

**LLM-BASED AUTOMATIC DISCOVERY OF OT AND IT
NETWORKS**

FIELD OF THE PRESENTLY DISCLOSED SUBJECT MATTER

The presently disclosed subject matter relates to computer systems and methods
5 of network discovery management and cybersecurity.

BACKGROUND

Network discovery is a process dedicated for the identification and understanding
of the various assets that constitute networks, and particularly operational technology
(OT) networks and Information Technology (IT) networks.

10 By applying network discovery in an organization, valuable information regarding
the organization can be obtained, including the identification, profiling, and mapping of
the different assets which constitute the OT and IT networks, and the identification of
connectivity and communication pathways between these assets.

Through network discovery, organizations can improve their understanding of the
15 network infrastructure, assess its security status, detect potential vulnerabilities, develop
risk mitigation plans, and implement suitable security measures to protect their
operational technology assets against malicious cyber attacks, safeguarding their
reliability and availability, and increasing the overall security posture of the network.

GENERAL DESCRIPTION

20 Network discovery in operational technology (OT) and information technology (IT)
networks is a challenging task due to technical complexities, manual labor, vendor
diversity, and software version variations. Traditionally, network administrators
undertake labor-intensive efforts to manually identify and document each asset within
the network.

25 In OT environments, the presence of legacy systems, proprietary protocols, and
heterogeneous architectures further complicate network discovery. The lack of

standardized documentation and the dependence on vendor-specific tools often impede efficient and accurate identification of assets. Additionally, frequent software updates and varying versions further hinder the network discovery process.

5 In IT networks, the dynamic nature of the infrastructure, constant addition of new devices, and frequent changes in software configurations, present additional challenges. The sheer scale of IT networks, distributed across multiple sites or subnets, further exacerbates the difficulty of asset discovery. Manual inventory processes become labor-intensive and time-consuming, making it challenging to maintain an up-to-date and accurate view of the network assets.

10 The existing solutions for network discovery often rely on a combination of network scans, manual inventory audits, and vendor-specific tools. These methods are resource-intensive, error-prone, and time-consuming. They require significant human effort, specialized knowledge, installation of software or physical hardware within the network, and ongoing coordination with different vendors and software versions.

15 The presently disclosed subject matter is related to an innovative approach that addresses these challenges and streamlines the network discovery process in both OT and IT networks. The suggested approach considerably reduces reliance on manual labor, vendor-specific tools, and the complexities associated with different software versions, while ensuring accurate, efficient, passive, and rapid network visibility and asset
20 identification.

The presently disclosed subject matter includes a computer system and method for linguistic passive automatic discovery of operational technology (OT) and information technology (IT) networks. The discovery can be applied on networks of various sizes, including large networks comprising many assets (e.g., tens of thousands of endpoints
25 and servers or more), as well as smaller networks such as subnets, multi sites, or even sites. In some examples, boundaries of a network (defining which assets belong to a specific network) are defined by a router.

The computer systems and methods disclosed herein utilize a novel large language

model (LLM) trained to comprehensively "understand" configuration files, notwithstanding the fact that configuration data stored in configuration files is not a natural language. Configuration files are special types of files that store information defining various assets and their interactions in an OT or IT network. Each configuration file describes a respective network asset. Assets include, for example, various devices, and particularly network infrastructure components such as switches, routers, gateways, firewalls, sensors, etc. Each configuration file contains information (referred to herein as "configuration data") of a respective asset (device).

The information in configuration files includes for example:

- Information on the assets (or "entities") in the network, including for example, data such as asset class (e.g., hardware, software), asset type (e.g., computer, router, switch, hub, application, operating system, etc.), IP addresses, hostname, network settings, access control, protocols, applications, etc.
- Information on the network topology, including, for example, data on physical and logical relations and links between different assets, the links bandwidth, etc.
- Rules and policies implemented by the different assets, including, for example, network security configurations and policies, cybersecurity parameters, routing protocol, protocol specific configuration, connectivity rules (e.g., which assets can be connected), etc.

The LLM incorporates Machine Learning techniques and is trained, using a large dataset of configuration files, to extract from configuration files in a network (OT and/or IT networks) relevant configuration data.

According to some examples, the configuration files used during training include synthetic configuration files generated by an innovative approach. The synthetic files can be generated by virtually producing numerous possible network architectures. This approach ensures a diverse and extensive training dataset, enhancing the accuracy and effectiveness of the LLM in understanding and extracting data from real-world

configuration files.

According to further examples disclosed herein, the data retrieved from the configuration files is stored in a graph data-structure, which enables easy and efficient processing, manipulation, and retrieval of the data, including comprehensive asset
5 management, risk assessment, and proactive threat hunting.

According to a first aspect of the presently disclosed subject matter there is provided a computer-implemented method of network discovery, comprising:

collecting configuration files from at least one network, each configuration file comprising configuration data that characterizes different assets in the at least one
10 network and their respective connections and properties;

applying on the configuration files a large language model (LLM) that is trained to provide, as output, to at least one received instruction ("prompt"), a respective response comprising specific configuration data extracted from the configuration files; wherein the LLM is capable of retrieving configuration data pertaining to multiple assets and their
15 respective connections in the at least one network, the retrieved configuration data enabling the generating of an ontology of the network, including, for example, information on the assets, information on network topology, and information on rules and policies.

In addition to the above features, the method according to this aspect of the
20 presently disclosed subject matter can optionally comprise one or more of features (i) to (xv) below, in any technically possible combination or permutation:

- I. using the LLM for generating, for each asset and/or connection in the network, a respective data object, wherein all data-objects are characterized by a uniform data-object format (e.g., JSON object).
- 25 II. generating a graph representing the retrieved configuration data, wherein each data-object of a respective asset retrieved from the configuration files is designated as a node in the graph, and each connection retrieved from the configuration files is designated as a vertex in the graph connecting between two

nodes, and wherein each node or vertex is stored with its respective properties retrieved from a respective configuration file;

- III. applying the LLM for determining a type of configuration file, and, responsive to determining a configuration file of a certain type, automatically generating a regular expression and/or a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data;
- IV. wherein the LLM includes a transformer model that provides a next token in a sequence based on previous tokens in the sequence;
- V. comparing configuration data in a respective response of one or more instructions to configuration data in the configuration files, and, in case of a discrepancy, discarding the respective response, and re-processing the one or more instructions;
- VI. generating a training dataset for training the LLM, comprising generating synthetic configuration files;
wherein the synthetic configuration files are generated by a process that utilizes network design software, comprising:
 - automatically manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and
 - executing the network designs to thereby obtain a respective diverse collection of configuration files;
- VII. wherein automatic manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining a configuration file that contains errors;
- VIII. wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and human annotated configuration files;
- IX. wherein generating the training dataset for training the LLM comprises providing,

as input, multiple instruction-response pairs, and training the LLM to provide, to a given instruction, a respective response comprising specific configuration data extracted from the configuration files;

- x. wherein the multiple instruction-response pairs include instruction-Regex
5 pairs used for training the LLM to, responsive to ascertaining that a configuration file is of a certain type, automatically generate a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.
- xi. wherein the multiple instruction-response pairs include instruction-software code
10 pairs used for training the LLM to, responsive to ascertaining that a configuration file is of a certain type, automatically generate a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data;
- xii. wherein the at least one network is an operation technology network and/or an
15 information technology network;
- xiii. wherein the trained LLM includes 'n' embedded tokens, each generated from a respective token extracted from the configuration files, the method comprising:
applying in the LLM a selective attention mechanism, when providing a response to a received instruction, comprising:
20 selecting a subset of 'm' from 'n' embedded tokens and using only the embedded tokens in the subset when applying the attention mechanism.
- xiv. The method further comprises:
for at least one embedded token generated from the received instruction,
calculating a dot product score using the subset of 'm' embedded tokens,
25 giving rise to n/m dot product scores;
selecting from the subset, a group of k embedded tokens with the highest respective dot product scores;
expanding each of 'k' embedded tokens with a plurality of additional 'p' adjacent tokens, giving rise to a final collection of $\frac{n}{m} + k * p$ embedded tokens;

applying a normalization function (e.g., SoftMax function) on respective dot product scores of the final collection of embedded tokens, thereby obtaining a respective set of weights, and applying the weights for obtaining a contextualized vector for the at least one embedded token.

- 5 xv. The method further comprising using a tokenizer specifically configured for tokenizing the configuration files and feeding the tokenized data to the LLM.

The presently disclosed subject matter further contemplates a computer system comprising at least one processing circuitry configured to execute a method according to the first aspect disclosed above.

- 10 The presently disclosed subject matter further contemplates a computer program product comprising a computer readable storage medium retaining a program of instructions, which, when read by a computer processor, causes the computer processor to perform a method according to the first aspect disclosed above.

- 15 The presently disclosed subject matter further contemplates a non-transitory program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform a method according to the first aspect disclosed above.

- 20 The system, the computer program product, and the non-transitory program storage device, disclosed in accordance with the presently disclosed subject matter, can optionally comprise one or more of features (i) to (xv) listed above, mutatis mutandis, in any technically possible combination or permutation.

- 25 The presently disclosed subject matter further contemplates a computer-program product operable in a computer and recorded on a computer-readable medium for (linguistic) automatic discovery of at least one network, wherein the computer-program product is produced by processes that include:

obtaining a training dataset that includes at least a collection of many configuration files;

training a large language model (LLM) using the configuration files to thereby obtain a trained LLM that provides, as output, to at least one

received instruction ("prompt"), a respective response comprising specific configuration data extracted from the configuration files.

The presently disclosed subject matter further contemplates a method of training a large language model (LLM) to provide, as output, to at least one received instruction ("prompt"), a respective response comprising specific configuration data extracted from configuration files; wherein the LLM is capable of retrieving configuration data pertaining to multiple assets and their respective connections in the at least one network, and generating an ontology of the network according to the retrieved configuration data.

The presently disclosed subject matter further contemplates a computer-product operable in a computer and recorded on a computer-readable medium comprising a large language model (LLM) for automatic discovery of at least one network, wherein the computer-product can be applied on a collection of configuration files obtained from at least one network and provide, as output, configuration data extracted from the configuration files that can be used for generating an ontology of the at least one network;

wherein the computer-product is produced by a process that includes: obtaining a training dataset that includes at least a collection of network configuration files;

training a large language model (LLM) using the network configuration files to thereby obtain a trained LLM that provides as output, to at least one received instruction, a respective response comprising specific configuration data extracted from the configuration files.

The computer program product can optionally comprise one or more of features (i) to (vii) listed below, mutatis mutandis, in any technically possible combination or permutation:

i. Wherein obtaining the training dataset comprises generating a training dataset for training the LLM that comprises synthetic configuration files.

ii. Wherein the process further includes generating the synthetic configuration files comprising:

manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of configuration files.

5 iii. Wherein manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining configuration file that contain errors.

iv. Wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and
10 human annotated configuration files.

v. Wherein generating the training dataset comprises providing, as input, multiple instruction-response pairs, and training the LLM to provide, in response to a given instruction, a respective response comprising specific configuration data extracted from the configuration files.

15 vi. Wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to determine, responsive to determining that a configuration file is of a certain type, and to automatically generate a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

20 vii. Wherein the multiple instruction-response pairs include instruction-software code pairs used for training the LLM to, responsive to ascertaining that a configuration file is of a certain type, automatically generate a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

25 The presently disclosed subject matter further contemplates a computer-implemented method of training a large language model (LLM) for automatic discovery of at least one network, wherein LLM can be applied on a collection of configuration files obtained from at least one network and provide, as output, configuration data extracted from the configuration files; wherein the training comprises: generating synthetic

configuration files and adding the synthetic configuration files to the training dataset.

The computer-implemented method comprising applying a specialized tokenizer on the configuration files is configured to use unique delimiters tailored to the syntax and structure of configuration files.

- 5 The computer-implemented method of training can optionally comprise one or more of features (i) to (vii) listed above with respect to the computer-program product, mutatis mutandis, in any technically possible combination or permutation.

BRIEF DESCRIPTION OF THE DRAWINGS

- 10 In order to understand the presently disclosed subject matter and to see how it may be carried out in practice, the subject matter will now be described, by way of non-limiting examples only, with reference to the accompanying drawings, in which:

Fig. 1 is a high-level block diagram schematically illustrating a computer system **100** dedicated for automatic network discovery, in accordance with an example of the presently disclosed subject matter;

15

Fig. 2 is a high-level flowchart showing operations carried out by system **100**, in accordance with an example of the presently disclosed subject matter;

Fig. 3 is a block diagram schematically illustrating data preparation computer and LLM training computer, in accordance with an example of the presently disclosed subject matter;

20

Fig. 4 is a flowchart of operations carried out during the data preparation process and training process, in accordance with an example of the presently disclosed subject matter;

Fig. 5 is a block diagram schematically illustrating LLM execution in a computer, in accordance with an example of the presently disclosed subject matter;

25

Fig. 6 is a flowchart of operations carried out during the automatic discovery process, in accordance with an example of the presently disclosed subject matter;

Fig. 7 is a schematic illustration of a graph representing the output of an automatic discovery process, in accordance with an example of the presently disclosed subject matter; and

Fig. 8 is a flowchart of operations carried out as part of a selective attention mechanism, in accordance with an example of the presently disclosed subject matter.

DETAILED DESCRIPTION

In the drawings and descriptions set forth, identical reference numerals indicate those components that are common to different embodiments or configurations. Elements in the drawings are not necessarily drawn to scale.

Unless specifically stated otherwise, as apparent from the following discussions, it is appreciated that, throughout the specification, discussions utilizing terms such as "collecting", "generating", "calculating", "determining", "applying", "performing", "assigning" or the like, include an action and/or processes of a computer that manipulate and/or transform data into other data, said data represented as physical quantities, e.g. such as electronic quantities, and/or said data representing the physical objects.

The terms "computer", "computer system", "computer device", "computerized device" or the like, should be expansively construed to include any kind of hardware-based electronic device with a data processing circuitry (e.g., digital signal processor (DSP), graphics processing unit (GPU), tensor processing unit (TPU), field programmable gate array (FPGA), application specific integrated circuit (ASIC), microcontroller, microprocessor etc.). The processing circuitry can comprise, for example, one or more processors operatively connected to computer memory, loaded with executable instructions for executing operations, as further described below.

The operations in accordance with the teachings herein may be performed by a computer specially constructed for the desired purposes, or by a general-purpose computer specially configured for the desired purpose by a computer program stored in

a computer readable storage medium.

As used herein, the phrase "for example," "such as", "for instance" and variants thereof, describe non-limiting embodiments of the presently disclosed subject matter. Reference in the specification to "one case", "some cases", "other cases", or variants thereof, means that a particular feature, structure, or characteristic described in connection with the embodiment(s), is included in at least one embodiment of the presently disclosed subject matter. Thus, the appearance of the phrase "one case", "some cases", "other cases", or variants thereof, does not necessarily refer to the same embodiment(s).

It is appreciated that certain features of the presently disclosed subject matter, which are, for clarity, described in the context of separate embodiments, may also be provided in combination in a single embodiment. Conversely, various features of the presently disclosed subject matter, which are, for brevity, described in the context of a single embodiment, may also be provided separately, or in any suitable sub-combination.

In embodiments of the presently disclosed subject matter, fewer, more and/or different stages than those shown in **Figs. 2, 4, 6, and 8** may be executed. In embodiments of the presently disclosed subject matter, one or more stages illustrated in the figures may be executed in a different order, and/or one or more groups of stages may be executed simultaneously.

Figs. 1, 3, and 5 illustrate a general schematic of the system architecture in accordance with certain examples of the presently disclosed subject matter. Elements in **Figs. 1, 3, and 5** can be made up of any combination of software and hardware and/or firmware that performs the functions as defined and explained herein. Elements in **Figs. 1, 3, and 5** may be centralized in one location or dispersed over more than one location. For example, each one of computers **110, 120, 130, and 140** can be located at a different geographical location, remote from the other computers. Furthermore, in some examples of the presently disclosed subject matter, the system may comprise fewer, more, and/or different elements than those shown in **Figs. 1, 3, and 5**. For example, **Figs. 1, 3, and 5**

show several separate computers, each dedicated for certain functions of the system, however it should be clear to any person skilled in the art that the functionalities can be divided otherwise, for instance in an alternative system-design the components of computers **110** and **120** can be consolidated in a single computer device, and/or
5 computers **130** and **140** can be consolidated in a single computer device.

Bearing the above in mind, attention is drawn to Fig. **1**, which is high-level schematic illustration of a computer system dedicated for linguistic automatic network-discovery as disclosed herein. Fig. **1** is a general example which demonstrates some principles of the presently disclosed subject matter. Components of system **100** shown in
10 Fig. **1** can be connected over any type of communication network, including, for example, any one of the following: the Internet, a local area network (LAN), wide area network (WAN), metropolitan area network (MAN), any type of telephone network (including, for example, PSTN with DSL technology) or mobile network (including for example 4G or 5G mobile communication technologies), or any combination thereof.

