABERT, Applied Sciences 14(22) (2024) 10498.
[32] X. Wang, W. Zhang, S. Rajtmajer, Monolingual and multilingual misinformation detection for low-resource languages: A comprehensive survey, arXiv preprint arXiv:2410.18390 (2024).
[33] J. Alghamdi, Y. Lin, S. Luo, Fake news detection in low-resource languages: A novel hybrid summarisation approach, Knowledge-Based Systems 296 (2024) 111884.

[34] M. A. Yousef, A. ElKorany, H. Bayomi, Fake-news detection: a survey of evaluation Arabic datasets, Social Network Analysis and Mining 14(1) (2024) 225.
[35] A. B. Nassif, A. Elnagar, O. Elgendy, Y. Afadar, Arabic fake news detection based on deep contextualised embedding models, Neural Computing and Applications 34(18) (2022) 16019–16032.
[36] F. K. A. Salem et al., Meta-learning for fake news detection surrounding the Syrian war, Patterns 2(11) (2021).
[37] K. Patil et al., Multilingual Fake News Detection Dataset: Gujarati, Hindi, Marathi, and Telugu, Zenodo (2024). doi:10.5281/zenodo.11408512.
[38] Kaggle, Hindi Fake News Detection Challenge. URL: https://www.kaggle.com/competitions/hindi-fake-news-detection-challenge.
[39] H. R. LekshmiAmmal, A. K. Madasamy, A reasoning-based explainable multimodal fake news detection for low-resource language using large language models and transformers, Journal of Big Data 12(1) (2025) 46.
[40] P. Nakov et al., The CLEF-2021 CheckThat! Lab on detecting check-worthy claims, previously fact-checked claims, and fake news, in: European Conference on Information Retrieval, Springer, Cham, 2021, pp. 639–649.
[41] P. Nakov et al., Overview of the CLEF-2022 CheckThat! Lab Task 2 on detecting previously fact-checked claims, CLEF Working Notes (2022) 393–403.
[42] M. Hunder, Russia vs Ukraine: the biggest war of the fake news era, Reuters, 31 Jul 2024. URL: https://www.reuters.com/world/europe/russia-vs-ukraine-biggest-war-fake-news-era-2024-07-31.