NetAssistant: Dialogue Based Network Diagnosis in Data Center Networks

Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, Weirong Jiang, and Yi Wang, *ByteDance Inc*

https://www.usenix.org/conference/nsdi24/presentation/wang-haopei

This paper is included in the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Open access to the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation is sponsored by

NetAssistant: Dialogue Based Network Diagnosis in Data Center Networks

Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin,
Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu,
Wei Zhou, Yongbin Dong, Weirong Jiang, Yi Wang
ByteDance

Abstract

In large-scale data center networks, answering network diagnosis queries from users still heavily rely on manual on-call services. A widespread scenario is when network users query whether any network issue is causing problems with their services/applications. However, this approach requires extensive experience and considerable efforts from network engineers who must repeatedly go through lots of monitoring dashboards and logs. It is notoriously slow, error-prone, and costly. We ask: is this the right solution, given the state of the art in network intelligence?

To answer, we first extensively study thousands of real network diagnosis cases and provide insights into how to address these issues more efficiently. Then we propose an AI enabled diagnosis framework and instantiate it in a task-oriented dialogue based diagnosis system, or colloquially, a chatbot, called NETASSISTANT. It accepts questions in natural language and performs proper diagnosis workflows in a timely manner. NETASSISTANT has been deployed and running in the data centers of our company for more than three years with hundreds of usages every day. We show it significantly decreases the number and duration of human involved oncalls. We share our experience on how to make it reliable and trustworthy and showcase how it helps solve real production issues efficiently.

1 Introduction

Providing high visibility is increasingly challenging in modern hyper-scale and heterogeneous data center networks. One fundamental and essential service is handling network diagnosis queries from network users. As networks continue to grow in scale, speed, and link utilization, it has become increasingly challenging for human operators to manually monitor and diagnose network issues. Traditional approaches to network management, which rely on human operators looking at a screen of data to understand the network state, are no longer sufficient. Instead, network management solutions have become more automated than before, with a shift towards software-defined networking (SDN) and automated processes that dynamically drill down based on current conditions and even automatically react to network events. These queries normally come from cloud applications/services generating anomalies which are likely caused by network incidents. For instance, colleagues from the advertisement recommendation team observe network timeout exceptions that lead to the crash of their machine learning jobs. They are eager to ascertain whether it is a network issue and the scope and severity of the issue so that they can make business decisions such as waiting for auto-recover, changing to another backup computing cluster, or temporarily downgrading the recommendation algorithms (e.g., using a simple client side algorithm instead of the server side one).

However, handling these network diagnosis queries has always been time consuming and labor intensive work in large data center networks. The reason behind this comes from both the network user side and the network engineer side. From the perspective of network users, limited by the network domain knowledge and necessary permissions, they normally lack a network-wide view and are not able to access a variety of network monitoring primitives. As a result, network users often need to turn to network engineers for manual assistance. We have a user behavior study to investigate what exactly the needs of network users are in their daily work (Section 2.1). From the perspective of network engineers, they have a variety of network monitoring primitives to monitor, a large amount of monitoring data to process, and too many small but not negligible network incidents to investigate (Section 2.2). As a consequence, extracting useful information from the enormous data and replying to network users becomes highly tedious and time consuming work, and highly relies on the expertise of each network engineer. Therefore, we argue that there is a **gap between the network diagnosis needs of network users and network monitoring primitives in big data center networks.** Network engineers are working hard to bridge the gap with their extensive experience and expertise. And big cloud companies have a considerable number of

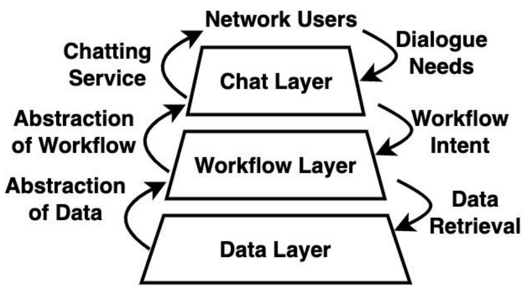


Figure 1: Level of Abstraction in NETASSISTANT

network engineers [11] who take turns to undertake the oncall services.

Researchers have built several querying [13, 15, 19] and diagnosis [12, 16, 17, 20, 23, 25, 28] tools to bridge part of the gap. However, building such automation tools has limited effectiveness since they are ad-hoc, not scalable, and introduce an extra learning curve to network users. Network users may not know these tools, or are not able/willing to learn them. Consequently, we believe a comprehensive solution should be general-purpose, query format independent, easy to leverage the O&M (operation and maintenance) experience from network engineers, and comply with performance and data scale requirements.

Inspired by the technique of task-oriented dialogue systems in the NLP (natural language processing) domain and the top-down idea in network telemetry [24], we come up with an idea to build a virtual assistant (chatbot) to take the networking related queries in natural language and leverage the experience from network engineers to perform proper diagnosis functions. While building a dialogue system/chatbot to help users/customers is already a common practice in many companies ([1, 3, 4, 10]), it is not trivial in the data center networking domain due to the requirement of high accuracy, scalability and efficiency. As shown in Figure 1, we divide the objectives into three levels of needs. Network users have the chat needs from the chat layer. Then the chat layer requires appropriate diagnosis functions from the workflow layer. The workflow layer needs efficient data retrieval from the network monitoring primitives. Conversely, each layer provides a corresponding abstraction to its upper layer. We notice that each layer encounters unique challenges as follows:

- First, networking diagnosis requires precise description and input, while a considerable proportion of queries from network users are not clear enough (Section 2.1). Therefore, it is hard to build a natural language understanding module since any misunderstanding or misconfiguration may cause immeasurable losses.
- Second, building suitable and high-quality methodologies for different diagnosis cases is challenging. We argue that network engineers have the most say on how to select and customize the diagnosis functions based on their experience and expertise. How to automatically leverage the know-how from network engineers and convert it to executable workflows is not trivial.
- Third, the performance bottleneck of a chat system for network questions mainly comes from the retrieval of underlying data. Considering the various types of network monitoring primitives and the huge volume of monitoring data, we need a comprehensive solution for data storage and data retrieval.