By way of non-limiting example system **100** is shown to include several computers, each configured to perform a certain general task, including data preparation computer **110**, LLM training computer **120**, LLM execution computer **130**, and graph generation and data retrieval computer **140**. Fig. **2** is a flowchart showing a high-level view of various processes carried out by system **100**, according to some examples of the presently
20 disclosed subject matter. The processes shown in Fig. **2** are described below in conjunction with the components of Fig. **1**.

At block **21**, a data preparation process is carried out (e.g., by data preparation computer **110**). During the data preparation process special training data dedicated specifically for training an LLM for the purpose of (linguistic) automatic network discovery
25 is assembled into a comprehensive training dataset.

At block **23** the LLM is trained (e.g., by LLM training computer **120**) using the special training data and by implementing a training process specifically designed for training the LLM for (linguistic) automatic network discovery.

At block **25** the (linguistic) automatic network discovery process is carried out using the trained LLM (e.g., by LLM execution computer **130**). The LLM is executed on configuration files obtained from actual OT/IT networks for automatically extracting the relevant configuration data from the files and generating the network discovery results (e.g., network ontology).

As further described below in some examples, the LLM output is stored in a graph data-structure, for easy and efficient data manipulation, retrieval, and display (block **27**; e.g., by graph generation and data retrieval computer **145**). The LLM output (i.e., the network discovery results) generated by the LLM are processed to generate the graph data-structure representing the LLM output. Generation of the graph can be performed for example by LLM execution computer **130** or by graph analysis and data retrieval computer **140**.

Following generation of the graph, various tools dedicated for querying and retrieval of information from the graph can be used for this purpose (block **29**). In some examples, graph generation and data retrieval computer **140** can be configured with data analytics tools that allow users to interact with the generated graph and obtain the desired information.

Turning to Fig. **3**, it shows a more detailed view of data preparation computer **110** and LLM training computer **120**, according to some examples of the presently disclosed subject matter. Fig. **4** is a flowchart showing operations carried out as part of the data preparation process and LLM training, and is described, by way of example, with reference to components of computers **110** and **120**.

At block **41** the training dataset required for training the LLM is prepared. As mentioned above, the data used for training is especially configured for the unique task of generating an LLM dedicated for automatic network discovery. The LLM is trained to comprehensively "understand" configuration files, identify from the configuration files information on the respective assets, and extract from the configuration files relevant data, including for example: network topology, device configurations, access controls,

cybersecurity parameters, and more.

In some examples, data preparation computer **110** comprises training data collector **111** configured to collect and arrange a training dataset for training the model. In general, the training dataset comprises a large corpus of configuration files (e.g.,
5 between tens of thousands and up to millions or more), as well as additional training data generated based on network configuration files.

As mentioned above, configuration files exhibit significant diversity. Configuration files which are used by different vendors, in different releases of the same vendors and in different networks, differ significantly in many aspects, including their content, structure,
10 and syntax. Thus, to obtain a robust LLM which is capable of performing accurate and efficient network discovery in different networks, each operating with different configuration files, the training corpus must include a large variety of configuration files that represent this diversity.

However, configuration files are generally a proprietary asset used by vendors for
15 network device configuration, and therefore availability of the configuration files is naturally limited. Furthermore, configuration files are constantly updated and adapted to accommodate updates in devices, software, standards, and network topology, and therefore existing configuration files do not provide a full picture of all possible variations in configuration files. Yet further, the errors, inconsistencies, and human annotations,
20 which appear in configuration files, can be also diverse. Therefore, even should availability of configuration files be much more abundant, the variation in configuration files would still be a limiting factor in the generation of a training dataset.

To bridge this gap a novel technique has been developed by the inventors for generating a configuration files dataset with high diversity. According to some examples
25 data preparation computer **110** (e.g., using data collector **111**) obtains configuration files from three different sources, including, proprietary configuration files, human annotated configuration files, and synthetic configuration files.

- Proprietary configuration files are files used by vendors for configuring devices

in OT/IT networks. These files are collected from various vendors and added to the collection of configuration files in the training dataset.

- Human annotated files are configurations which are manually modified by human users to thereby increase variability of the configuration files. Importantly, human annotators can insert "human contrived" modification which may scarcely appear in proprietary configuration files, and thereby enrich the variability of the configuration files in the training dataset. This may include non-default configuration parameters (which, in most cases, are left in their default settings by network and cybersecurity managers), and manually inserted human errors and/or comments.
- Synthetic configuration files.

The presently disclosed subject matter contemplates a computer system, a computer implemented method, and a non-transitory program storage device loaded with computer instructions, all dedicated for the synthesis of configuration files (block **43**). According to some examples, data preparation computer **110** comprises data synthesis module **112** configured for synthesizing configuration files, as further discussed below. In other examples, data synthesis module **112** can be implemented in a different computer, e.g., in a dedicated processing circuitry configured to generate the synthetic files and deliver them to data preparation computer **110**.

According to one example, configuration files synthesis is performed by computer-based automatic manipulation of network design software output. Network design software tools are types of software programs which are used for designing a network (e.g., Graphic Network Simulator 3 (GNS3)). Once a network design is finalized, part of the output are respective configuration files of the newly created network. According to the presently disclosed subject matter, a computer implemented method is disclosed which is configured to interface with the network design software and automatically manipulate the design parameters to thereby provide a large variety of automatically generated design instructions. By executing the design instructions and generating the respective network topologies, the respective configuration files of the different topologies are

obtained. Manipulation of the design parameters includes, for example, adding/removing devices, changing the device types of existing devices, increasing/decreasing the size of the network, inserting various alternative software types, inserting various alternative security and access settings, inserting various alternative types and definitions of
5 connections between assets in the network, etc.

Manipulation of the design parameters may further include the intentional insertion of errors into the network design. This may include, for example, inserting undefined values, gibberish, incorrect values, empty files, empty fields, overflowing files, misplaced comments, etc. This creates configuration files that contain a variety of errors
10 and enables to train the LLM to cope with such errors that may be found in configuration files during network discovery of actual networks and thus further increase robustness of the LLM.

The generation of synthetic configuration files can be carried out by a software (implemented for example by data synthesis module **112**) that combines fuzzers (aka
15 "fuzz testing tools"), obfuscators, and generative approaches. Fuzzers are used for intentionally inserting errors into the network designs. Obfuscators are used for obscuring data such as IP addresses and passwords for further complicating the network design. Generative approaches use algorithms or AI to automatically create or modify configuration files, ensuring consistency, security, and compliance with predefined
20 templates, policies, or requirements. This combination ensures the efficient generation of diverse synthetic network configuration files. The raw training dataset, including the collection of configuration files, can be stored in a dedicated computer storage device (including non-transitory computer storage, e.g., raw training dataset repository **114**).

In addition to the configuration files, the training dataset includes other data. At
25 block **43** additional types of training data are generated. The instruction layer of the LLM enables users to provide guidance to the LLM, guiding the model to generate the desired response to each different instruction (also known as "prompt"). The training dataset therefore includes many examples of a variety of pairs of instructions (or "prompts") and the respective desired response, i.e., the relevant configuration data that correctly

responds to the prompt. Notably, in the following description the terms "instruction" and "prompt" are used interchangeably.

According to some examples, various types of instructions are used, including instructions for obtaining assets (devices), instructions for obtaining connections
5 between assets, and instructions for obtaining properties of assets. The training dataset therefore includes different types of instruction-response pairs for training the model to respond to all types of instructions. Through the inclusion of different types of instruction-response pairs in the training dataset, the model gains the ability to respond effectively to all types of instructions, which in turn enables greater granularity and flexibility when
10 retrieving data from configuration files. During training, the LLM is fed with the instruction-response pairs to facilitate training of the model to provide, in response to a specific prompt, the appropriate configuration data. The instruction-response pairs are generated as part of the training dataset preparation.

According to examples of the presently disclosed subject matter, the LLM is
15 trained to provide the responses in a predefined uniform format. One non-limiting example of such a response format is a JSON (JavaScript Object Notation) object. In other examples other formats may be used, e.g., XML format. Accordingly, the responses of the instruction-response pairs are provided in the correct format, e.g., JSON format, where each response corresponding to any asset in the network is provided and stored in the
20 same predefined format. This enables to train the LLM to provide configuration data, in response to instructions, in a uniform format.

The following are examples of instructions:

"Extract all endpoints with no endpoint-detection";

"Extract groups and privileges"; and

25 "Extract inbound rules from external networks".

The response includes a JSON object for each asset, which includes various properties related to the asset. Notably, a single JSON object can be constructed from

configuration data obtained using more than one instruction.

Another type of training data which is generated as part of the training dataset preparation process are instruction-Regex pairs. Some configuration files are very large. Due to certain limitations of the transformer model, which is used as part of the LLM, it performs poorly (e.g., exhibits a dramatic slowdown during inference) when applied on very large files (e.g., greater than 8K tokens). The size of some configuration files (e.g., firewall configuration files), which is very large, may therefore impede training of the entire files concurrently. Normally this problem is solved by dividing large files into smaller chunks and processing each chunk separately. However, since contextual information in configuration files is distributed over the entire file, processing of the entire files is imperative to enable the model to correctly derive relations between the different assets in the file and learn how to correctly respond to instructions and retrieve the relevant data. For this reason, division of the file into smaller chunks, and training each chunk individually, is impractical.

A novel approach is disclosed that enables to train the LLM to process large files in a special way that alleviates the above-mentioned technical problem. This includes using regular expressions (Regex) dedicated for extracting from the configuration files the relevant configuration data. The LLM is trained to identify large files and dynamically and automatically generate an appropriate Regex configured to extract the configuration data required for the respective response.

According to some examples, during training the LLM is fed with instruction-response pairs dedicated for training the model to discern between file types. Such an instruction-response pair includes an instruction that queries the type of configuration file, and a corresponding response that indicates the respective file type.

In addition, the LLM is fed with many different examples of instruction-Regex pairs to facilitate training of the model to generate, following identification of certain configuration file types, an appropriate Regex that can extract for the configuration file the appropriate configuration data. As mentioned above, this approach is used for

processing large files such as firewall configuration files. The LLM is further trained to apply the generated Regex on the relevant configuration file and provide the relevant configuration data. The instruction-Regex pairs are generated, for example, as part of the training dataset preparation.

5 The presently disclosed subject matter further contemplates the use of computer program code dynamically and automatically generated for extracting, from the configuration files, the relevant configuration data. The LLM is trained to identify large files and automatically generate a computer code configured to extract the configuration data required for the respective response.

10 According to some examples, during training the LLM is fed with instruction-response pairs dedicated for training the model to discern between file types. Such an instruction-response pair includes an instruction that queries the type of configuration file, and a corresponding response that indicates the respective file type.

 In addition, the LLM is fed with many different examples of instruction-computer
15 code pairs to facilitate training of the model to generate, following identification of certain configuration file types, an appropriate computer code that can parse the configuration files and extract therefrom the appropriate configuration data. As before, this approach can be used for processing large files such as firewall configuration files. The LLM is further trained to dynamically apply the generated computer code on the
20 relevant configuration file and provide the relevant configuration data. The instruction-computer code pairs are generated for example, as part of the training dataset preparation.

 The use of computer code presents advantages over regular expressions for parsing and analyzing configuration files to determine their various properties. While
25 Regex is suitable for simple pattern matching, it becomes cumbersome and difficult to maintain when dealing with complex configurations and nested structures commonly found in network configuration files, such as hierarchical settings, conditional statements, and diverse syntax elements. In contrast, utilizing computer code, such as Python or

JavaScript, enables the implementation of sophisticated parsing logic capable of efficiently handling these intricacies.

Computer code offers the flexibility to adapt to different file formats, validate data, manage dependencies, and incorporate comprehensive error handling. Moreover,
5 computer code allows for the incorporation of dynamic changes, modular functions, and detailed documentation, ensuring enhanced readability and maintainability.

Additionally, computer code facilitates dynamic and conditional parsing, which is particularly advantageous when dealing with configuration files containing conditional statements or environment-specific settings. By utilizing a programming language, it is
10 possible to incorporate logic that handles these conditions and dynamically adjusts the parsing process accordingly. This adaptability is important for responding to changes in configuration formats or content without necessitating extensive rewrites of the parsing logic.

At block **45** the configuration files in the training dataset are tokenized. As
15 configuration files are not natural language, a special tokenization process was devised for this purpose. The tokenization process uses delimiters, which are different to standard tokenization delimiters, and which are adapted to the words and symbols found in different configuration files. A vocabulary of tokens can be generated and mapped to a unique index.

20 In some examples a dedicated Machine Learning model is trained to execute the tokenization. The Machine Learning model can be, for example, a supervised model that is trained with a dataset of annotated configuration files. The trained machine learning tokenizer can be used during training and execution of the LLM for tokenizing the configuration files.

25 The tokenizer plays a central role in LLMs, facilitating the conversion of text into tokens and back. It functions independently within the LLM pipeline, utilizing its own specialized training datasets and algorithms, such as Byte Pair Encoding (BPE). After undergoing training, the tokenizer is equipped to carry out two primary functions:

`encode()`, which transforms text into tokens, and `decode()`, which reverts tokens back into text. This process is central in attaining effective operation of LLMs, ensuring that the model can accurately interpret and generate language.

5 In the context of configuration files, which do not follow natural language patterns, a specialized tokenization process has been developed. This process employs unique delimiters tailored to the syntax and structure of various configuration files, differing from those used in standard text tokenization. By utilizing these customized delimiters, a vocabulary of tokens can be created, each mapped to a unique index, enabling efficient and accurate processing of configuration file contents.

10 Training a custom tokenizer for an LLM designed to interpret network configuration files is important for several reasons. Firstly, network configuration files contain highly specific syntax and terminology unique to networking protocols and device configurations. Standard tokenizers, trained on natural language data, are ill-equipped to handle this specialized content, leading to ineffective tokenization and sub-optimal model
15 performance.

Secondly, by developing a tokenizer that understands the intricacies of network configuration files, the LLM can more accurately parse and interpret the information within these files. This allows the model to convert the raw configuration data into meaningful knowledge on the network's structure, performance, and security posture.

20 Finally, a custom tokenizer ensures that the LLM can handle the wide variety of syntax and symbols found across different types of network configuration files. This adaptability is crucial for creating a robust and versatile model capable of functioning effectively in diverse networking environments. By mapping each unique token to an index, the tokenizer facilitates efficient data processing and enhances the model's ability
25 to learn from and act on network configuration data. The tokenizer is also used during inference for tokenizing configuration files which are provided as input to the trained model.

In some examples, data preparation computer **110** comprises configuration files

tokenizer **116** configured for executing the tokenization process as described above. In other examples, configuration files tokenizer **116** can be implemented in a different computer e.g., in a dedicated processing circuitry configured to receive the configuration files, tokenize the files, and transmit the tokenizer output back for further training of the LLM.

The training dataset, ready for being used for training the LLM, can be stored in a dedicated (e.g., non-transitory) computer storage device (e.g., final training dataset repository **118**). The final training dataset includes the tokenized configuration files, instruction-response pairs, and instruction-Regex pairs.

Once the training dataset is ready, it is made available for training the model (block **49**). In some examples, the training dataset is provided to the LLM hub (**122**) in LLM training computer **120**, which is configured, *inter alia*, to coordinate between various layers of the LLM training process.

According to some examples, the training process can be divided into three layers, each layer being dedicated for performing a certain part of the training process, including a fundamental layer, instruct layer, and reward and reinforcement layer. In the illustrated example, LLM training computer **120** comprises three respective modules (**124**, **126**, and **128**) operatively connected to model hub **122**, each module configured for executing operations related to one of the layers.

The fundamental layer (implemented by module **124**) is configured to receive the configuration files in the training dataset as input and to execute word embedding on the files, where the tokenized configuration files are converted into a respective sequence of word vectors (referred to herein as "embedded tokens"), which preserve their semantic relations. The fundamental layer is further configured for training a transformer model (e.g., autoregressive transformer model) which is a neural network model that learns to predict the next token in a sequence, given the previous tokens in the sequence. The transformer model also includes a self-attention mechanism that enables the model to identify relationships and dependencies between different tokens in the input and

improve next token prediction.

The instruct (or "instruction") layer (implemented by module **126**) is configured to further train the model to follow specific instructions. The instruct layer can be a separate component or integrated within the model's input pipeline, depending on the specific implementation. The instruct layer is dedicated for fine-tuning the fundamental layer to improve user interaction with the model. In a deeper sense, the instruct layer causes the weights of the neural network to change, so that the output of the model is better suited to the instruction received as input.