We propose our solution, NETASSISTANT, which leverages the technique of task-oriented dialogue systems to provide virtual assistant services to network users. NETASSISTANT takes the queries from network users in natural language, selects appropriate processing workflows that are created by network engineers, and responds to the users in a timely manner. NETASSISTANT contains three novel functional modules to achieve the three layers of abstraction and address the challenges. NETASSISTANT is deployed on top of a large scale distributed network monitoring infrastructure and deals with terabyte level of monitoring data every day.

In summary, we make the following contributions:

- We provide a user study of network users, a quantitative study of network incidents, and monitoring primitives in our production network.
- We propose an end-to-end distributed system, NETASSISTANT, which can make use of the O&M experience of network engineers to answer network diagnosis questions from network users. NETASSISTANT has been deployed and iterated in our production network for over three years and has tens of thousands of usage.
- We share our experience and lessons learned in customer service on the front line for network users.

We construct our paper as follows. Section 2 provides a network user behavior study, a quantitative study of network incidents and monitoring primitives, and motivates this paper. Section 3 describes the system design of NETASSISTANT. Section 4 provides deployment details, case studies, and evaluation results. In Section 5, We share interesting lessons we have learned from our experience of providing network diagnosis and troubleshooting functions. Section 6 and 7 describe related work and conclude the paper.

2 Measurement Study and Motivation

A typical interaction between a network user and a network engineer is that the network user raises oncall questions while the network engineer on duty answers them. The oncall process is time consuming and labor intensive work, and this paper aims to provide a comprehensive solution to help both network users and engineers. Thus, the first step is to understand the needs of network users and the challenges faced by network engineers. In order to accurately grasp the most real and urgent needs of network users, we leverage years of network oncall history and propose a network user behavior study. To better understand the difficulties and challenges of network diagnosis from the perspective of network engineers, we conduct quantitative analysis of network incidents and monitoring primitives collected from our data centers.

2.1 Network User Behavior Study

We believe grasping the most real and urgent needs of network users is the first step to design a good network diagnosis tool. To achieve this, we leverage the daily network oncall records[1] in our cloud networks, and conduct a user behavior study. These oncall records include network related questions (containing both underlay and overlay network) raised by network users and processing dialogues between network users and engineers.

By Type		**By Objectives**	
Configuration Request	2196	DC/AZ Related	1867
Host Network Issue	837	Host(S) Related	4557
Network Consultation	3073	Link Related	462
Others / Void Oncall	1083	Device Related	113
Total Count	7189	Others	190

Table 1: Network User Behavior Study

Table 1 shows the statistics about the collected oncall records. We collect about 7000 network oncall records from our oncall system and try to categorize them by question types and objectives. We observe that, by question types, most questions can be categorized into **Configuration Request**, **Network Issue**, and **Network Consultation** types. Configuration requests are to seek manual assistance in network related configurations such as BGP, firewalls, gateways, and virtual switches. For example, frequent requests include virtual IP/public IP assignment, firewall allow-list setup, and BGP configuration setup. Host (physical server, virtual machine, etc.) network issue questions are raised when users observe obvious network issues with their host and seek quick fixes. Network consultation questions ask about network health status, which is the category with the highest proportion. The most commonly asked question is "*Is there any problem with the network in xxx*". We believe that answering most of these three types of questions can be automated, while some dangerous operations should be manually confirmed before being taken into action.

We also analyze the oncall questions from their objectives. As summarized in Table 1, most questions are in the dimension of host level, including physical hosts, virtual machines, IP pairs, subnets, and clusters. Besides, questions related to DC (data center) and AZ (available zone) also occupy a certain proportion. The reason behind this is that questions from network users usually originate from the servers they use or the data centers they are aware of.

[1]This user behavior study only includes data center network oncall questions. But our tool also supports other types of networks, e.g., IT/office networks, edge/CDN networks, etc.

We also conclude that network users have the following behavioral characteristics when submitting an oncall question.

- First, the questions from network users are normally very broad, e.g., the network health status of a large cluster with thousands of servers or an entire data center.
- Second, the objectives of the questions are sometimes very vague. While data center networks have a strict and accurate naming specification, most network users are unaware of that. For instance, one user may ask about the network health in Santa Clara, which refers to an AZ named US-WEST-1 in the specification[2].
- Third, some questions are even missing a clear intent. Sometimes, users are very anxious to seek manual services and may ask questions like "*Any network incident right now?*" or "*US Redis service is lagging.*" It is hard to understand directly from their questions which part of the network they are asking.

Our tool is designed to take, understand, and respond to network users who will ask similar questions as in oncalls. In Section 2.3, we will discuss the corresponding research challenges in terms of question understanding and processing.

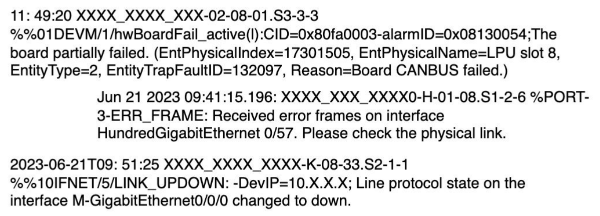
2.2 Network Incidents and Monitoring Primitives Study

Handling network diagnosis requests is challenging for network engineers since the volume of monitoring data and the detected network incidents is huge. Network engineers need to go through a large amount of data to find the possible root causes. Therefore, network diagnosis becomes a time consuming and label intensive work. In order to better understand the difficulties and challenges, we leverage our detected network incidents and collected network monitoring data in our production network to provide a measurement study.

First of all, we aim to know how frequently network incidents happen in daily operations. We define network incidents as **any network abnormal behaviors that could violate the service-level agreement (SLA) of the network.** More specifically, network incidents include abnormalities of the metrics (e.g., packet drop, latency, bandwidth) that can be perceived by the end users and abnormalities monitored from the network components. Network components can be hardware (e.g., switch, circuit, optical module, and network card) or software (e.g., operating system, virtual switch, and configuration). The number of daily network incidents can illustrate the complexity and effort of diagnosing network failures.