According to the presently disclosed subject matter, the instruct layer receives, as input, the instruction-result pairs, and learns to predict the next word in a sequence, given the instructions and the previous words, such that the appropriate result is provided to a given instruction. As explained above, the LLM is trained to provide the results in a certain predefined uniform format (e.g., JSON objects).

As explained above, according to some examples, a technical solution for processing large configuration files is provided, which includes training the model to generate regular expressions (Regex) dedicated for extracting the relevant data from large configuration files (e.g., of 8K tokens or more). According to this example, the instruct layer is trained to identify types of configuration files (e.g., based on corresponding instruction-response pairs), and in response to the identification of a large file (e.g., firewall configuration file), to automatically and dynamically generate an appropriate Regex designed to provide information according to the respective instruction. To this end the LLM is fed with multiple examples of instructions-Regex pairs, and is trained, based on these pairs, to generate the appropriate Regex.

In some examples, training computer further includes a reinforcement layer (implemented by module **128**) configured to learn from feedback, and thus further fine-tune the model and provide more accurate results, which are also, in general, more human accustomed. This fine-tuning process guides the model towards generating more desirable outputs for a target task, in this case providing more accurate information on

assets in the network during network discovery in response to a respective instruction. As with the instruct layer, the reinforcement layer also causes the weights of the neural network to change so that the output of the model is augmented.

Once the training process is complete, the trained model is made available for
5 execution (e.g., in trained LLM computer data repository).

Proceeding to Fig. 5, it shows a more detailed view of LLM execution computer
130, according to some examples of the presently disclosed subject matter. Fig. 6 is a
flowchart showing operations carried out as part of the LLM execution of an automatic
network discovery process, according to some examples, and is described, by way of
10 example, with reference to components of LLM execution computer **130**.

Assuming a certain network is being processed, at block **601** the configuration files
of the different devices in the network are retrieved from the network (e.g., by
configuration files collector **131**).

According to some examples, application of the LLM on the retrieved
15 configuration files involves conditional activation of prompts (instructions). To this end, a
set of prompts are designed, and prompt activation conditions are defined. In general, it
is desired to create a set of prompts that are specifically adapted to align with various
conditions or criteria, where each prompt is constructed to direct the LLM towards
producing the intended or desired response. In the current case, prompts that can be
20 used for obtaining different types of configuration data are generated. Prompt activation
conditions determine which prompt should be activated under different circumstances.
The conditions can be based on various factors, such as input characteristics, context, user
preferences, or intended results.

In some examples, LLM execution computer **130** includes an interactive prompt
25 generator, configured to implement prompt activation, either based on predefined
prompt activation programs or in response to real-time user defined prompt activation.
Prompt activation programs (that include the prompts and prompt conditions) can be
stored in a prompt database operatively connected to LLM execution computer **130**.

In some examples, the configuration files are assessed to determine whether any of the files is considered a large file i.e., a file that exceeds a certain size (block **603**; e.g., by file classifier **132**). Alternatively, or additionally, the configuration files are assessed to determine their type, e.g., whether any of the files is a firewall configuration file. This assessment can be done using an appropriate prompt configured to determine the type of a configuration file. According to some examples, initially an attempt is made to process the configuration file using the transformer model (with the instructions), and only if this fails, the Regex is used for extracting data from the file instead.

By applying conditional activation of prompts, the process continues according to the type of file, where, in case the file is a large file (and/or a firewall configuration file), the response involves the generation of an appropriate Regex (block **607**), which is then applied on the respective "large file" to thereby extract the relevant configuration data from the file (block **609**). For other types of files, the trained LLM (transformer model) is applied on the configuration files retrieved from the network (which are tokenized as described above) to thereby extract the relevant configuration data from the file (block **609**).

The configuration data extracted from the configuration files includes the various assets, the relations, and interconnections between the assets and properties of the assets and the connections. Examples of connections include vlan – interface; vlan – vlan; and interface – endpoint.

The configuration data extracted from the configuration files using the LLM model (with or without automatic Regex generation and application) is transformed into the predefined output format (block **611**; e.g., by data formatting module **135**). For example, the configuration data extracted from each configuration file can be represented by a respective JSON object.

In some examples, the output of the automatic network discovery is validated (block **613**; e.g., by validation module **137**). This includes processing the output data received in response to one or more instructions, and determining whether it includes

false output (e.g., AI hallucinations). To this end, the output configuration data (e.g., in the JSON object) is compared with the original configuration files, and if a discrepancy between the data generated by the LLM and the data in the original files is found, the false configuration data is removed (block **615**; e.g., by cleansing module **138**) and the instructions are sent back to be re-processed by the LLM (block **609**). In some examples, the network discovery output (e.g., validated JSON objects) is stored in a dedicated (e.g., non-transitory) computer storage device (block **617**).

According to some examples, the network discovery output is stored in a graph data-structure that logically represents the network ontology including the different devices, their properties, and the connections between the devices in the network (block **619**, e.g., by graph generator **139**).

In some examples, each device in the network is represented as a node in a graph, and relations between devices are represented by vertexes. Each node and vertex can be implemented as an object that comprises the relevant information characterizing the respective device or connection. Fig. 7 is an example of a graph generated based on the linguistic network discovery process disclosed herein. The graph shows various nodes, each representing a certain asset (e.g., router, switches, interfaces, vlans, endpoints, etc.) and their interconnection. Notably, the graph is presented for illustrative purposes and does not include all information, such as the various properties of each node or connection.

Computer **130** can be configured to execute the LLM repeatedly, e.g., periodically, or in response to a certain predefined event, to account for any changes in the network and enable to maintain an updated picture of the topology of the network.

The generated graph can be stored in an appropriate computer storage device (e.g., graph DB **140**). The graph provides a powerful tool for organization, representation, and retrieval of configuration data of the network. The graph data-structure enables administrators and cybersecurity personnel to easily identify vulnerabilities, potential attack vectors, and conduct proactive threat hunting, thereby enhancing the overall

network security.

As mentioned above, system **100** may further include graph analysis and data retrieval computer **140**, configured to enable analysis of the graph by users to deduce insights with respect to assessing its security status, detect potential vulnerabilities, develop risk mitigation plans, and more. The graph DB can be made accessible to computer **140** for this purpose. In some examples, graph generation (block **619**, where graph generator **139** is implemented as part of computer **140**) can be also executed by computer **140** upon receiving from LLM execution computer **130** the network discovery output.

The attention (or self-attention) mechanism is a key feature of the transformer model neural network architecture. The attention mechanism enables the transformer model to focus on different parts of the input sequence when determining how relevant is a given element (e.g., token) to the current context, and to generate outputs accordingly. It helps to capture long-range contextual dependencies and relationships between elements in a sequence. The attention mechanism weighs word vectors according to the context of the word and provides contextualized representation of vectors, thus pushing closer words vectors which are contextually more related than others.

As part of the attention mechanism, scores are calculated for each word vector by applying the dot product (aka "key-query dot product") between the word vectors and any other word vector (including itself, giving rise to n^2 scores). The calculated scores indicate contextual proximity between the original words. The scores are normalized, using, for example, the SoftMax function, to thereby obtain corresponding probability scores (or weights) that collectively add up to 1. The weights are then used for weighing the original word vectors and obtaining respective context vectors, which capture the importance of different word vectors according to the context.

Following the generation of the model, during inference, a decoder is implemented taking as input the embeddings of previously generated tokens and providing predictions for the next token in the sequence. The decoder utilizes self-

attention to calculate attention scores, and the attention scores are used to weight the contributions of different tokens during the generation process.

In the current application the context size is very large, often with as many as 100,000 tokens or more. This is a huge amount of data which creates a significant
5 processing bottleneck when applying the attention mechanism during inference.

According to some examples of the presently disclosed subject matter, a selective attention mechanism is applied by the decoder during inference. A selective attention mechanism is less processing intensive, while enabling a sufficiently accurate prediction by the LLM model.

10 Fig. 8 is a flowchart of operations carried out as part of a selective attention mechanism, according to some examples of the presently disclosed subject matter. Considering a certain instruction is received, the instruction is tokenized to obtain respective tokens (block 801) and word embedding is applied on each token to obtain respective vectors (herein after "embedded tokens"; block 803).

15 According to the selective attention mechanism disclosed herein, only a selection of embedded tokens in the model are used. Instead of calculating a dot product score for every embedded token in the model, a dot product is calculated using a subset of the embedded tokens. In some examples, a dot product is generated using one embedded token out of every 'm' embedded tokens in the model, such that n/m dot product
20 operations are performed for a given token in the instruction instead of $n * n$ (where n is the total number of tokens in the model; block 805). According to an example, $m=32$. Assuming 100,000 tokens are being processed, the dot product is applied on 3125 embedded tokens instead of all 100,000, for a total of 3125 dot product scores.

In the following stage, a group of 'k' embedded tokens which produced the largest
25 dot product scores are selected from the subset of n/m dot product scores (e.g., $k=64$ (block 807). One example of selecting the k tokens involves applying a SoftMax function on the n/m key-query dot product values and selecting (k) tokens with the largest key-query dot product values. By setting the selection cutoff according to key-query dot product distribution, dynamic selection is obtained.

Next, the group of k embedded tokens is expanded, by adding for each embedded token of the k embedded tokens in the group, ' p ' adjacent embedded tokens. In some examples, ' p ' equals ' m ', where in case $m = 32$, $p=32$ (block **809**). In some examples, for a given embedded token from the group of ' k ' embedded tokens, the $p/2$ (e.g., 16) embedded tokens preceding the given token and the $p/2$ embedded tokens proceeding the given token are added to the group. The resulting output is an expanded group of $k * p$ dot product scores, each calculated using a respective embedded token in the model (i.e., the final collection of embedded tokens). The final collection of $\frac{n}{m} + k * p$ dot product scores are normalized using a normalization function such as the SoftMax function (block **811**), thereby converting the scores to probability scores (weights).

Finally, the weights are used for weighting the original embedded tokens and obtaining respective context vectors, which capture the importance of different embedded tokens according to the context (block **813**).

Thus, according to the selective attention mechanism, attention is applied on only part of the tokens, while other tokens remain with their original embedded values without the additional weighting. The inventors have found that this approach, when implemented on a large collection of tokens (extracted in this case from configuration files), helps to reduce the processing intensiveness while providing sufficiently accurate contextual information which enables to train the LLM model to provide the required configuration data from the files.

It will also be understood that the system according to the presently disclosed subject matter may be a suitably programmed computer. Likewise, the presently disclosed subject matter contemplates a computer program being readable by a computer for executing the methods of the presently disclosed subject matter. The presently disclosed subject matter further contemplates a machine-readable non-transitory memory tangibly embodying a program of instructions executable by the machine for executing methods of the presently disclosed subject matter.

It is to be understood that the presently disclosed subject matter is not limited in its application to the details set forth in the description contained herein or illustrated in

the drawings. The presently disclosed subject matter is capable of other embodiments and of being practiced and carried out in various ways. Hence, it is to be understood that the phraseology and terminology employed herein are for the purpose of description and should not be regarded as limiting. As such, those skilled in the art will appreciate that

5 the conception upon which this disclosure is based may readily be utilized as a basis for designing other structures, methods, and systems for carrying out the several purposes of the present presently disclosed subject matter.

CLAIMS:

1. A computer-implemented method of network discovery, comprising:
collecting configuration files from at least one network, each configuration file comprising configuration data that characterizes different assets in the at least one
5 network and their respective connections and properties;
applying on the configuration files a large language model (LLM) trained to provide as output responsive to at least one received instruction, a respective response comprising specific configuration data extracted from the configuration files; wherein the LLM is capable of retrieving configuration data pertaining to multiple assets and their
10 respective connections in the at least one network.
2. The computer-implemented method of claim 1 comprising generating, based on the retrieved configuration data, an ontology of the at least one network including, at least part of: information on the assets, information on network topology, and information on rules and policies.
- 15 3. The computer-implemented method of claim 1 or 2 comprising:
using the LLM for generating, for each asset and/or connection in the at least one network, a respective data object, wherein all data-objects are characterized by a uniform data-object format.
- 20 4. The computer-implemented method of any one of the preceding claims comprising: generating a graph representing the retrieved configuration data, wherein each data-object of a respective asset retrieved from the retrieved configuration data is designated as a node in the graph, and each connection retrieved from the retrieved configuration data is designated as a vertex in the graph connecting between two nodes, and where each node or vertex is stored with its respective properties.
- 25 5. The computer-implemented method of any one of the preceding claims comprising:
applying the LLM on the configuration files in the at least one network for determining a type of configuration file, and, responsive to determining a configuration

file of a certain type, automatically generating a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

6. The computer-implemented method of any one of the preceding claims
5 comprising:

applying the LLM on the configuration files in the at least one network for determining a type of configuration file, and, responsive to determining a configuration file of a certain type, automatically generating a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective
10 configuration data.

7. The computer-implemented method of any of the preceding claims, wherein the LLM includes a transformer model that provides a next token in a sequence based on previous tokens in the sequence.

8. The computer-implemented method of any one of the preceding claims
15 comprising: comparing configuration data in a respective response of one or more instructions to configuration data in the configuration files, and, in case of a discrepancy, discarding the respective response and re-processing the one or more instructions by the LLM.

9. The computer-implemented method of any one of the preceding claims
20 wherein a training dataset for training the LLM comprises synthetic configuration files.

10. The computer-implemented method of claim 9, wherein the synthetic configuration files are generated by a process that utilizes network design software, comprising:

manipulating parameters of the network design software to thereby obtain a
25 diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of configuration files.

11. The computer-implemented method of claim 10, wherein manipulation of

parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining configuration files that contain errors.

12. The computer-implemented method of any one of claims 9 to 11, wherein
5 the training dataset includes an assembly of the synthetic configuration files, proprietary configuration files, and human annotated configuration files.

13. The computer-implemented method of any one of claims 1 to 12, wherein
the training dataset comprises providing, as input, multiple instruction-response pairs,
and training the LLM to provide, in response to a given instruction, a respective response
10 comprising specific configuration data extracted from the configuration files.

14. The computer-implemented method of claim 13, wherein the multiple
instruction-response pairs include instruction-Regex pairs used for training the LLM to
automatically generate, responsive to determining that a configuration file is of a certain
type, a regular expression adapted for parsing the configuration file and retrieving from
15 the configuration file the respective configuration data.

15. The computer-implemented method of claim 13, wherein the multiple
instruction-response pairs include instruction-computer code pairs used for training the
LLM to automatically generate, responsive to determining that a configuration file is of a
certain type, a computer program code adapted for parsing the configuration file and
20 retrieving from the configuration file the respective configuration data.

16. The computer-implemented method of any one of the preceding claims,
wherein the at least one network is an operation technology network and/or an
information technology network.

17. The computer-implemented method of any one of claims 1 to 16 wherein
25 the LLM includes 'n' embedded tokens, each generated from a respective token extracted
from the configuration files, the computer-implemented method comprising:
applying in the LLM a selective attention mechanism, when providing a response
to a received instruction, comprising:

selecting a subset of 'm' from the 'n' embedded tokens and using only the embedded tokens in the subset when applying the attention mechanism.

18. The computer-implemented method of claim 17, further comprising:
for at least one embedded token generated from the received instruction:
5 calculating a dot product using the subset of 'm' embedded tokens, giving rise to $\frac{n}{m}$ dot products;

selecting from the subset, a group of k embedded tokens with the highest respective dot products;

expanding each of 'k' embedded tokens with a plurality of additional 'p'
10 adjacent tokens, giving rise to a final collection of $\frac{n}{m} + k * p$ embedded tokens;
applying a normalization function on respective dot product of the final collection of embedded tokens, thereby obtaining a respective set of weights, and applying the weights for obtaining a contextualized vector for the at least one embedded token.

19. A computer program product comprising a computer readable storage
15 medium retaining a program of instructions, which, when read by a computer processor, causes the computer processor to perform a method according to any one of claims 1 to 18.

20. A non-transitory program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform a method
20 according to any one of claims 1 to 18.

21. A computer system comprising at least one processing circuitry configured to execute a method of network discovery using an LLM according to any one of claims 1 to 18.

22. A computer-program product operable in a computer and recorded on a
25 non-transitory computer-readable medium comprising a large language model (LLM) for automatic discovery of at least one network, wherein the computer-product product can be applied on a collection of configuration files obtained from at least one network, and

provide, as output, configuration data extracted from the configuration files that can be used for generating an ontology of the at least one network;

wherein the computer-product product is produced by a process that includes:

obtaining a training dataset that includes at least a collection of network
5 configuration files;

training a large language model (LLM) using the network configuration files to thereby obtain a trained LLM that provides, as output, to at least one received instruction, a respective response comprising specific configuration data extracted from the configuration files.

10 23. The computer-program product of claim 22, wherein obtaining the training dataset comprises:

generating a training dataset for training the LLM that comprises synthetic configuration files.

15 24. The computer-program product of claim 23, wherein the generating includes generating the synthetic configuration files comprising:

manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of configuration files.

20 25. The computer-program product of claim 24, wherein manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining configuration files that contain errors.

25 26. The computer-program product of any one of claims 22 to 25, wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and human annotated configuration files.

27. The computer-program product of any one of claims 22 to 26, wherein generating the training dataset comprises providing, as input, multiple instruction-response pairs, and training the LLM to provide, in response to a given instruction, a respective response comprising specific configuration data extracted from the configuration files.

28. The computer-program product of claim 27, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

29. The computer-program product of claim 27, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

30. A computer-implemented method of training a large language model (LLM) for automatic discovery of at least one network, wherein the LLM can be applied on a collection of configuration files obtained from at least one network, and provide, as output, configuration data extracted from the configuration files;

wherein the training comprises:

generating synthetic configuration files and adding the synthetic configuration files to a training dataset.

31. The computer-implemented method of claim 30 comprising generating the synthetic configuration files, comprising:

manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of

configuration files.

32. The computer-implemented method of claim 31, wherein manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining
5 configuration files that contain errors.

33. The computer-implemented method of any one of claims 30 to 32, wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and human annotated configuration files.

10 34. The computer-implemented method of any one of claims 30 to 33, wherein generating the training dataset comprises providing, as input, multiple instruction-response pairs, and training the LLM to provide, in response to a given instruction, a respective response comprising specific configuration data extracted from the configuration files.

15 35. The computer-implemented method of claim 34, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

20 36. The computer-implemented method of claim 34, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

25 37. The computer-implemented method of any one of claims 30 to 36 comprising applying a specialized tokenizer on the configuration files configured to use unique delimiters tailored to syntax and structure of configuration files.

Abstract

The presently disclosed subject matter includes a computer system and computer-implemented method for linguistic passive automatic discovery of operational technology
5 (OT) and information technology (IT) networks, which utilize a novel large language model (LLM) trained to comprehensively "understand" configuration files, notwithstanding the fact that configuration data stored in configuration files is not a natural language. The discovery can be applied on networks of various sizes, including large networks comprising many assets (e.g., tens of thousands of endpoints and servers or more), as
10 well as smaller networks such as subnets, multi sites, or even sites.