In order to avoid bias caused by subjective factors like threshold, we choose the switch running errors/exceptions, which are detected by switch syslog to represent network incidents for the quantitative study. Syslog [8, 9] is collected

[2]We use our venue city name to represent a data center.



```
11: 49:20 XXXX_XXXX_XXX-02-08-01.S3-3-3
%%01DEVM/1/hwBoardFail_active(l):CID=0x80fa0003-alarmID=0x08130054;The
board partially failed. (EntPhysicalIndex=17301505, EntPhysicalName=LPU slot 8,
EntityType=2, EntityTrapFaultID=132097, Reason=Board CANBUS failed.)
            Jun 21 2023 09:41:15.196: XXXX_XXX_XXXX0-H-01-08.S1-2-6 %PORT-
            3-ERR_FRAME: Received error frames on interface
            HundredGigabitEthernet 0/57. Please check the physical link.

2023-06-21T09: 51:25 XXXX_XXXX_XXXX-K-08-33.S2-1-1
%%10IFNET/5/LINK_UPDOWN: -DevIP=10.X.X.X; Line protocol state on the
interface M-GigabitEthernet0/0/0 changed to down.
```

Figure 2: Sample Syslogs Indicating Switch Exceptions

from switch logging services and contains crucial running information of switches. Sample syslogs are shown in Figure 2, which indicates switch hardware abnormalities like board/port/link failure or software abnormalities like protocol (e.g., BGP, ISIS) down/flapping[3]. We choose one of our data centers containing about 14,000 physical switches and count all syslog errors/exceptions for the **entire month** of June in 2023. The statistics are shown in Table 2.

Error Type		Error Type	
Total Count	118,123,984	Linecard Issue	20,690
Queue Issue	57,036,587	Process Exception	83,465
Interface Issue	44,662,097	Frame Issue	2,280,956
Protocol Issue	8,360,567	Others	5,679,622

Table 2: Syslog Statistics for One Month

We can observe that even for just one data center, we encounter around 4 million different kinds of switch abnormalities per day. While most abnormalities will auto-recover very soon and most are in the packet/frame level, it is still annoying for network engineers since it is hard to tell whether the abnormalities are user-insensitive or not. Correspondingly, when network users experience network anomalies, they also need network engineers to help them diagnose and demarcate within such a huge number of switch abnormalities.

Other than the switch syslog, there are also several other types of primitives that monitor the status of different components of the network. For example, we track the packet drop and latency metrics by using the connectivity monitoring primitives and network traffic volume and composition by using the traffic monitoring primitives. We conduct another measurement study for the daily data volume of each type of monitoring primitives for the same data center. The results are shown in Table 3.

From the statistics shown in Table 3, almost every category of monitoring primitives has a huge amount of data every day. Meanwhile, most diagnosis requests from network users often require a combination of different types of data to analyze. For example, we may use connectivity monitoring data (e.g., PingMesh [18]) to check whether there is any connectivity issue and locate the IP pairs of abnormal traffic and then use traffic monitoring data (e.g., sFlow) to further troubleshoot the packer headers to find the root cause. This puts a huge burden on network engineers to retrieve the data reactively. Therefore, an automated tool that can perform various diagnosis procedures will be helpful and unleash network engineers.

[3]While different vendors have different syslog formats, we parse and normalize the syslog according to their specifications.

Monitoring Primitive Category	Data Volume per Day
Connectivity (e.g., PingMesh [18], EverFlow [27], etc.)	65GB
Traffic (sFlow, SNMP, etc.)	12TB
Switch Syslog	35GB
Host Monitoring	4.3GB
Routing Configuration	425G
Optical Module (DDM or DOM)	5.5GB
Other Monitoring Primitives	27GB

Table 3: Data Volume of Monitoring Primitives per Day

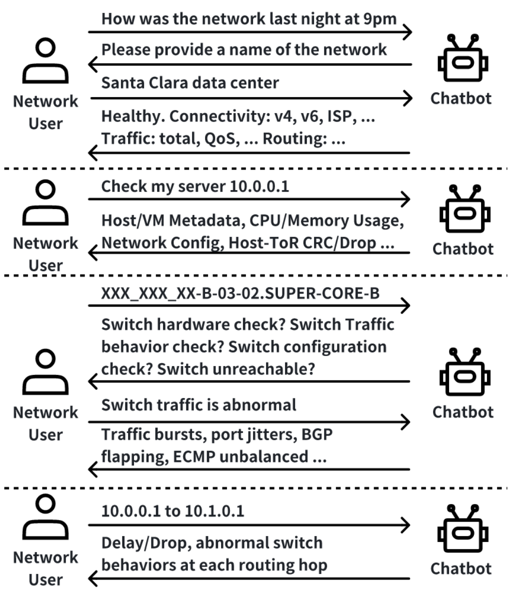
2.3 Motivation and Research Challenges

We aim to automatically answer questions from network users. An existing idea is the text-to-SQL [22] parsing solution, where the natural language question could be converted into a SQL query. However, this solution is limited in our scenario since data retrieval is just one step in the overall diagnosis process and our network monitoring infrastructure involves variety of data sources in addition to relational databases, for example, non-SQL database, file system, message queue, SSH agent, etc. Therefore, our idea is to leverage NLP technique to map user queries to pre-defined workflows. As illustrated in Figure 1, we model the whole system into three abstraction layers with their unique challenges.

The **chat layer** is responsible for providing a typical task-oriented dialogue service for network users. However, it is challenging since network diagnosis requires precise input while the questions from users are arbitrary (as shown in Section 2.1). This places three requirements on the NLU (Natural Language Understanding) module. First, it should be able to cover various granularity of data center networks, from global regions to small network components. Second, besides the standard terminologies that are strictly defined in the specifications, the NLU module should also understand most common expressions that people might use to describe data center networking in everyday conversations at work. Third, when word slots or the intent is not clear, the chatbot should leverage the dialogue function to guide users to provide more information or ask more precise questions.

Figure 3 illustrates the ideal dialogue service we aim to provide to network users. Our chatbot performs corresponding diagnosis functions to answer user questions if both objectives and the intent are clear. If not, our chatbot will use dialogue to guide network users to provide more information

The **workflow layer** is responsible for providing and performing diagnosis functions for the chat layer. We argue that network engineers have the most say on how to diagnose and troubleshoot network questions as they have extensive expe-



How was the network last night at 9pm

Please provide a name of the network

Santa Clara data center

Healthy. Connectivity: v4, v6, ISP, ... Traffic: total, QoS, ... Routing: ...

Check my server 10.0.0.1

Host/VM Metadata, CPU/Memory Usage, Network Config, Host-ToR CRC/Drop ...

XXX_XXX_XX-B-03-02.SUPER-CORE-B

Switch hardware check? Switch Traffic behavior check? Switch configuration check? Switch unreachable?

Switch traffic is abnormal

Traffic bursts, port jitters, BGP flapping, ECMP unbalanced ...

10.0.0.1 to 10.1.0.1

Delay/Drop, abnormal switch behaviors at each routing hop

Figure 3: Sample Dialogues between Users and Chatbot

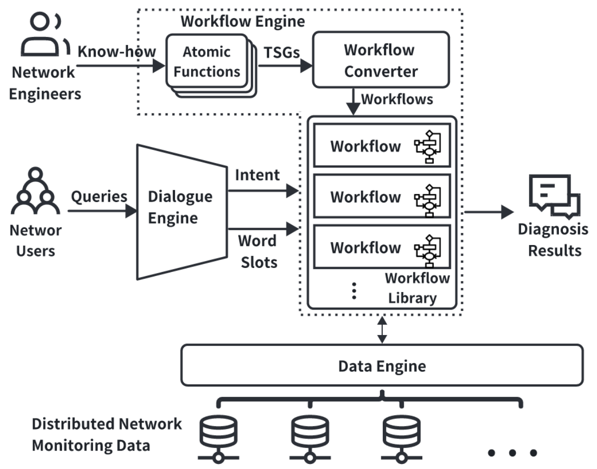


Figure 4: System Design of NETASSISTANT

rience in diagnosing and dealing with network problems on the front line and have a deep understanding of the specific network situation, e.g., hardware vendors and models, network architecture, common issues, and user traffic patterns. Inspired by this, our tool introduces a novel functional module to collect and convert the experience from network engineers into a considerable number of executable workflows.

The **data layer** is responsible for providing the abstraction of underlying monitoring data. The performance bottleneck of our chatbot mainly comes from the retrieval of distributed and large volume underlying monitoring data. Reactively analyzing the data takes a long time and is unacceptable for a chatbot. We come up with a solution that combines on-demand querying and proactive alerting to achieve a good trade-off between the response latency and the timeliness of diagnosis results.

3 System Design

In this paper, we propose a virtual assistant tool, named NETASSISTANT, to answer the network diagnosis questions in natural language from network users. The key idea behind is that NETASSISTANT aims to understand what intent and objects network users are asking about and makes responses using corresponding predefined workflows, which are learned from network engineers. To this end, NETASSISTANT is designed as three-fold. First, in offline, NETASSISTANT automatically converted the O&M knowledge and experience from network engineers into multiple executable workflow functions. Second, in runtime, NETASSISTANT analyzes the intent and word slots of user queries, forwards this information as parameters into corresponding workflow functions, and responds to the users with the outputs of the function. Third, NETASSISTANT applies several optimization and trade-offs of the underlying monitoring data storage and retrieval component to meet the performance needs as a real-time chat assistant. In this section, we describe the design and implementation details of NETASSISTANT.

3.1 System Architecture

NETASSISTANT works as an always-on service for the data center networking environment. As shown in Figure 4, NETASSISTANT consists of three main functional modules: 1) **Dialogue Engine**, which provides a dialogue environment for users and understands the intent and word slots from the user queries; 2) **Workflow Engine**, which converts the knowledge and experience from network engineers into workflows and processes proper workflows for each user query; 3) **Data Engine**, which manages all underlying distributed network monitoring data storage and provides high-performance data retrieval. In the following subsections, we describe the design details of each functional module.

3.2 Dialogue Engine

Dialogue Engine module provides network users with a dialogue environment that presents a conversational experience with multiple rounds of question and answer. The input from the user is plain text in natural language. The output form is relatively rich, which can be rich text, pictures, interactive components, and hyperlinks. We leverage the typical framework of a task-oriented dialogue system, and the key challenge is to build the Natural Language Understanding

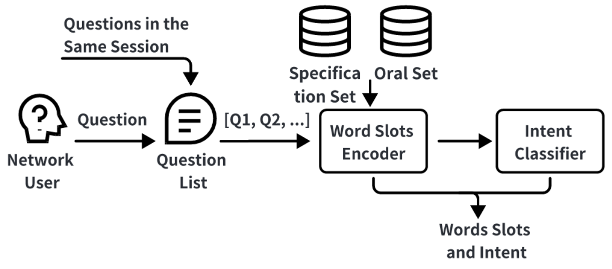


Figure 5: Dialogue Engine Module

(NLU) component. The key target of the NLU component is to understand the **word slots** and **intent**. The intent of each question will decide which diagnosis function (workflow) that we will use, and the word slots will be the input parameters. Figure 5 shows the detailed processing steps for language understanding, which will be explained in the following.

The Dialogue Engine provides session based dialogue management to manage user questions. All user questions belonging to the same session will be converted into a question list for understanding. The first step for understanding is to understand and parse the word slots from the questions. The word slot here could be the name of any component in the data center networks. Therefore, we collect nouns from our production network and categorize them into different types according to their role in the networks, e.g., region, country, state/province, AZ, office, building, pod, subnet/cluster, server, switch, circuit, and optical module.

We put the words into two word sets. The first word set contains terminologies strictly defined in our datacenter network specification, which we call the **Specification Naming Set**. The second word set contains the terminologies and their oral expressions, which we call the **Oral Naming Set**. The Specification Naming Set is a subset of the Oral Naming Set. We build a multi-to-multi mapping between the two sets. It is because one terminology may have multiple oral expressions (both "Santa Clara" and "US West" refer to "US-WEST-1"), and sometimes there may have multiple AZs in the same place (both "US-WEST-1" and "US-WEST-2" locate in "Santa Clara").