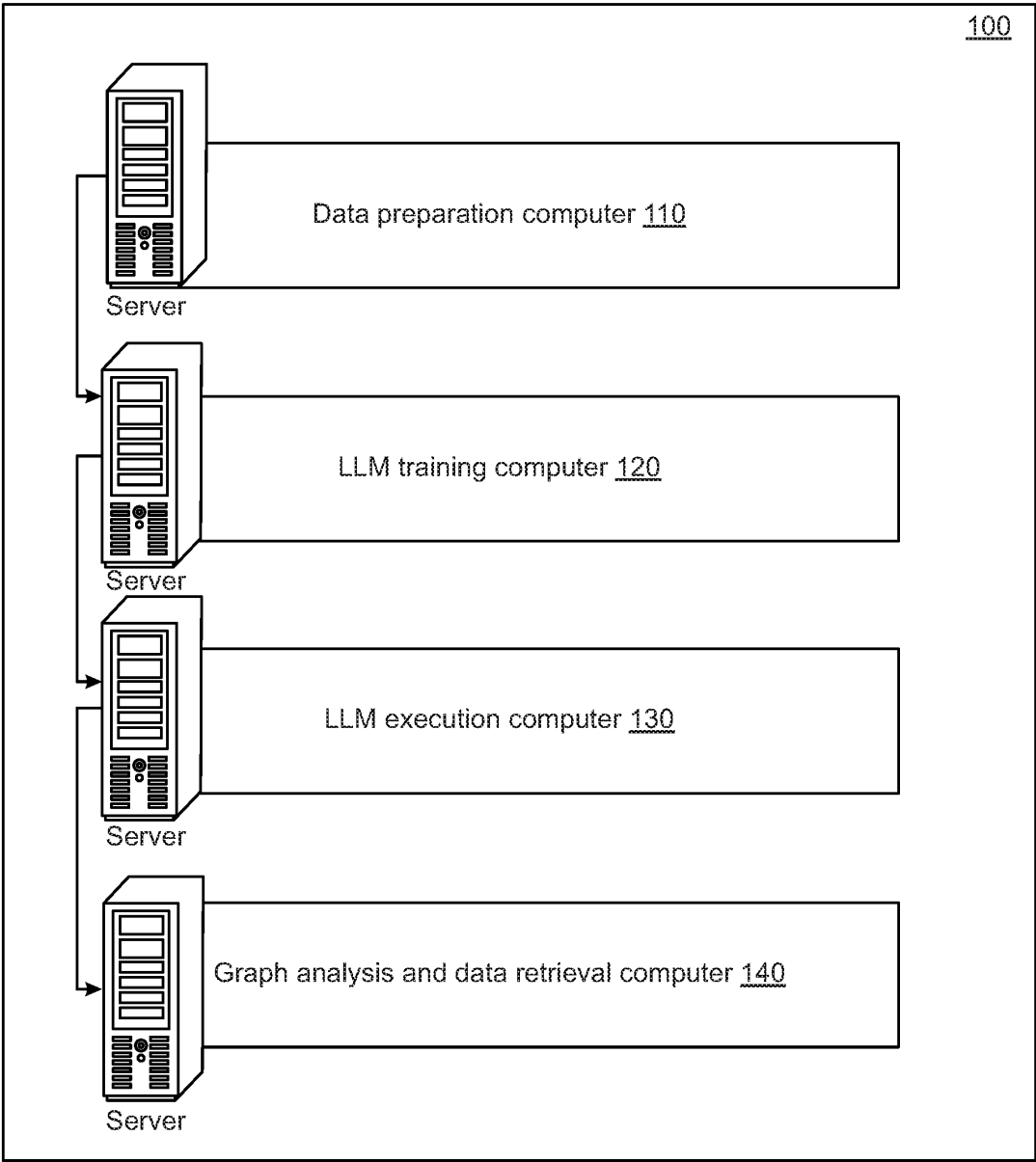


FIG. 1

2/8

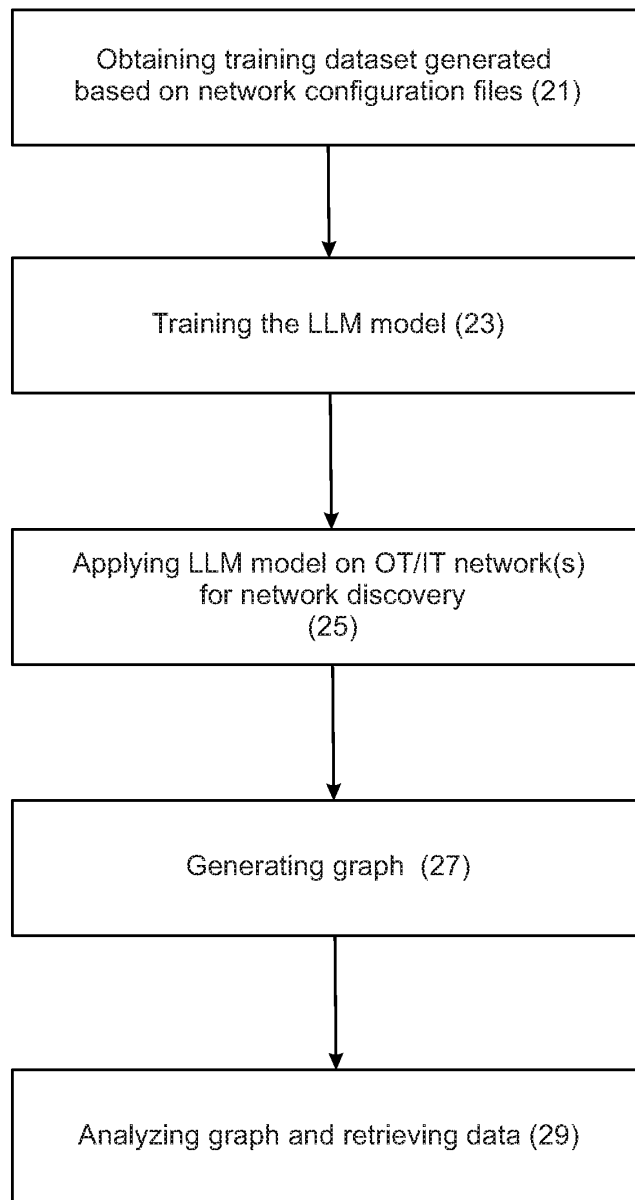


FIG. 2

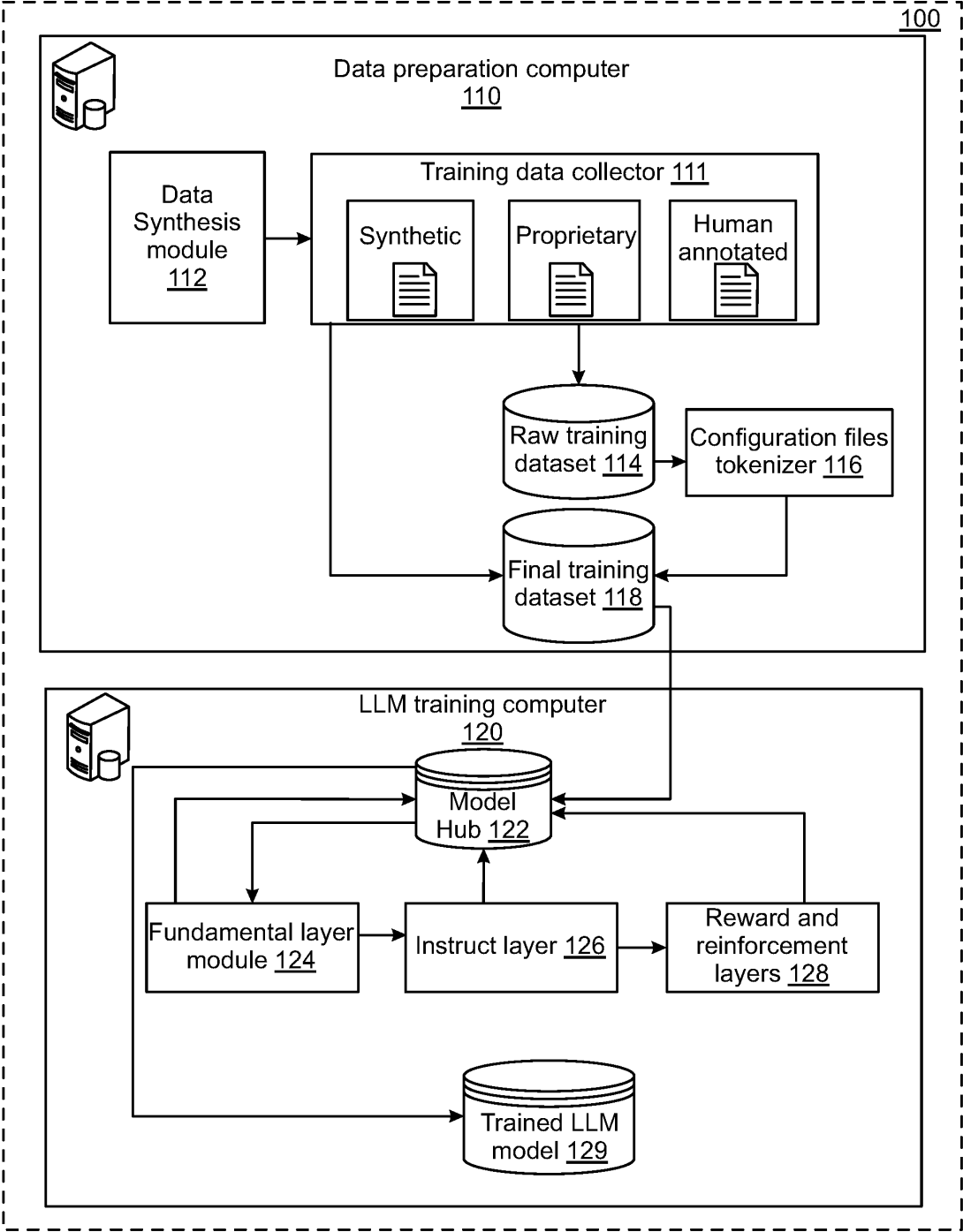


FIG. 3

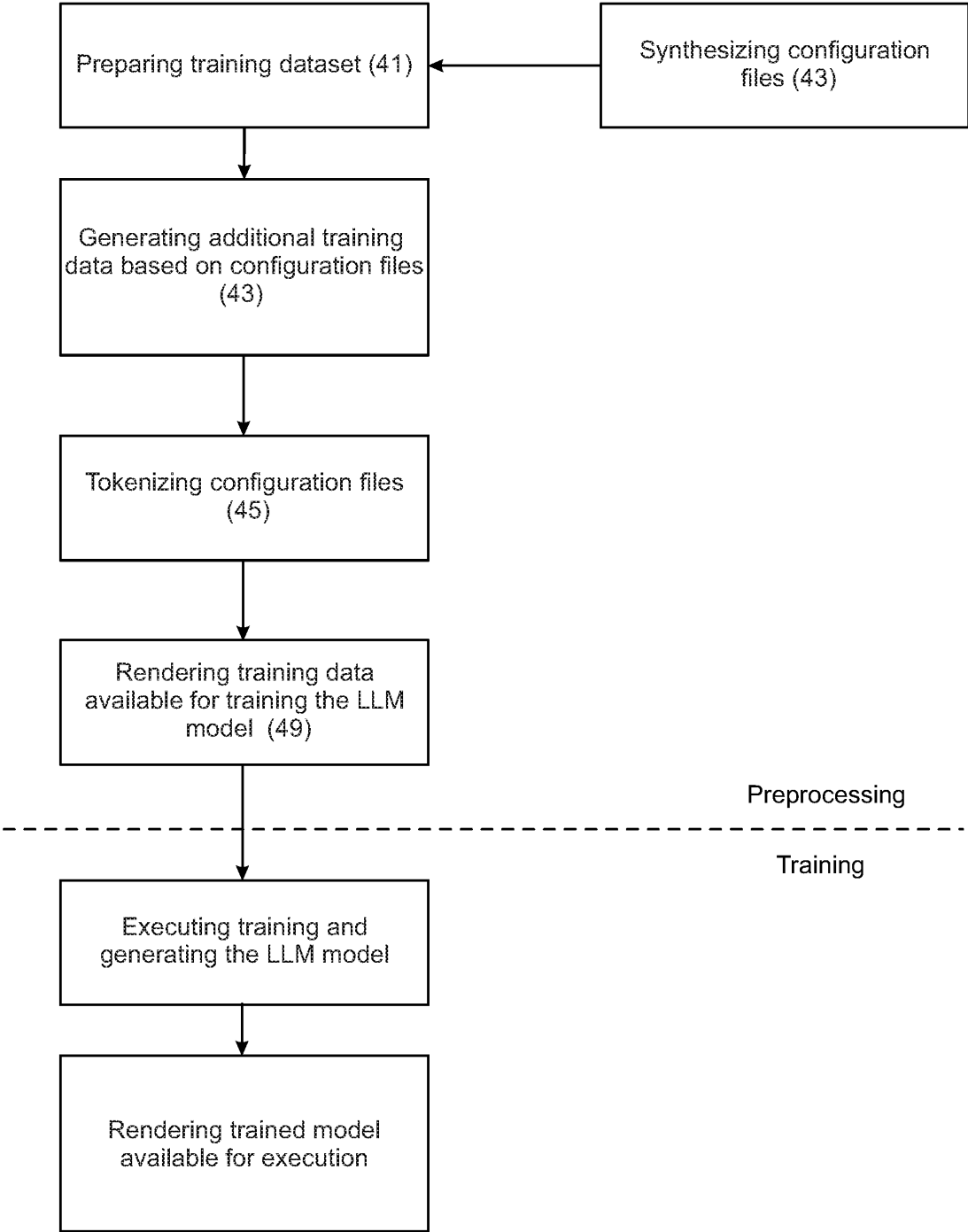


FIG. 4

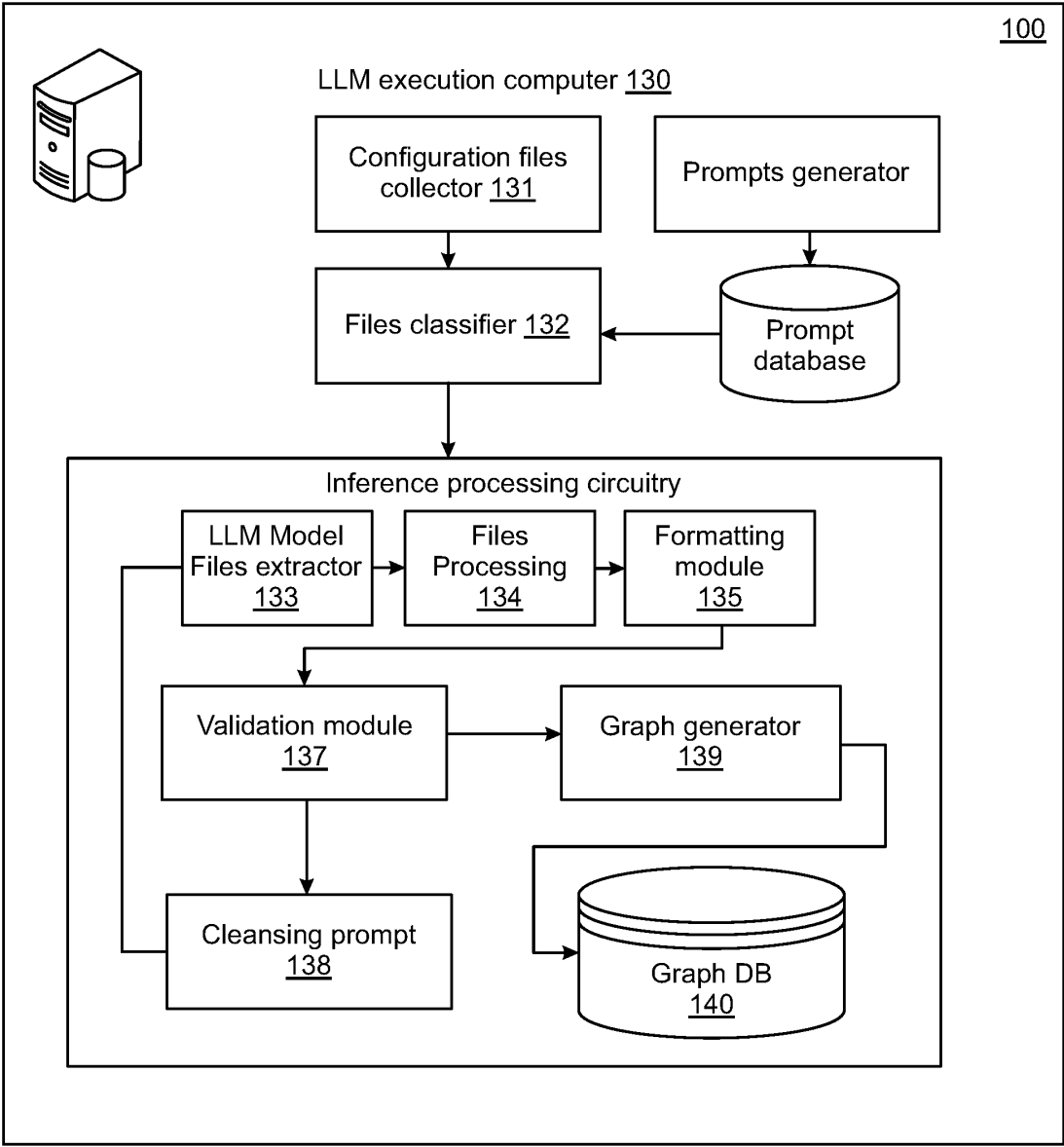


FIG. 5

6/8

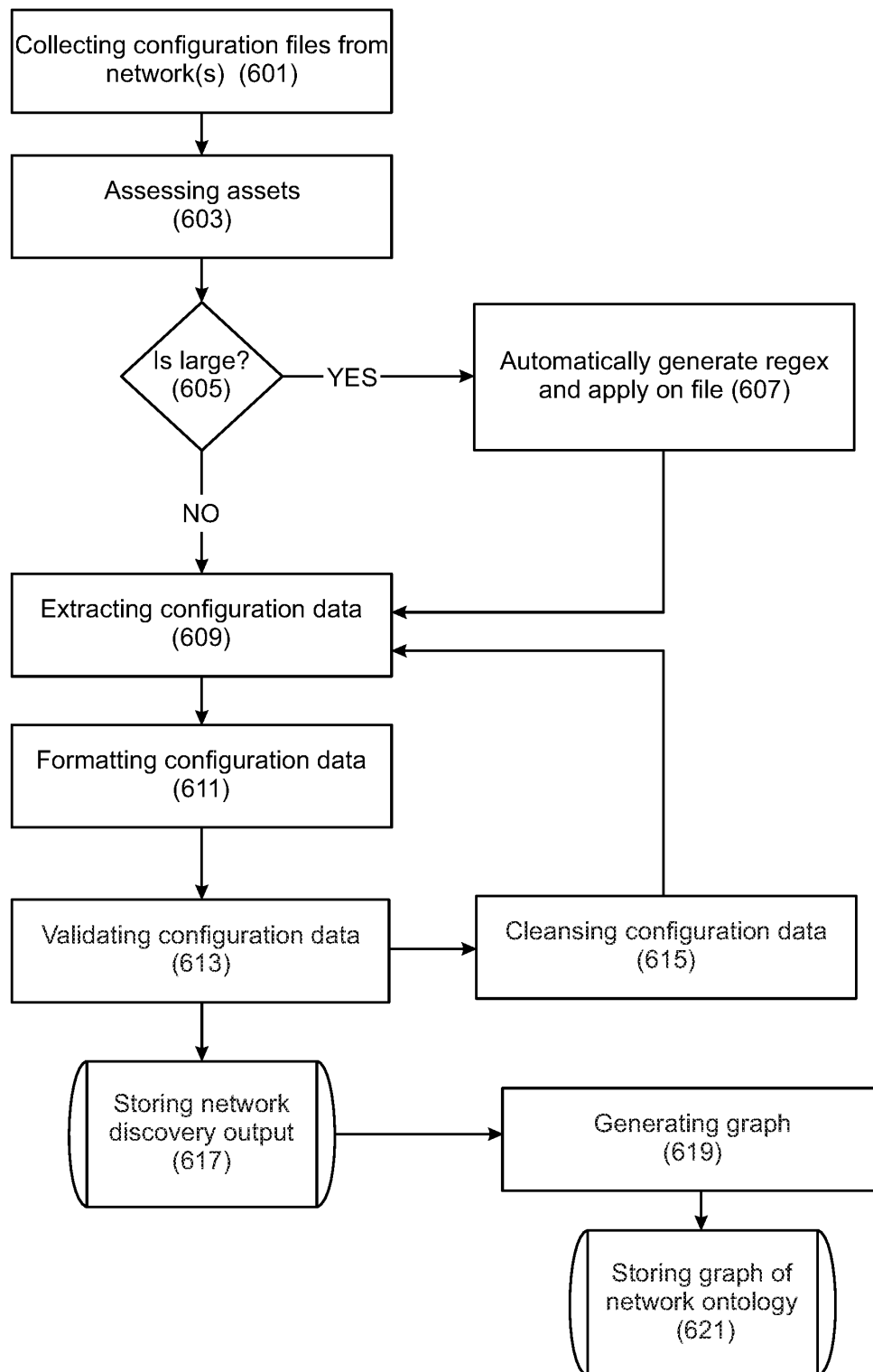


FIG. 6

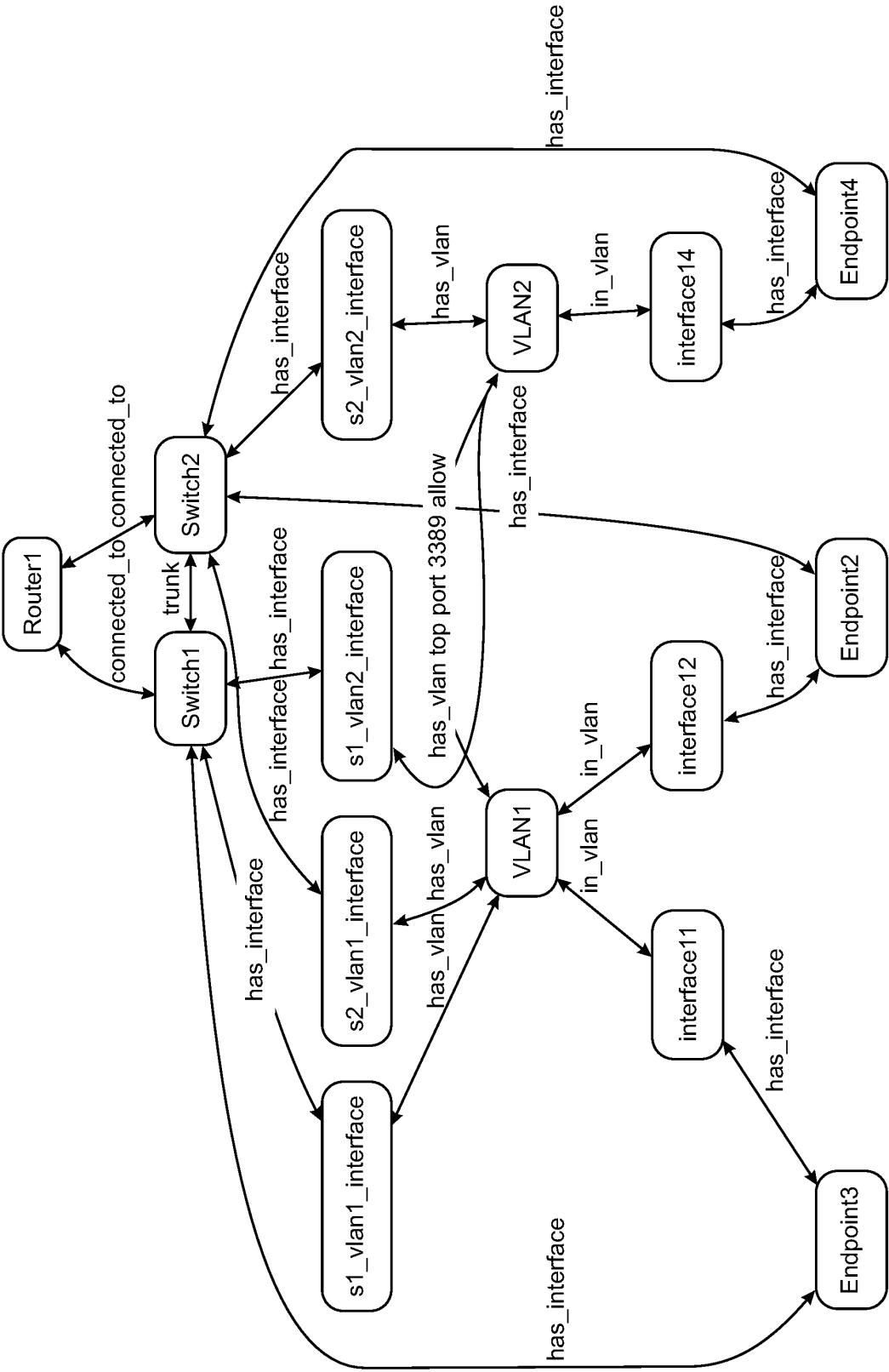


FIG. 7

8/8

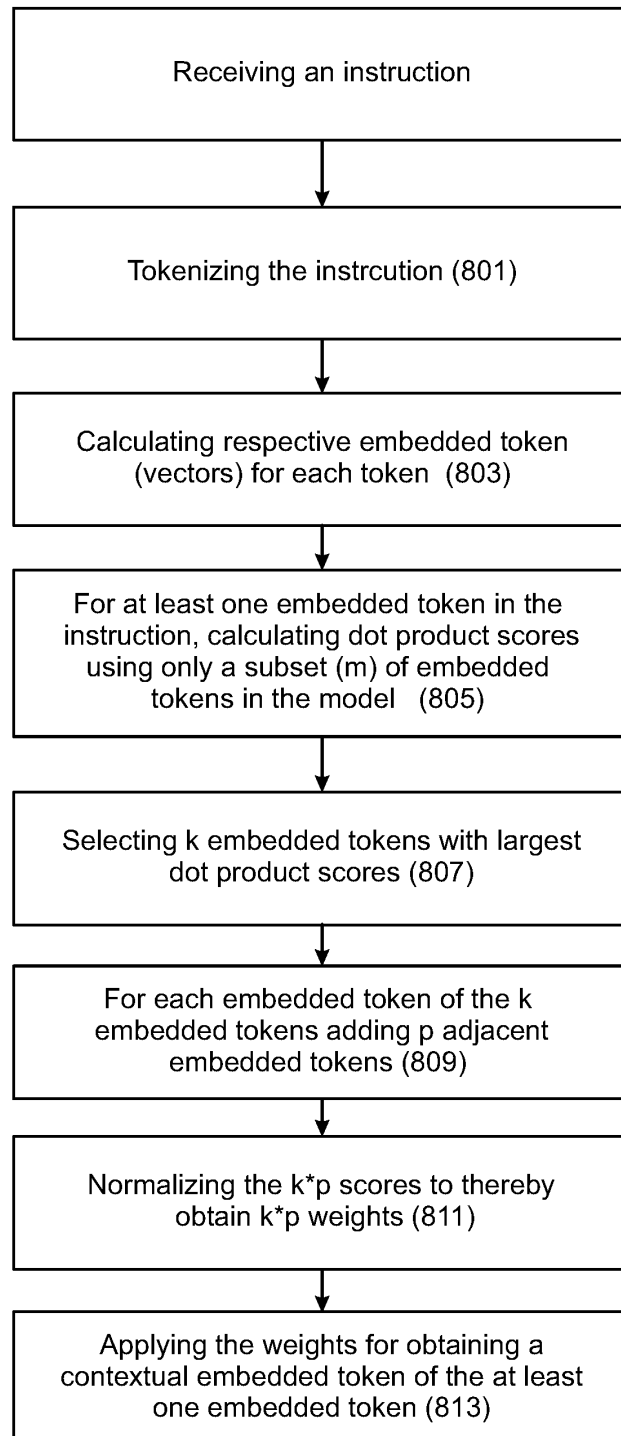


FIG. 8

**LLM-BASED AUTOMATIC DISCOVERY OF OT AND IT
NETWORKS**

FIELD OF THE PRESENTLY DISCLOSED SUBJECT MATTER

The presently disclosed subject matter relates to computer systems and methods
5 of network discovery management and cybersecurity.

BACKGROUND

Network discovery is a process dedicated for the identification and understanding
of the various assets that constitute networks, and particularly operational technology
(OT) networks and Information Technology (IT) networks.

10 By applying network discovery in an organization, valuable information regarding
the organization can be obtained, including the identification, profiling, and mapping of
the different assets which constitute the OT and IT networks, and the identification of
connectivity and communication pathways between these assets.

Through network discovery, organizations can improve their understanding of the
15 network infrastructure, assess its security status, detect potential vulnerabilities, develop
risk mitigation plans, and implement suitable security measures to protect their
operational technology assets against malicious cyber attacks, safeguarding their
reliability and availability, and increasing the overall security posture of the network.

GENERAL DESCRIPTION

20 Network discovery in operational technology (OT) and information technology (IT)
networks is a challenging task due to technical complexities, manual labor, vendor
diversity, and software version variations. Traditionally, network administrators
undertake labor-intensive efforts to manually identify and document each asset within
the network.

25 In OT environments, the presence of legacy systems, proprietary protocols, and
heterogeneous architectures further complicate network discovery. The lack of

standardized documentation and the dependence on vendor-specific tools often impede efficient and accurate identification of assets. Additionally, frequent software updates and varying versions further hinder the network discovery process.

5 In IT networks, the dynamic nature of the infrastructure, constant addition of new devices, and frequent changes in software configurations, present additional challenges. The sheer scale of IT networks, distributed across multiple sites or subnets, further exacerbates the difficulty of asset discovery. Manual inventory processes become labor-intensive and time-consuming, making it challenging to maintain an up-to-date and accurate view of the network assets.

10 The existing solutions for network discovery often rely on a combination of network scans, manual inventory audits, and vendor-specific tools. These methods are resource-intensive, error-prone, and time-consuming. They require significant human effort, specialized knowledge, installation of software or physical hardware within the network, and ongoing coordination with different vendors and software versions.

15 The presently disclosed subject matter is related to an innovative approach that addresses these challenges and streamlines the network discovery process in both OT and IT networks. The suggested approach considerably reduces reliance on manual labor, vendor-specific tools, and the complexities associated with different software versions, while ensuring accurate, efficient, passive, and rapid network visibility and asset
20 identification.