We build these two sets for the word slots parsing, standardizing, and encoding. We first use the Oral Naming Set to parse the word slots from the user questions. Then, we use the mapping between the two sets and the Specification Naming Set to standardize each word. After that, we encode each word by using its category name. For example, we encode the word "Santa Clara" with "available_zone_#14" from the original sentence for further intent understanding and parse a parameter {"az_1": ["US-WEST-1"]} for workflow processing.

Then, we process the encoded questions for intent understanding (i.e., text classification). We iterate through two technical solutions for text classification. The first one is based on word-level Convolutional Neural Network (or ConvNet, CNN) [26]. Motivated by the technological breakthrough in the area of Large Language Model (LLM), our second solution is an LLM-based solution. We will detail the two technical solutions in the following.

Word-Level CNN-Based Solution. Our idea is to leverage a training based classifier to classify the category/intent of the given questions. To achieve this, we collect a considerable number of questions for each intent as the training dataset for a word-level CNN classifier offline. In the runtime after the encoding process, the classifier generates a classification score of the text for each intent and selects matched intent(s) based on a predefined threshold.

LLM-Based Solution. Our idea is to leverage the powerful comprehension skills of the LLM to do multiple-choice questions. We use the technique of *few-shot prompting* to add in-context learning where we demonstrate different intents in the prompt and let the language model make selections. One sample of the prompt is:

Intent check_switch_traffic is to check the traffic indicators of a switch. Intent check_host_network is to ... Please select corresponding intents for the following questions: "My devbox is unreachable. VM host_ip_#19."

Since we already have >100 intents, which will make the generated prompt sentence too long, we would like to select a relatively smaller set of intents (around 10) to shorten the prompt. To achieve this, we apply *vector embedding* to both the user questions and the description of intents and utilize a similarity based searching method to select the intents.

Based on the classification results of the user questions, there could be four different cases. 1. If both the intent and words (parameters) are clear, the chatbot will proceed with the selected diagnosis function. 2. If the intent is clear, but some words are missing or have more than one possibility, the chatbot will indicate the user to supplement this part of the information. 3. If the classifier selects more than one intent, the chatbot will indicate the user to select one. 4. If no intent is selected, the chatbot will respond with the manual page containing how to use the chatbot and popular examples.

3.3 Workflow Engine

The Workflow Engine module provides network engineers with the framework to create multiple workflows based on their expertise. The module consists of three primary components: atomic functions, workflow converter, and workflow library, which work seamlessly together to enable efficient workflow creation and execution. The expressive atomic functions allow network engineers to create simple, flexible, and interactive diagnosis logic. The workflow converter converts the logic into executable workflows, and the workflow library supports triggering and executing the workflows at runtime.

Atomic functions: The atomic functions form the build-

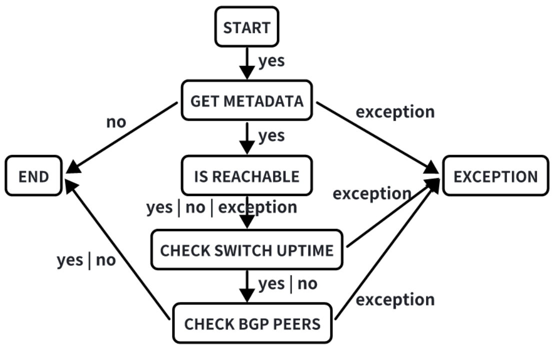


Figure 6: Example of TSG for Switch Reachability Check

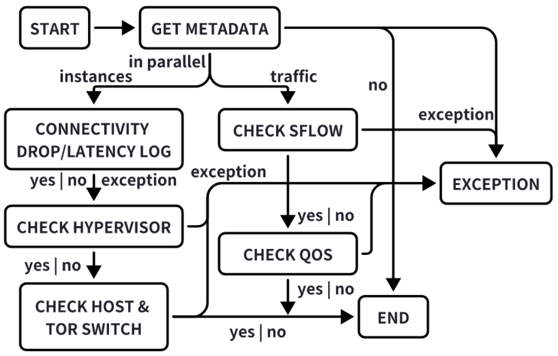


Figure 7: Example of TSG for Storage Service Check

ing blocks of our workflow system, with each encapsulating a particular method unique to a specific network entity and its metrics data. Entities could include switches, links, interfaces, available zones, and more. The decision-making method checks the metrics data of the entity over a time range and determines if there is any abnormality. Additionally, atomic functions may include an optional action process to analyze specific scenarios within the workflow, such as pausing a service on a switch to check if it caused overload or triggering a traceroute to display possible routing paths.

For all common network components, we tailor a set of atomic functions based on the monitoring primitives to meet various diagnosis requirements. For example, based on the sFlow data of a link, we can check if the overall utilization is high, or if there is any kind of traffic violating the QoS rules, or if there is any service traffic experiencing a sudden increase or decrease. There may be multiple algorithm implementations for the same function. The threshold details can be customized based on the experience and expertise of network engineers.

Workflow converter: Network engineers can configure a set of atomic functions to become a **Troubleshooting Guide** (TSG). A TSG comprises multiple atomic functions that allow specifying the data entity to be analyzed within a given time range using decision-making logic. The TSG operates like a state machine, existing at any given point at a specific stage (atomic function). The outcome of each atomic function determines the transitions between stages.

Network engineers can configure these TSGs through a visual interface. The configuration includes the selection and threshold of atomic functions, arranging them in to a flowchart and specifying the input such as switch names or time periods. It is possible to customize different versions of a TSG with the same name depending on the input value. For example, switches from different vendors may have different *check_switch_hardware* TSGs. TSGs can be updated and adjusted dynamically with the change of network architecture.

Workflow Engine then converts the TSGs to executable workflow logic and adds essential exception handling, supplementary background knowledge, associated network events, and common help entries. For instance, all workflows will be added the logic to check if there are any known/ongoing network issues or changes and recommend other tools for non-network issues. Executable workflows are stored in the workflow library.

Workflow library: The workflow library is responsible for selecting and executing proper workflows based on the extracted intents and parameters. Some diagnostic workflows are only available to specific people (e.g., specific network engineers). Thus, we apply role-based access control before executing the workflows. We list the most commonly used workflows in Section 4.2. All monitoring data access will go through the Data Engine which we will describe in Section 3.4.

Figure 6 and 7 presents two simplified examples of TSGs designed to check switch reachability and storage service health (e.g., a SQL cluster). Respectively, upon receiving the switch name or service name as input, the workflows consists of multiple atomic functions, including getting metadata, retrieving monitoring data, detecting abnormalities and making judgments. Finally, if all checks pass and no abnormality is found, the entity is appropriately marked as healthy.

3.4 Data Engine

In order to track various aspects of the network, we utilize multiple monitoring primitives, as outlined in Section 2.2. The Data Engine is responsible for providing the abstraction and query entry of network monitoring data. However, as shown in Table 3, extensive analysis of large data sources is impractical, such as analyzing the last X minutes of sFlow data across multiple links. This is the performance bottleneck of the entire system.

The Data Engine utilizes two strategies to ensure efficient data retrieval. Firstly, data sources that are relatively low in volume, such as switch configurations, can be stored and analyzed in their current form. These sources do not change frequently, so analyzing them in real time is feasible.

However, high volume data sources are more challenging to manage. While the data generated may contain a significant amount of normal, expected behavior, only a small fraction of it is valuable for diagnosis. As such, we use a proactive approach to monitor the data for anomalies, which are flagged as alerts. We move the execution of some atomic functions that use high volume data sources to become always-on operators generating alerts. The alerts are then tracked, and when we need to query high volume data sources, we retrieve the alerts instead of the full data. For example, only a small portion of syslog data, namely BGP flapping or line card errors, can actually help diagnose underlying network issues. For connectivity monitoring data, normally, only jitters that violate the SLAs will get the attention of network engineers.

Trade-off: Leveraging the proactive alerts achieves a trade-off. We reduce the response time for generating diagnosis results but sacrifice some timeliness since generating alerts is not real-time but periodic and increases some computational burden. We have set the following criteria to determine which data sources are suitable for proactive alerts. First, the data sources should have high volume and only the abnormality is valuable for diagnosis. Second, the quality of data sources should be high enough that the network team may directly get involved in the discovered alerts. Third, the latency increased by the alerts should not violate the service level objectives of incident discovery and incident diagnosis promised by the network team. For instance, the network team promises to detect network issues within 2 minutes, consisting of a delay from monitoring data and a delay from generating alerts.

Implementation of Alerting System: We implement and deploy the alerting system as a standalone distributed system which consumes real time monitoring data and generates **alerts** such as *batch BGP peers down*, *interface flapping*, etc. The alerting system is responsible for generating, aggregating, prioritizing and publishing alerts, and managing the life cycle of each alert. The Data Engine of NETASSISTANT is one of the consumers and it subscribes to a subset of the generated alerts. The alert generating processes are implemented as streaming tasks in the monitoring data collection pipeline before the data storage. This design reduces the overhead of heavy data I/O compared with frequent database queries. The streaming tasks subscribe to the monitoring data from message queues (e.g., Kafka [2]) and utilize a sliding window for periodic detection. The period is usually set to a minute level due to our promises about the service level objectives. Some high priority alerts will directly engage the network team and trigger a fix, which we will discuss this situation in Section 5.1.

4 Evaluation

In this section, we first provide the implementation and deployment details of NETASSISTANT. Then, we share several interesting case studies. At last, we present the evaluation results of our tool in terms of usage, accuracy, and performance.

4.1 Implementation and Deployment

We describe the implementation and deployment details of NETASSISTANT, including Dialogue Engine, Workflow Engine and Data Engine. The Dialogue Engine consists of client-side and server-side. The client-side is built on top of a commercial business chat and collaboration tool as a chatbot application. The chatbot application is able to access content, collect information, conduct operations, and support interaction through messages in both private chat and group chat. The server-side implements NLU and is deployed in a cluster of three server machines. For the NLU component, we collect around 1,000 data entries for each workflow to train the CNN-based classifier and leverage a 13-billion-parameter large language model. The Workflow Engine is deployed in a computing cluster with 14 instances in different data centers for the consideration of disaster tolerance and high throughput. It also contains a front end for the network engineers to create and configure TSGs. The Data Engine is deployed using a server alongside the monitoring data storage in every available zone of our network. The whole system was launched in April 2020 as an always-on company wide service. And it has been continuously iterated in terms of techniques and supported workflows.

4.2 Case Study

We share several diagnosis cases, including user side behavior, diagnosis workflow, and follow-up actions. We also summarise the most commonly used workflows in daily work.

4.2.1 Case 1: Host Network Issue

User Question: A user queried an IP address as her SQL cluster encountered a connection exception to this instance.
Diagnosis Workflow: The bot identified it as an IP of a virtual machine and performed the *check_iaas_ip_network* workflow. After conducting a multi-dimensional diagnosis on this virtual machine, the workflow found that the link between its physical machine and the ToR (Top-of-Rack) switch started to have considerable CRC errors from the SNMP monitoring data, and the ToR switch also reported CRC errors from syslog. Thus, the bot responded with an unhealthy diagnosis conclusion and informed the current oncall network engineer about this issue.
Follow-up Actions: The CRC issue was fixed right after cleaning up the port on the switch.

4.2.2 Case 2: Data Center Network Issue

User Question: A user reported a suspicious network issue in a data center at around 2 AM since her managed service had a wave of failures, and the log showed many connection errors.

Diagnosis Workflow: The bot performed the *check_idc_network* workflow and found that there was a spike of packet drop and latency increase, which lasted for around 2min according to the connectivity monitoring results. Involved IP pairs can be aggregated into a /48 IPv6 subnet. The workflow further noticed that there were several BGP withdrawn events whose peers pointed to the same switch and several ECMP imbalance exceptions from nearby switches syslog. Thus, the bot made a conclusion that there was a short-lived network issue and the root cause was due to a switch down event.