The presently disclosed subject matter includes a computer system and method for linguistic passive automatic discovery of operational technology (OT) and information technology (IT) networks. The discovery can be applied on networks of various sizes, including large networks comprising many assets (e.g., tens of thousands of endpoints
25 and servers or more), as well as smaller networks such as subnets, multi sites, or even sites. In some examples, boundaries of a network (defining which assets belong to a specific network) are defined by a router.

The computer systems and methods disclosed herein utilize a novel large language

model (LLM) trained to comprehensively "understand" configuration files, notwithstanding the fact that configuration data stored in configuration files is not a natural language. Configuration files are special types of files that store information defining various assets and their interactions in an OT or IT network. Each configuration file describes a respective network asset. Assets include, for example, various devices, and particularly network infrastructure components such as switches, routers, gateways, firewalls, sensors, etc. Each configuration file contains information (referred to herein as "configuration data") of a respective asset (device).

The information in configuration files includes for example:

- Information on the assets (or "entities") in the network, including for example, data such as asset class (e.g., hardware, software), asset type (e.g., computer, router, switch, hub, application, operating system, etc.), IP addresses, hostname, network settings, access control, protocols, applications, etc.
- Information on the network topology, including, for example, data on physical and logical relations and links between different assets, the links bandwidth, etc.
- Rules and policies implemented by the different assets, including, for example, network security configurations and policies, cybersecurity parameters, routing protocol, protocol specific configuration, connectivity rules (e.g., which assets can be connected), etc.

The LLM incorporates Machine Learning techniques and is trained, using a large dataset of configuration files, to extract from configuration files in a network (OT and/or IT networks) relevant configuration data.

According to some examples, the configuration files used during training include synthetic configuration files generated by an innovative approach. The synthetic files can be generated by virtually producing numerous possible network architectures. This approach ensures a diverse and extensive training dataset, enhancing the accuracy and effectiveness of the LLM in understanding and extracting data from real-world

configuration files.

According to further examples disclosed herein, the data retrieved from the configuration files is stored in a graph data-structure, which enables easy and efficient processing, manipulation, and retrieval of the data, including comprehensive asset
5 management, risk assessment, and proactive threat hunting.

According to a first aspect of the presently disclosed subject matter there is provided a computer-implemented method of network discovery, comprising:

collecting configuration files from at least one network, each configuration file comprising configuration data that characterizes different assets in the at least one
10 network and their respective connections and properties;

applying on the configuration files a large language model (LLM) that is trained to provide, as output, to at least one received instruction ("prompt"), a respective response comprising specific configuration data extracted from the configuration files; wherein the LLM is capable of retrieving configuration data pertaining to multiple assets and their
15 respective connections in the at least one network, the retrieved configuration data enabling the generating of an ontology of the network, including, for example, information on the assets, information on network topology, and information on rules and policies.

In addition to the above features, the method according to this aspect of the
20 presently disclosed subject matter can optionally comprise one or more of features (i) to (xv) below, in any technically possible combination or permutation:

- I. using the LLM for generating, for each asset and/or connection in the network, a respective data object, wherein all data-objects are characterized by a uniform data-object format (e.g., JSON object).
- 25 II. generating a graph representing the retrieved configuration data, wherein each data-object of a respective asset retrieved from the configuration files is designated as a node in the graph, and each connection retrieved from the configuration files is designated as a vertex in the graph connecting between two

nodes, and wherein each node or vertex is stored with its respective properties retrieved from a respective configuration file;

- III. applying the LLM for determining a type of configuration file, and, responsive to determining a configuration file of a certain type, automatically generating a regular expression and/or a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data;
- IV. wherein the LLM includes a transformer model that provides a next token in a sequence based on previous tokens in the sequence;
- V. comparing configuration data in a respective response of one or more instructions to configuration data in the configuration files, and, in case of a discrepancy, discarding the respective response, and re-processing the one or more instructions;
- VI. generating a training dataset for training the LLM, comprising generating synthetic configuration files;
wherein the synthetic configuration files are generated by a process that utilizes network design software, comprising:
 - automatically manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and
 - executing the network designs to thereby obtain a respective diverse collection of configuration files;
- VII. wherein automatic manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining a configuration file that contains errors;
- VIII. wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and human annotated configuration files;
- IX. wherein generating the training dataset for training the LLM comprises providing,

as input, multiple instruction-response pairs, and training the LLM to provide, to a given instruction, a respective response comprising specific configuration data extracted from the configuration files;

- 5 X. wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to, responsive to ascertaining that a configuration file is of a certain type, automatically generate a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.
- 10 XI. wherein the multiple instruction-response pairs include instruction-software code pairs used for training the LLM to, responsive to ascertaining that a configuration file is of a certain type, automatically generate a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data;
- 15 XII. wherein the at least one network is an operation technology network and/or an information technology network;
- XIII. wherein the trained LLM includes 'n' embedded tokens, each generated from a respective token extracted from the configuration files, the method comprising:
 applying in the LLM a selective attention mechanism, when providing a response to a received instruction, comprising:
20 selecting a subset of 'm' from 'n' embedded tokens and using only the embedded tokens in the subset when applying the attention mechanism.
- XIV. The method further comprises:
 for at least one embedded token generated from the received instruction,
 calculating a dot product score using the subset of 'm' embedded tokens,
25 giving rise to n/m dot product scores;
 selecting from the subset, a group of k embedded tokens with the highest respective dot product scores;
 expanding each of 'k' embedded tokens with a plurality of additional 'p' adjacent tokens, giving rise to a final collection of $\frac{n}{m} + k * p$ embedded tokens;

applying a normalization function (e.g., SoftMax function) on respective dot product scores of the final collection of embedded tokens, thereby obtaining a respective set of weights, and applying the weights for obtaining a contextualized vector for the at least one embedded token.

- 5 xv. The method further comprising using a tokenizer specifically configured for tokenizing the configuration files and feeding the tokenized data to the LLM.

The presently disclosed subject matter further contemplates a computer system comprising at least one processing circuitry configured to execute a method according to the first aspect disclosed above.

- 10 The presently disclosed subject matter further contemplates a computer program product comprising a computer readable storage medium retaining a program of instructions, which, when read by a computer processor, causes the computer processor to perform a method according to the first aspect disclosed above.

- 15 The presently disclosed subject matter further contemplates a non-transitory program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform a method according to the first aspect disclosed above.

- 20 The system, the computer program product, and the non-transitory program storage device, disclosed in accordance with the presently disclosed subject matter, can optionally comprise one or more of features (i) to (xv) listed above, mutatis mutandis, in any technically possible combination or permutation.

- 25 The presently disclosed subject matter further contemplates a computer-program product operable in a computer and recorded on a computer-readable medium for (linguistic) automatic discovery of at least one network, wherein the computer-program product is produced by processes that include:

obtaining a training dataset that includes at least a collection of many configuration files;

training a large language model (LLM) using the configuration files to thereby obtain a trained LLM that provides, as output, to at least one

received instruction ("prompt"), a respective response comprising specific configuration data extracted from the configuration files.

The presently disclosed subject matter further contemplates a method of training a large language model (LLM) to provide, as output, to at least one received instruction ("prompt"), a respective response comprising specific configuration data extracted from configuration files; wherein the LLM is capable of retrieving configuration data pertaining to multiple assets and their respective connections in the at least one network, and generating an ontology of the network according to the retrieved configuration data.

The presently disclosed subject matter further contemplates a computer-product operable in a computer and recorded on a computer-readable medium comprising a large language model (LLM) for automatic discovery of at least one network, wherein the computer-product can be applied on a collection of configuration files obtained from at least one network and provide, as output, configuration data extracted from the configuration files that can be used for generating an ontology of the at least one network;

wherein the computer-product is produced by a process that includes: obtaining a training dataset that includes at least a collection of network configuration files;

training a large language model (LLM) using the network configuration files to thereby obtain a trained LLM that provides as output, to at least one received instruction, a respective response comprising specific configuration data extracted from the configuration files.

The computer program product can optionally comprise one or more of features (i) to (vii) listed below, mutatis mutandis, in any technically possible combination or permutation:

i. Wherein obtaining the training dataset comprises generating a training dataset for training the LLM that comprises synthetic configuration files.

ii. Wherein the process further includes generating the synthetic configuration files comprising:

manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of configuration files.

5 iii. Wherein manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining configuration file that contain errors.

iv. Wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and
10 human annotated configuration files.

v. Wherein generating the training dataset comprises providing, as input, multiple instruction-response pairs, and training the LLM to provide, in response to a given instruction, a respective response comprising specific configuration data extracted from the configuration files.

15 vi. Wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to determine, responsive to determining that a configuration file is of a certain type, and to automatically generate a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

20 vii. Wherein the multiple instruction-response pairs include instruction-software code pairs used for training the LLM to, responsive to ascertaining that a configuration file is of a certain type, automatically generate a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

25 The presently disclosed subject matter further contemplates a computer-implemented method of training a large language model (LLM) for automatic discovery of at least one network, wherein LLM can be applied on a collection of configuration files obtained from at least one network and provide, as output, configuration data extracted from the configuration files; wherein the training comprises: generating synthetic

configuration files and adding the synthetic configuration files to the training dataset.

The computer-implemented method comprising applying a specialized tokenizer on the configuration files is configured to use unique delimiters tailored to the syntax and structure of configuration files.

- 5 The computer-implemented method of training can optionally comprise one or more of features (i) to (vii) listed above with respect to the computer-program product, mutatis mutandis, in any technically possible combination or permutation.

BRIEF DESCRIPTION OF THE DRAWINGS

- 10 In order to understand the presently disclosed subject matter and to see how it may be carried out in practice, the subject matter will now be described, by way of non-limiting examples only, with reference to the accompanying drawings, in which:

Fig. 1 is a high-level block diagram schematically illustrating a computer system **100** dedicated for automatic network discovery, in accordance with an example of the presently disclosed subject matter;

15

Fig. 2 is a high-level flowchart showing operations carried out by system **100**, in accordance with an example of the presently disclosed subject matter;

Fig. 3 is a block diagram schematically illustrating data preparation computer and LLM training computer, in accordance with an example of the presently disclosed subject matter;

20

Fig. 4 is a flowchart of operations carried out during the data preparation process and training process, in accordance with an example of the presently disclosed subject matter;

Fig. 5 is a block diagram schematically illustrating LLM execution in a computer, in accordance with an example of the presently disclosed subject matter;

25

Fig. 6 is a flowchart of operations carried out during the automatic discovery process, in accordance with an example of the presently disclosed subject matter;

Fig. 7 is a schematic illustration of a graph representing the output of an automatic discovery process, in accordance with an example of the presently disclosed subject matter; and

Fig. 8 is a flowchart of operations carried out as part of a selective attention mechanism, in accordance with an example of the presently disclosed subject matter.

DETAILED DESCRIPTION

In the drawings and descriptions set forth, identical reference numerals indicate those components that are common to different embodiments or configurations. Elements in the drawings are not necessarily drawn to scale.

Unless specifically stated otherwise, as apparent from the following discussions, it is appreciated that, throughout the specification, discussions utilizing terms such as "collecting", "generating", "calculating", "determining", "applying", "performing", "assigning" or the like, include an action and/or processes of a computer that manipulate and/or transform data into other data, said data represented as physical quantities, e.g. such as electronic quantities, and/or said data representing the physical objects.

The terms "computer", "computer system", "computer device", "computerized device" or the like, should be expansively construed to include any kind of hardware-based electronic device with a data processing circuitry (e.g., digital signal processor (DSP), graphics processing unit (GPU), tensor processing unit (TPU), field programmable gate array (FPGA), application specific integrated circuit (ASIC), microcontroller, microprocessor etc.). The processing circuitry can comprise, for example, one or more processors operatively connected to computer memory, loaded with executable instructions for executing operations, as further described below.

The operations in accordance with the teachings herein may be performed by a computer specially constructed for the desired purposes, or by a general-purpose computer specially configured for the desired purpose by a computer program stored in

a computer readable storage medium.

As used herein, the phrase "for example," "such as", "for instance" and variants thereof, describe non-limiting embodiments of the presently disclosed subject matter. Reference in the specification to "one case", "some cases", "other cases", or variants thereof, means that a particular feature, structure, or characteristic described in connection with the embodiment(s), is included in at least one embodiment of the presently disclosed subject matter. Thus, the appearance of the phrase "one case", "some cases", "other cases", or variants thereof, does not necessarily refer to the same embodiment(s).

It is appreciated that certain features of the presently disclosed subject matter, which are, for clarity, described in the context of separate embodiments, may also be provided in combination in a single embodiment. Conversely, various features of the presently disclosed subject matter, which are, for brevity, described in the context of a single embodiment, may also be provided separately, or in any suitable sub-combination.

In embodiments of the presently disclosed subject matter, fewer, more and/or different stages than those shown in **Figs. 2, 4, 6, and 8** may be executed. In embodiments of the presently disclosed subject matter, one or more stages illustrated in the figures may be executed in a different order, and/or one or more groups of stages may be executed simultaneously.

Figs. 1, 3, and 5 illustrate a general schematic of the system architecture in accordance with certain examples of the presently disclosed subject matter. Elements in **Figs. 1, 3, and 5** can be made up of any combination of software and hardware and/or firmware that performs the functions as defined and explained herein. Elements in **Figs. 1, 3, and 5** may be centralized in one location or dispersed over more than one location. For example, each one of computers **110, 120, 130, and 140** can be located at a different geographical location, remote from the other computers. Furthermore, in some examples of the presently disclosed subject matter, the system may comprise fewer, more, and/or different elements than those shown in **Figs. 1, 3, and 5**. For example, **Figs. 1, 3, and 5**

show several separate computers, each dedicated for certain functions of the system, however it should be clear to any person skilled in the art that the functionalities can be divided otherwise, for instance in an alternative system-design the components of computers **110** and **120** can be consolidated in a single computer device, and/or
5 computers **130** and **140** can be consolidated in a single computer device.

Bearing the above in mind, attention is drawn to Fig. **1**, which is high-level schematic illustration of a computer system dedicated for linguistic automatic network-discovery as disclosed herein. Fig. **1** is a general example which demonstrates some principles of the presently disclosed subject matter. Components of system **100** shown in
10 Fig. **1** can be connected over any type of communication network, including, for example, any one of the following: the Internet, a local area network (LAN), wide area network (WAN), metropolitan area network (MAN), any type of telephone network (including, for example, PSTN with DSL technology) or mobile network (including for example 4G or 5G mobile communication technologies), or any combination thereof.

By way of non-limiting example system **100** is shown to include several computers, each configured to perform a certain general task, including data preparation computer **110**, LLM training computer **120**, LLM execution computer **130**, and graph generation and data retrieval computer **140**. Fig. **2** is a flowchart showing a high-level view of various processes carried out by system **100**, according to some examples of the presently
20 disclosed subject matter. The processes shown in Fig. **2** are described below in conjunction with the components of Fig. **1**.

At block **21**, a data preparation process is carried out (e.g., by data preparation computer **110**). During the data preparation process special training data dedicated specifically for training an LLM for the purpose of (linguistic) automatic network discovery
25 is assembled into a comprehensive training dataset.

At block **23** the LLM is trained (e.g., by LLM training computer **120**) using the special training data and by implementing a training process specifically designed for training the LLM for (linguistic) automatic network discovery.

At block **25** the (linguistic) automatic network discovery process is carried out using the trained LLM (e.g., by LLM execution computer **130**). The LLM is executed on configuration files obtained from actual OT/IT networks for automatically extracting the relevant configuration data from the files and generating the network discovery results (e.g., network ontology).

As further described below in some examples, the LLM output is stored in a graph data-structure, for easy and efficient data manipulation, retrieval, and display (block **27**; e.g., by graph generation and data retrieval computer **145**). The LLM output (i.e., the network discovery results) generated by the LLM are processed to generate the graph data-structure representing the LLM output. Generation of the graph can be performed for example by LLM execution computer **130** or by graph analysis and data retrieval computer **140**.

Following generation of the graph, various tools dedicated for querying and retrieval of information from the graph can be used for this purpose (block **29**). In some examples, graph generation and data retrieval computer **140** can be configured with data analytics tools that allow users to interact with the generated graph and obtain the desired information.

Turning to Fig. **3**, it shows a more detailed view of data preparation computer **110** and LLM training computer **120**, according to some examples of the presently disclosed subject matter. Fig. **4** is a flowchart showing operations carried out as part of the data preparation process and LLM training, and is described, by way of example, with reference to components of computers **110** and **120**.

At block **41** the training dataset required for training the LLM is prepared. As mentioned above, the data used for training is especially configured for the unique task of generating an LLM dedicated for automatic network discovery. The LLM is trained to comprehensively "understand" configuration files, identify from the configuration files information on the respective assets, and extract from the configuration files relevant data, including for example: network topology, device configurations, access controls,

cybersecurity parameters, and more.

In some examples, data preparation computer **110** comprises training data collector **111** configured to collect and arrange a training dataset for training the model. In general, the training dataset comprises a large corpus of configuration files (e.g.,
5 between tens of thousands and up to millions or more), as well as additional training data generated based on network configuration files.

As mentioned above, configuration files exhibit significant diversity. Configuration files which are used by different vendors, in different releases of the same vendors and in different networks, differ significantly in many aspects, including their content, structure,
10 and syntax. Thus, to obtain a robust LLM which is capable of performing accurate and efficient network discovery in different networks, each operating with different configuration files, the training corpus must include a large variety of configuration files that represent this diversity.

However, configuration files are generally a proprietary asset used by vendors for
15 network device configuration, and therefore availability of the configuration files is naturally limited. Furthermore, configuration files are constantly updated and adapted to accommodate updates in devices, software, standards, and network topology, and therefore existing configuration files do not provide a full picture of all possible variations in configuration files. Yet further, the errors, inconsistencies, and human annotations,
20 which appear in configuration files, can be also diverse. Therefore, even should availability of configuration files be much more abundant, the variation in configuration files would still be a limiting factor in the generation of a training dataset.

To bridge this gap a novel technique has been developed by the inventors for generating a configuration files dataset with high diversity. According to some examples
25 data preparation computer **110** (e.g., using data collector **111**) obtains configuration files from three different sources, including, proprietary configuration files, human annotated configuration files, and synthetic configuration files.

- Proprietary configuration files are files used by vendors for configuring devices

in OT/IT networks. These files are collected from various vendors and added to the collection of configuration files in the training dataset.

- Human annotated files are configurations which are manually modified by human users to thereby increase variability of the configuration files. Importantly, human annotators can insert "human contrived" modification which may scarcely appear in proprietary configuration files, and thereby enrich the variability of the configuration files in the training dataset. This may include non-default configuration parameters (which, in most cases, are left in their default settings by network and cybersecurity managers), and manually inserted human errors and/or comments.
- Synthetic configuration files.

The presently disclosed subject matter contemplates a computer system, a computer implemented method, and a non-transitory program storage device loaded with computer instructions, all dedicated for the synthesis of configuration files (block 43). According to some examples, data preparation computer 110 comprises data synthesis module 112 configured for synthesizing configuration files, as further discussed below. In other examples, data synthesis module 112 can be implemented in a different computer, e.g., in a dedicated processing circuitry configured to generate the synthetic files and deliver them to data preparation computer 110.

According to one example, configuration files synthesis is performed by computer-based automatic manipulation of network design software output. Network design software tools are types of software programs which are used for designing a network (e.g., Graphic Network Simulator 3 (GNS3)). Once a network design is finalized, part of the output are respective configuration files of the newly created network. According to the presently disclosed subject matter, a computer implemented method is disclosed which is configured to interface with the network design software and automatically manipulate the design parameters to thereby provide a large variety of automatically generated design instructions. By executing the design instructions and generating the respective network topologies, the respective configuration files of the different topologies are

obtained. Manipulation of the design parameters includes, for example, adding/removing devices, changing the device types of existing devices, increasing/decreasing the size of the network, inserting various alternative software types, inserting various alternative security and access settings, inserting various alternative types and definitions of
5 connections between assets in the network, etc.

Manipulation of the design parameters may further include the intentional insertion of errors into the network design. This may include, for example, inserting undefined values, gibberish, incorrect values, empty files, empty fields, overflowing files, misplaced comments, etc. This creates configuration files that contain a variety of errors
10 and enables to train the LLM to cope with such errors that may be found in configuration files during network discovery of actual networks and thus further increase robustness of the LLM.

The generation of synthetic configuration files can be carried out by a software (implemented for example by data synthesis module **112**) that combines fuzzers (aka
15 "fuzz testing tools"), obfuscators, and generative approaches. Fuzzers are used for intentionally inserting errors into the network designs. Obfuscators are used for obscuring data such as IP addresses and passwords for further complicating the network design. Generative approaches use algorithms or AI to automatically create or modify configuration files, ensuring consistency, security, and compliance with predefined
20 templates, policies, or requirements. This combination ensures the efficient generation of diverse synthetic network configuration files. The raw training dataset, including the collection of configuration files, can be stored in a dedicated computer storage device (including non-transitory computer storage, e.g., raw training dataset repository **114**).

In addition to the configuration files, the training dataset includes other data. At
25 block **43** additional types of training data are generated. The instruction layer of the LLM enables users to provide guidance to the LLM, guiding the model to generate the desired response to each different instruction (also known as "prompt"). The training dataset therefore includes many examples of a variety of pairs of instructions (or "prompts") and the respective desired response, i.e., the relevant configuration data that correctly

responds to the prompt. Notably, in the following description the terms "instruction" and "prompt" are used interchangeably.

According to some examples, various types of instructions are used, including instructions for obtaining assets (devices), instructions for obtaining connections
5 between assets, and instructions for obtaining properties of assets. The training dataset therefore includes different types of instruction-response pairs for training the model to respond to all types of instructions. Through the inclusion of different types of instruction-response pairs in the training dataset, the model gains the ability to respond effectively to all types of instructions, which in turn enables greater granularity and flexibility when
10 retrieving data from configuration files. During training, the LLM is fed with the instruction-response pairs to facilitate training of the model to provide, in response to a specific prompt, the appropriate configuration data. The instruction-response pairs are generated as part of the training dataset preparation.

According to examples of the presently disclosed subject matter, the LLM is
15 trained to provide the responses in a predefined uniform format. One non-limiting example of such a response format is a JSON (JavaScript Object Notation) object. In other examples other formats may be used, e.g., XML format. Accordingly, the responses of the instruction-response pairs are provided in the correct format, e.g., JSON format, where each response corresponding to any asset in the network is provided and stored in the
20 same predefined format. This enables to train the LLM to provide configuration data, in response to instructions, in a uniform format.

The following are examples of instructions:

"Extract all endpoints with no endpoint-detection";

"Extract groups and privileges"; and

25 "Extract inbound rules from external networks".

The response includes a JSON object for each asset, which includes various properties related to the asset. Notably, a single JSON object can be constructed from

configuration data obtained using more than one instruction.

Another type of training data which is generated as part of the training dataset preparation process are instruction-Regex pairs. Some configuration files are very large. Due to certain limitations of the transformer model, which is used as part of the LLM, it performs poorly (e.g., exhibits a dramatic slowdown during inference) when applied on very large files (e.g., greater than 8K tokens). The size of some configuration files (e.g., firewall configuration files), which is very large, may therefore impede training of the entire files concurrently. Normally this problem is solved by dividing large files into smaller chunks and processing each chunk separately. However, since contextual information in configuration files is distributed over the entire file, processing of the entire files is imperative to enable the model to correctly derive relations between the different assets in the file and learn how to correctly respond to instructions and retrieve the relevant data. For this reason, division of the file into smaller chunks, and training each chunk individually, is impractical.

A novel approach is disclosed that enables to train the LLM to process large files in a special way that alleviates the above-mentioned technical problem. This includes using regular expressions (Regex) dedicated for extracting from the configuration files the relevant configuration data. The LLM is trained to identify large files and dynamically and automatically generate an appropriate Regex configured to extract the configuration data required for the respective response.

According to some examples, during training the LLM is fed with instruction-response pairs dedicated for training the model to discern between file types. Such an instruction-response pair includes an instruction that queries the type of configuration file, and a corresponding response that indicates the respective file type.

In addition, the LLM is fed with many different examples of instruction-Regex pairs to facilitate training of the model to generate, following identification of certain configuration file types, an appropriate Regex that can extract for the configuration file the appropriate configuration data. As mentioned above, this approach is used for

processing large files such as firewall configuration files. The LLM is further trained to apply the generated Regex on the relevant configuration file and provide the relevant configuration data. The instruction-Regex pairs are generated, for example, as part of the training dataset preparation.

5 The presently disclosed subject matter further contemplates the use of computer program code dynamically and automatically generated for extracting, from the configuration files, the relevant configuration data. The LLM is trained to identify large files and automatically generate a computer code configured to extract the configuration data required for the respective response.

10 According to some examples, during training the LLM is fed with instruction-response pairs dedicated for training the model to discern between file types. Such an instruction-response pair includes an instruction that queries the type of configuration file, and a corresponding response that indicates the respective file type.

 In addition, the LLM is fed with many different examples of instruction-computer
15 code pairs to facilitate training of the model to generate, following identification of certain configuration file types, an appropriate computer code that can parse the configuration files and extract therefrom the appropriate configuration data. As before, this approach can be used for processing large files such as firewall configuration files. The LLM is further trained to dynamically apply the generated computer code on the
20 relevant configuration file and provide the relevant configuration data. The instruction-computer code pairs are generated for example, as part of the training dataset preparation.

 The use of computer code presents advantages over regular expressions for parsing and analyzing configuration files to determine their various properties. While
25 Regex is suitable for simple pattern matching, it becomes cumbersome and difficult to maintain when dealing with complex configurations and nested structures commonly found in network configuration files, such as hierarchical settings, conditional statements, and diverse syntax elements. In contrast, utilizing computer code, such as Python or

JavaScript, enables the implementation of sophisticated parsing logic capable of efficiently handling these intricacies.

Computer code offers the flexibility to adapt to different file formats, validate data, manage dependencies, and incorporate comprehensive error handling. Moreover, 5 computer code allows for the incorporation of dynamic changes, modular functions, and detailed documentation, ensuring enhanced readability and maintainability.

Additionally, computer code facilitates dynamic and conditional parsing, which is particularly advantageous when dealing with configuration files containing conditional statements or environment-specific settings. By utilizing a programming language, it is 10 possible to incorporate logic that handles these conditions and dynamically adjusts the parsing process accordingly. This adaptability is important for responding to changes in configuration formats or content without necessitating extensive rewrites of the parsing logic.

At block **45** the configuration files in the training dataset are tokenized. As 15 configuration files are not natural language, a special tokenization process was devised for this purpose. The tokenization process uses delimiters, which are different to standard tokenization delimiters, and which are adapted to the words and symbols found in different configuration files. A vocabulary of tokens can be generated and mapped to a unique index.

20 In some examples a dedicated Machine Learning model is trained to execute the tokenization. The Machine Learning model can be, for example, a supervised model that is trained with a dataset of annotated configuration files. The trained machine learning tokenizer can be used during training and execution of the LLM for tokenizing the configuration files.

25 The tokenizer plays a central role in LLMs, facilitating the conversion of text into tokens and back. It functions independently within the LLM pipeline, utilizing its own specialized training datasets and algorithms, such as Byte Pair Encoding (BPE). After undergoing training, the tokenizer is equipped to carry out two primary functions:

``encode()``, which transforms text into tokens, and ``decode()``, which reverts tokens back into text. This process is central in attaining effective operation of LLMs, ensuring that the model can accurately interpret and generate language.

5 In the context of configuration files, which do not follow natural language patterns, a specialized tokenization process has been developed. This process employs unique delimiters tailored to the syntax and structure of various configuration files, differing from those used in standard text tokenization. By utilizing these customized delimiters, a vocabulary of tokens can be created, each mapped to a unique index, enabling efficient and accurate processing of configuration file contents.

10 Training a custom tokenizer for an LLM designed to interpret network configuration files is important for several reasons. Firstly, network configuration files contain highly specific syntax and terminology unique to networking protocols and device configurations. Standard tokenizers, trained on natural language data, are ill-equipped to handle this specialized content, leading to ineffective tokenization and sub-optimal model
15 performance.

Secondly, by developing a tokenizer that understands the intricacies of network configuration files, the LLM can more accurately parse and interpret the information within these files. This allows the model to convert the raw configuration data into meaningful knowledge on the network's structure, performance, and security posture.

20 Finally, a custom tokenizer ensures that the LLM can handle the wide variety of syntax and symbols found across different types of network configuration files. This adaptability is crucial for creating a robust and versatile model capable of functioning effectively in diverse networking environments. By mapping each unique token to an index, the tokenizer facilitates efficient data processing and enhances the model's ability
25 to learn from and act on network configuration data. The tokenizer is also used during inference for tokenizing configuration files which are provided as input to the trained model.

In some examples, data preparation computer **110** comprises configuration files

tokenizer **116** configured for executing the tokenization process as described above. In other examples, configuration files tokenizer **116** can be implemented in a different computer e.g., in a dedicated processing circuitry configured to receive the configuration files, tokenize the files, and transmit the tokenizer output back for further training of the LLM.

The training dataset, ready for being used for training the LLM, can be stored in a dedicated (e.g., non-transitory) computer storage device (e.g., final training dataset repository **118**). The final training dataset includes the tokenized configuration files, instruction-response pairs, and instruction-Regex pairs.

Once the training dataset is ready, it is made available for training the model (block **49**). In some examples, the training dataset is provided to the LLM hub (**122**) in LLM training computer **120**, which is configured, *inter alia*, to coordinate between various layers of the LLM training process.

According to some examples, the training process can be divided into three layers, each layer being dedicated for performing a certain part of the training process, including a fundamental layer, instruct layer, and reward and reinforcement layer. In the illustrated example, LLM training computer **120** comprises three respective modules (**124**, **126**, and **128**) operatively connected to model hub **122**, each module configured for executing operations related to one of the layers.

The fundamental layer (implemented by module **124**) is configured to receive the configuration files in the training dataset as input and to execute word embedding on the files, where the tokenized configuration files are converted into a respective sequence of word vectors (referred to herein as "embedded tokens"), which preserve their semantic relations. The fundamental layer is further configured for training a transformer model (e.g., autoregressive transformer model) which is a neural network model that learns to predict the next token in a sequence, given the previous tokens in the sequence. The transformer model also includes a self-attention mechanism that enables the model to identify relationships and dependencies between different tokens in the input and

improve next token prediction.

The instruct (or "instruction") layer (implemented by module **126**) is configured to further train the model to follow specific instructions. The instruct layer can be a separate component or integrated within the model's input pipeline, depending on the specific implementation. The instruct layer is dedicated for fine-tuning the fundamental layer to improve user interaction with the model. In a deeper sense, the instruct layer causes the weights of the neural network to change, so that the output of the model is better suited to the instruction received as input.

According to the presently disclosed subject matter, the instruct layer receives, as input, the instruction-result pairs, and learns to predict the next word in a sequence, given the instructions and the previous words, such that the appropriate result is provided to a given instruction. As explained above, the LLM is trained to provide the results in a certain predefined uniform format (e.g., JSON objects).

As explained above, according to some examples, a technical solution for processing large configuration files is provided, which includes training the model to generate regular expressions (Regex) dedicated for extracting the relevant data from large configuration files (e.g., of 8K tokens or more). According to this example, the instruct layer is trained to identify types of configuration files (e.g., based on corresponding instruction-response pairs), and in response to the identification of a large file (e.g., firewall configuration file), to automatically and dynamically generate an appropriate Regex designed to provide information according to the respective instruction. To this end the LLM is fed with multiple examples of instructions-Regex pairs, and is trained, based on these pairs, to generate the appropriate Regex.

In some examples, training computer further includes a reinforcement layer (implemented by module **128**) configured to learn from feedback, and thus further fine-tune the model and provide more accurate results, which are also, in general, more human accustomed. This fine-tuning process guides the model towards generating more desirable outputs for a target task, in this case providing more accurate information on

assets in the network during network discovery in response to a respective instruction. As with the instruct layer, the reinforcement layer also causes the weights of the neural network to change so that the output of the model is augmented.

Once the training process is complete, the trained model is made available for
5 execution (e.g., in trained LLM computer data repository).

Proceeding to Fig. 5, it shows a more detailed view of LLM execution computer
130, according to some examples of the presently disclosed subject matter. Fig. 6 is a
flowchart showing operations carried out as part of the LLM execution of an automatic
network discovery process, according to some examples, and is described, by way of
10 example, with reference to components of LLM execution computer **130**.

Assuming a certain network is being processed, at block **601** the configuration files
of the different devices in the network are retrieved from the network (e.g., by
configuration files collector **131**).

According to some examples, application of the LLM on the retrieved
15 configuration files involves conditional activation of prompts (instructions). To this end, a
set of prompts are designed, and prompt activation conditions are defined. In general, it
is desired to create a set of prompts that are specifically adapted to align with various
conditions or criteria, where each prompt is constructed to direct the LLM towards
producing the intended or desired response. In the current case, prompts that can be
20 used for obtaining different types of configuration data are generated. Prompt activation
conditions determine which prompt should be activated under different circumstances.
The conditions can be based on various factors, such as input characteristics, context, user
preferences, or intended results.