Follow-up Actions: The bot responded with the diagnosis results, estimated impact and suggestions to the user. Network engineers further confirmed the switch was down and isolated the switch. However, the bot did not see switch unreachable alert from the switch reachability monitoring results the first time, and then the alert was 2 hours late. The network engineers further found that the switch reachability checker only checked the switch management port, which in this case was not down immediately, but the loopback port was down. So, they learned from this case and updated the corresponding TSGs with new switch reachability checking logic.

4.2.3 Case 3: Large Scale Network Failure

User Question: To investigate the cause of performance degradation in a core service, a user queried the network status of her service.

Diagnosis Workflow: The *check_computing_service* workflow discovered multiple ping drops and high latency alerts between regions, along with configuration changes that were made to bring up a plane. The workflow also found the network team had already noticed and engaged in this issue in advance and had an initial conclusion that the configuration change might be the root cause. Therefore, the bot responded with the conclusion from the network team and suggested a downgrade to the service algorithm.

Follow-up Actions: Further investigation by the network engineers revealed that the misconfiguration causing the issue was not on the recently updated devices but rather in previously misconfigured route maps on a Point of Presence (PoP) device. This misconfiguration increased the priority of routes advertised by the PoP, leading to congestion and packet drop. To prevent such issues in the future, they plan to add a verification module (e.g., control plane and data plane verifiers) that will detect control and data plane issues.

4.2.4 Most Commonly Used Workflows

We list the most commonly used workflows in Table 4. Similar workflows are categorized together. Most DC level and IP level workflows support checking a single target or a pair (source and destination), and a time point or period is optional.

Workflows	Explanation
check_pod_network check_az_network check_idc_network check_region_network	Data center level network status workflows, including connectivity (internal, external, overlay, underlay, v4, v6, subnets and etc.), bandwidth & utilization (different types of links, different granularity), switches and existing network incidents & changes.
check_phy_ip_network check_iaas_ip_network check_vip_network check_rdma_network check_p4_network	IP level network status workflows, including software stack check, hardware status check, network environment (nearby switches) check
check_switch_reachable check_switch_hardware check_switch_traffic check_switch_config	Switch health status check, including metrics from switch OS (syslog), protocol (e.g., SNMP, BMP), hardware (e.g., linecard, OTN) and external monitors.
check_direct_connect check_bbone_link check_isp_link	Physical link level status workflows, mainly used by network team, including physical metrics, traffic and protocol status checking.
check_storage_service check_computing_service	Network service level workflow, checking network status of involved servers, upstream and downstream network traffic, QoS management and etc.

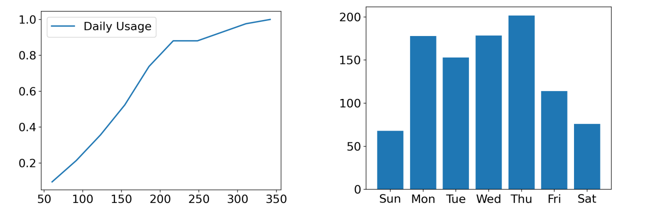
Table 4: Most Commonly Used Workflows

4.3 Usage Evaluation

We count the daily usage and the number of distinct users to evaluate the impact of our tool. We collect the usage data for two whole months of July and August 2023. A complete dialogue session counts as **one** use of our tool. The daily usage distribution results are shown in Figure 8. We can observe that there are approximately 100 to 200 uses per day on most days. The distribution by day of the week is interesting. We find that the average usage on weekends is much less than on workdays. This is due to big changes (including software changes and network changes) are not allowed on weekends so that the entire data center network is running relatively stable. We also count and analyze what kinds of users are using our chatbot services. For July and August 2023, there are a total of 476 individual users. Most of them are from the engineering team, e.g., site reliability engineers, computing or storage infrastructure engineers, and network engineers. When there is a big network outage, there will be a wave of heavy usage (normally 20-50) of our chatbot since many people get affected, and they would like to know the situation and impact.

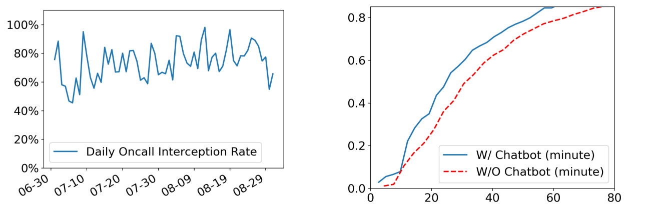
Next, we will measure how our tool reduces the oncall workload for network engineers. Basically, there are three different cases. The first case is that a user does not raise any oncall after using the chatbot. The second case is that a user brings the diagnosis results from the chatbot to an oncall for further operations[4]. The third case is that a user directly raises

[4]Users can raise oncalls either from the oncall platform or through the chatbot, which will directly share the diagnosis results to the oncall.



(a) Daily Oncall Usage for July and August in 2023 (CDF)
(b) The 50th Percentile Usage by Day of the Week

Figure 8: Daily Usage Results of NETASSISTANT



(a) Daily Oncall Interception Rate
(b) Oncall Duration Time Comparison (CDF)

Figure 9: How NETASSISTANT Benefits Daily Oncall

	Mar	Apr	May	Jun	Jul	Aug
FP	9.48%	12.33%	11.6%	10.63%	9.62%	8.45%
FN	0%	0%	0%	0.43%	1.25%	0%

Table 5: Accuracy Evaluation Results of NETASSISTANT

an oncall without using the chatbot, maybe because she does not know the tool.

We name the **first case** as an **effective use** of our tool. We consider the situation where an individual user makes several **effective uses** within a short time (e.g., an hour) as a successful **oncall interception**. Based on our statistics, the number of intercepted oncalls is about one-third of the total number of effective uses of our tool. The insight behind is that one user may continuously make several queries for just one issue. We further define the **oncall interception rate** as the number of intercepted oncalls divided by the sum of the number of raised oncalls and the number of intercepted oncalls. For the second and third cases, we will compare the oncall time with and without using the chatbot to measure whether the chatbot can benefit the oncall process.