In some examples, LLM execution computer **130** includes an interactive prompt
25 generator, configured to implement prompt activation, either based on predefined
prompt activation programs or in response to real-time user defined prompt activation.
Prompt activation programs (that include the prompts and prompt conditions) can be
stored in a prompt database operatively connected to LLM execution computer **130**.

In some examples, the configuration files are assessed to determine whether any of the files is considered a large file i.e., a file that exceeds a certain size (block **603**; e.g., by file classifier **132**). Alternatively, or additionally, the configuration files are assessed to determine their type, e.g., whether any of the files is a firewall configuration file. This assessment can be done using an appropriate prompt configured to determine the type of a configuration file. According to some examples, initially an attempt is made to process the configuration file using the transformer model (with the instructions), and only if this fails, the Regex is used for extracting data from the file instead.

By applying conditional activation of prompts, the process continues according to the type of file, where, in case the file is a large file (and/or a firewall configuration file), the response involves the generation of an appropriate Regex (block **607**), which is then applied on the respective "large file" to thereby extract the relevant configuration data from the file (block **609**). For other types of files, the trained LLM (transformer model) is applied on the configuration files retrieved from the network (which are tokenized as described above) to thereby extract the relevant configuration data from the file (block **609**).

The configuration data extracted from the configuration files includes the various assets, the relations, and interconnections between the assets and properties of the assets and the connections. Examples of connections include vlan – interface; vlan – vlan; and interface – endpoint.

The configuration data extracted from the configuration files using the LLM model (with or without automatic Regex generation and application) is transformed into the predefined output format (block **611**; e.g., by data formatting module **135**). For example, the configuration data extracted from each configuration file can be represented by a respective JSON object.

In some examples, the output of the automatic network discovery is validated (block **613**; e.g., by validation module **137**). This includes processing the output data received in response to one or more instructions, and determining whether it includes

false output (e.g., AI hallucinations). To this end, the output configuration data (e.g., in the JSON object) is compared with the original configuration files, and if a discrepancy between the data generated by the LLM and the data in the original files is found, the false configuration data is removed (block **615**; e.g., by cleansing module **138**) and the
5 instructions are sent back to be re-processed by the LLM (block **609**). In some examples, the network discovery output (e.g., validated JSON objects) is stored in a dedicated (e.g., non-transitory) computer storage device (block **617**).

According to some examples, the network discovery output is stored in a graph data-structure that logically represents the network ontology including the different
10 devices, their properties, and the connections between the devices in the network (block **619**, e.g., by graph generator **139**).

In some examples, each device in the network is represented as a node in a graph, and relations between devices are represented by vertexes. Each node and vertex can be implemented as an object that comprises the relevant information characterizing the
15 respective device or connection. Fig. 7 is an example of a graph generated based on the linguistic network discovery process disclosed herein. The graph shows various nodes, each representing a certain asset (e.g., router, switches, interfaces, vlans, endpoints, etc.) and their interconnection. Notably, the graph is presented for illustrative purposes and does not include all information, such as the various properties of each node or
20 connection.

Computer **130** can be configured to execute the LLM repeatedly, e.g., periodically, or in response to a certain predefined event, to account for any changes in the network and enable to maintain an updated picture of the topology of the network.

The generated graph can be stored in an appropriate computer storage device
25 (e.g., graph DB **140**). The graph provides a powerful tool for organization, representation, and retrieval of configuration data of the network. The graph data-structure enables administrators and cybersecurity personnel to easily identify vulnerabilities, potential attack vectors, and conduct proactive threat hunting, thereby enhancing the overall

network security.

As mentioned above, system **100** may further include graph analysis and data retrieval computer **140**, configured to enable analysis of the graph by users to deduce insights with respect to assessing its security status, detect potential vulnerabilities, develop risk mitigation plans, and more. The graph DB can be made accessible to computer **140** for this purpose. In some examples, graph generation (block **619**, where graph generator **139** is implemented as part of computer **140**) can be also executed by computer **140** upon receiving from LLM execution computer **130** the network discovery output.

The attention (or self-attention) mechanism is a key feature of the transformer model neural network architecture. The attention mechanism enables the transformer model to focus on different parts of the input sequence when determining how relevant is a given element (e.g., token) to the current context, and to generate outputs accordingly. It helps to capture long-range contextual dependencies and relationships between elements in a sequence. The attention mechanism weighs word vectors according to the context of the word and provides contextualized representation of vectors, thus pushing closer words vectors which are contextually more related than others.

As part of the attention mechanism, scores are calculated for each word vector by applying the dot product (aka "key-query dot product") between the word vectors and any other word vector (including itself, giving rise to n^2 scores). The calculated scores indicate contextual proximity between the original words. The scores are normalized, using, for example, the SoftMax function, to thereby obtain corresponding probability scores (or weights) that collectively add up to 1. The weights are then used for weighing the original word vectors and obtaining respective context vectors, which capture the importance of different word vectors according to the context.

Following the generation of the model, during inference, a decoder is implemented taking as input the embeddings of previously generated tokens and providing predictions for the next token in the sequence. The decoder utilizes self-

attention to calculate attention scores, and the attention scores are used to weight the contributions of different tokens during the generation process.

In the current application the context size is very large, often with as many as 100,000 tokens or more. This is a huge amount of data which creates a significant processing bottleneck when applying the attention mechanism during inference.

According to some examples of the presently disclosed subject matter, a selective attention mechanism is applied by the decoder during inference. A selective attention mechanism is less processing intensive, while enabling a sufficiently accurate prediction by the LLM model.

Fig. 8 is a flowchart of operations carried out as part of a selective attention mechanism, according to some examples of the presently disclosed subject matter. Considering a certain instruction is received, the instruction is tokenized to obtain respective tokens (block 801) and word embedding is applied on each token to obtain respective vectors (herein after "embedded tokens"; block 803).

According to the selective attention mechanism disclosed herein, only a selection of embedded tokens in the model are used. Instead of calculating a dot product score for every embedded token in the model, a dot product is calculated using a subset of the embedded tokens. In some examples, a dot product is generated using one embedded token out of every 'm' embedded tokens in the model, such that n/m dot product operations are performed for a given token in the instruction instead of $n * n$ (where n is the total number of tokens in the model; block 805). According to an example, $m=32$. Assuming 100,000 tokens are being processed, the dot product is applied on 3125 embedded tokens instead of all 100,000, for a total of 3125 dot product scores.

In the following stage, a group of 'k' embedded tokens which produced the largest dot product scores are selected from the subset of n/m dot product scores (e.g., $k=64$ (block 807). One example of selecting the k tokens involves applying a SoftMax function on the n/m key-query dot product values and selecting (k) tokens with the largest key-query dot product values. By setting the selection cutoff according to key-query dot product distribution, dynamic selection is obtained.

Next, the group of k embedded tokens is expanded, by adding for each embedded token of the k embedded tokens in the group, ' p ' adjacent embedded tokens. In some examples, ' p ' equals ' m ', where in case $m = 32$, $p=32$ (block **809**). In some examples, for a given embedded token from the group of ' k ' embedded tokens, the $p/2$ (e.g., 16) embedded tokens preceding the given token and the $p/2$ embedded tokens proceeding the given token are added to the group. The resulting output is an expanded group of $k * p$ dot product scores, each calculated using a respective embedded token in the model (i.e., the final collection of embedded tokens). The final collection of $\frac{n}{m} + k * p$ dot product scores are normalized using a normalization function such as the SoftMax function (block **811**), thereby converting the scores to probability scores (weights).

Finally, the weights are used for weighting the original embedded tokens and obtaining respective context vectors, which capture the importance of different embedded tokens according to the context (block **813**).

Thus, according to the selective attention mechanism, attention is applied on only part of the tokens, while other tokens remain with their original embedded values without the additional weighting. The inventors have found that this approach, when implemented on a large collection of tokens (extracted in this case from configuration files), helps to reduce the processing intensiveness while providing sufficiently accurate contextual information which enables to train the LLM model to provide the required configuration data from the files.

It will also be understood that the system according to the presently disclosed subject matter may be a suitably programmed computer. Likewise, the presently disclosed subject matter contemplates a computer program being readable by a computer for executing the methods of the presently disclosed subject matter. The presently disclosed subject matter further contemplates a machine-readable non-transitory memory tangibly embodying a program of instructions executable by the machine for executing methods of the presently disclosed subject matter.

It is to be understood that the presently disclosed subject matter is not limited in its application to the details set forth in the description contained herein or illustrated in

the drawings. The presently disclosed subject matter is capable of other embodiments and of being practiced and carried out in various ways. Hence, it is to be understood that the phraseology and terminology employed herein are for the purpose of description and should not be regarded as limiting. As such, those skilled in the art will appreciate that

5 the conception upon which this disclosure is based may readily be utilized as a basis for designing other structures, methods, and systems for carrying out the several purposes of the present presently disclosed subject matter.

CLAIMS:

1. A computer-implemented method of network discovery, comprising:
collecting configuration files from at least one network, each configuration file comprising configuration data that characterizes different assets in the at least one
5 network and their respective connections and properties;
applying on the configuration files a large language model (LLM) trained to provide as output responsive to at least one received instruction, a respective response comprising specific configuration data extracted from the configuration files; wherein the LLM is capable of retrieving configuration data pertaining to multiple assets and their
10 respective connections in the at least one network.
2. The computer-implemented method of claim 1 comprising generating, based on the retrieved configuration data, an ontology of the at least one network including, at least part of: information on the assets, information on network topology, and information on rules and policies.
- 15 3. The computer-implemented method of claim 1 or 2 comprising:
using the LLM for generating, for each asset and/or connection in the at least one network, a respective data object, wherein all data-objects are characterized by a uniform data-object format.
- 20 4. The computer-implemented method of any one of the preceding claims comprising: generating a graph representing the retrieved configuration data, wherein each data-object of a respective asset retrieved from the retrieved configuration data is designated as a node in the graph, and each connection retrieved from the retrieved configuration data is designated as a vertex in the graph connecting between two nodes, and where each node or vertex is stored with its respective properties.
- 25 5. The computer-implemented method of any one of the preceding claims comprising:
applying the LLM on the configuration files in the at least one network for determining a type of configuration file, and, responsive to determining a configuration

file of a certain type, automatically generating a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

6. The computer-implemented method of any one of the preceding claims
5 comprising:

applying the LLM on the configuration files in the at least one network for determining a type of configuration file, and, responsive to determining a configuration file of a certain type, automatically generating a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective
10 configuration data.

7. The computer-implemented method of any of the preceding claims, wherein the LLM includes a transformer model that provides a next token in a sequence based on previous tokens in the sequence.

8. The computer-implemented method of any one of the preceding claims
15 comprising: comparing configuration data in a respective response of one or more instructions to configuration data in the configuration files, and, in case of a discrepancy, discarding the respective response and re-processing the one or more instructions by the LLM.

9. The computer-implemented method of any one of the preceding claims
20 wherein a training dataset for training the LLM comprises synthetic configuration files.

10. The computer-implemented method of claim 9, wherein the synthetic configuration files are generated by a process that utilizes network design software, comprising:

manipulating parameters of the network design software to thereby obtain a
25 diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of configuration files.

11. The computer-implemented method of claim 10, wherein manipulation of

parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining configuration files that contain errors.

12. The computer-implemented method of any one of claims 9 to 11, wherein
5 the training dataset includes an assembly of the synthetic configuration files, proprietary configuration files, and human annotated configuration files.

13. The computer-implemented method of any one of claims 1 to 12, wherein
the training dataset comprises providing, as input, multiple instruction-response pairs,
and training the LLM to provide, in response to a given instruction, a respective response
10 comprising specific configuration data extracted from the configuration files.

14. The computer-implemented method of claim 13, wherein the multiple
instruction-response pairs include instruction-Regex pairs used for training the LLM to
automatically generate, responsive to determining that a configuration file is of a certain
type, a regular expression adapted for parsing the configuration file and retrieving from
15 the configuration file the respective configuration data.

15. The computer-implemented method of claim 13, wherein the multiple
instruction-response pairs include instruction-computer code pairs used for training the
LLM to automatically generate, responsive to determining that a configuration file is of a
certain type, a computer program code adapted for parsing the configuration file and
20 retrieving from the configuration file the respective configuration data.

16. The computer-implemented method of any one of the preceding claims,
wherein the at least one network is an operation technology network and/or an
information technology network.

17. The computer-implemented method of any one of claims 1 to 16 wherein
25 the LLM includes 'n' embedded tokens, each generated from a respective token extracted
from the configuration files, the computer-implemented method comprising:
applying in the LLM a selective attention mechanism, when providing a response
to a received instruction, comprising:

selecting a subset of 'm' from the 'n' embedded tokens and using only the embedded tokens in the subset when applying the attention mechanism.

18. The computer-implemented method of claim 17, further comprising:
for at least one embedded token generated from the received instruction:
5 calculating a dot product using the subset of 'm' embedded tokens, giving rise to $\frac{n}{m}$ dot products;

selecting from the subset, a group of k embedded tokens with the highest respective dot products;

expanding each of 'k' embedded tokens with a plurality of additional 'p'
10 adjacent tokens, giving rise to a final collection of $\frac{n}{m} + k * p$ embedded tokens;
applying a normalization function on respective dot product of the final collection of embedded tokens, thereby obtaining a respective set of weights, and applying the weights for obtaining a contextualized vector for the at least one embedded token.

19. A computer program product comprising a computer readable storage
15 medium retaining a program of instructions, which, when read by a computer processor, causes the computer processor to perform a method according to any one of claims 1 to 18.

20. A non-transitory program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform a method
20 according to any one of claims 1 to 18.

21. A computer system comprising at least one processing circuitry configured to execute a method of network discovery using an LLM according to any one of claims 1 to 18.

22. A computer-program product operable in a computer and recorded on a
25 non-transitory computer-readable medium comprising a large language model (LLM) for automatic discovery of at least one network, wherein the computer-product product can be applied on a collection of configuration files obtained from at least one network, and

provide, as output, configuration data extracted from the configuration files that can be used for generating an ontology of the at least one network;

wherein the computer-product product is produced by a process that includes:

obtaining a training dataset that includes at least a collection of network
5 configuration files;

training a large language model (LLM) using the network configuration files to thereby obtain a trained LLM that provides, as output, to at least one received instruction, a respective response comprising specific configuration data extracted from the configuration files.

10 23. The computer-program product of claim 22, wherein obtaining the training dataset comprises:

generating a training dataset for training the LLM that comprises synthetic configuration files.

15 24. The computer-program product of claim 23, wherein the generating includes generating the synthetic configuration files comprising:

manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of configuration files.

20 25. The computer-program product of claim 24, wherein manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining configuration files that contain errors.

25 26. The computer-program product of any one of claims 22 to 25, wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and human annotated configuration files.

27. The computer-program product of any one of claims 22 to 26, wherein generating the training dataset comprises providing, as input, multiple instruction-response pairs, and training the LLM to provide, in response to a given instruction, a respective response comprising specific configuration data extracted from the configuration files.

28. The computer-program product of claim 27, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

29. The computer-program product of claim 27, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

30. A computer-implemented method of training a large language model (LLM) for automatic discovery of at least one network, wherein the LLM can be applied on a collection of configuration files obtained from at least one network, and provide, as output, configuration data extracted from the configuration files;

wherein the training comprises:

generating synthetic configuration files and adding the synthetic configuration files to a training dataset.

31. The computer-implemented method of claim 30 comprising generating the synthetic configuration files, comprising:

manipulating parameters of a network design software to thereby obtain a diverse collection of network designs; and

executing the network designs to thereby obtain a respective diverse collection of

configuration files.

32. The computer-implemented method of claim 31, wherein manipulation of parameters of network design software includes intentionally inserting errors into network designs in the diverse collection of network designs, thereby obtaining
5 configuration files that contain errors.

33. The computer-implemented method of any one of claims 30 to 32, wherein generating the training dataset for training the LLM comprises assembling the synthetic configuration files with proprietary configuration files and human annotated configuration files.

10 34. The computer-implemented method of any one of claims 30 to 33, wherein generating the training dataset comprises providing, as input, multiple instruction-response pairs, and training the LLM to provide, in response to a given instruction, a respective response comprising specific configuration data extracted from the configuration files.

15 35. The computer-implemented method of claim 34, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a regular expression adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

20 36. The computer-implemented method of claim 34, wherein the multiple instruction-response pairs include instruction-Regex pairs used for training the LLM to automatically generate, responsive to determining that a configuration file is of a certain type, a computer program code adapted for parsing the configuration file and retrieving from the configuration file the respective configuration data.

25 37. The computer-implemented method of any one of claims 30 to 36 comprising applying a specialized tokenizer on the configuration files configured to use unique delimiters tailored to syntax and structure of configuration files.