Besides the NETASSISTANT usage data, we also collect our data center network oncall records for July and August 2023 and calculate the daily oncall interception rate and average processing time. The results are shown in Figure 9. We can observe that NETASSISTANT can reduce around 50% - 90% of daily oncalls. In most reduced cases, the user gets a response of health network status from the chatbot and does not continue to raise the oncall service. For the results of oncall time with and without the diagnosis from our tool, we can find that the information provided by NETASSISTANT can shorten the average oncall time. Figure 9(b) shows that NETASSISTANT can save around 20% - 25% of an oncall time in most cases. The saved time should have been spent by network engineers going through various monitoring data. With our tool, network engineers can focus more on the issue fix and communication with the network users.

4.4 Accuracy Evaluation

As a diagnosis tool, NETASSISTANT must achieve high accuracy to gain user trust. In this subsection, we will show our evaluation results of the false positives and false negatives of the diagnosis results. False positive is a non-network issue, but the chatbot diagnoses it as a network issue. To calculate the false positive ratio, we perform manual verification of results that are diagnosed as network issues over a period of time. We find that the false positives are normally due to noise from the monitoring data. False negative is a network issue that is not detected by the chatbot. For every user-perceived network incident, we have the postmortem session that will check whether it has been (or can be) detected by our chatbot. Based on our experience, false negatives are much more harmful since the results may mislead network users and engineers and delay the progress in fixing issues.

We collect the false positives and false negatives data for half a year (March 2023 - August 2023), as shown in Table 5. We can observe that the false positive ratio is around 10%. We also find that the majority of false positives come from sampling based monitoring data (due to the nonrepresentative data as a result of chance) and switch monitoring data (especially when the switch is under high CPU utilization or software upgrade). Based on our experience, false positives are hard to avoid. Thus, we take a hierarchical approach to the network issues we detect. The chatbot will make an "unhealthy" result only for severe and wide-ranging network issues. For other detected small issues, the chatbot will make a "warning" result to inform the user that there could be a network issue. Compared with false positives, false negatives basically rarely happen. They are mostly due to problems occurring in places not covered by monitoring infrastructure. So, after every false negative happens, we will quickly improve the monitoring items and related diagnostic workflows. For example, in the second case in Section 4.2, we found a false negative generated by the switch reachability monitoring data since the switch loopback port had not been covered, and we fixed this flaw after the postmortem.

4.5 Performance Evaluation

As a dialogue system, the response latency is an important part of the user experience. To address the performance bot-

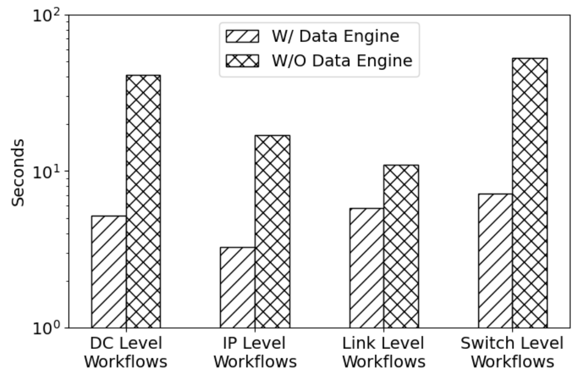


Figure 10: The 99th Percentile Running Time of Workflows

tleneck introduced by monitoring data retrieval, we design and implement the Data Engine module as shown in Section 3.4. To verify the performance improvement, we measure the response latency of the commonly used workflows listed in Table 4 with and without using the proactively generated alerts in Data Engine. We count the 99th percentile response latency for different categories of workflows. As shown in Figure 10, compared with retrieving monitoring data on the fly, leveraging proactively generated alerts can save significant time. The Data Engine reduces the overhead of big data query and transmission. After the optimization, most commonly used workflows take 5-10s for execution. Based on user experience feedback, the 5-10s response delay is acceptable but almost at the limit for a dialogue system. Considering we are still developing and adding new monitoring items to the diagnostic logic, performance optimization will be a continuous work in the future.

5 Experience Learned

NETASSISTANT is designed to help both network users who use the data center networks and network engineers who are engaged in front-line network O&M work. The user experience is the key to the success of our tool. Therefore, in addition to the iteration of the tool itself, we also provide thorough customer services including seminars, office hours, and surveys. We would like to share several lessons we have learned during the development and operation of our tool.

5.1 Can we proactively find alerts, engage the network team, and inform users?

As described in Section 3.4, Data Engine will set up operators to proactively generate alerts for high volume monitoring data. The straightforward question is, can we directly leverage the alerts to engage the network team and inform impacted network users in advance before they ask? The answer is partially yes. We only cover a small portion of high-priority alerts with severe influence. There are a few reasons. Firstly, as explained in Section 2.2, network incidents will flood the network team and users, which is impractical to manage. Secondly, the coverage and granularity of some monitoring techniques are limited by old models, device performance, and vendor support, which affects their quality. Finally, the resource budget is insufficient to proactively run all workflows, so we reserve them only for the high-priority ones.

5.2 Quality user experience

We prioritize ensuring a positive user experience and high engagement with our tool during network diagnosis and troubleshooting. To achieve this, we strongly emphasize the accuracy of our tool, which is achieved through the release of new workflows only after extensive testing. In addition, in our experience, we recognized that users often require further details beyond the initial technical diagnostic report, including information about the network issue, its service impact, and where to turn for help. To address this need, the chatbot provides users with additional details and resources. For instance, the chatbot will provide a brief explanation of the monitoring metrics and why it is (not) a network issue. Moreover, the chatbot can recommend relevant technical documents and similar problem operations and even enter an oncall or ticket request based on the question and diagnosis results.

5.3 Empowering our users

Our natural language understanding component is trained or tuned by user queries. On the contrary, our chatbot can also influence how users ask questions. We observe that after using our chatbot, user questions become more and more accurate and concise, and the communication between network users and network engineers has been dramatically simplified. For example, users get to know more network knowledge and terminologies and better descriptions of their questions. Further, the feedback and suggestions from users can help us polish our tool better. We are happy to see such an effect and think it is a virtuous circle.

5.4 Limitations and Future Work

We understand that our tool, while a crucial advancement in automated network diagnosis and troubleshooting, presents several areas for further research.

First, the state-of-the-art Large Language Model (LLM) has shown significant improvement in natural language understanding and generation, and has great potential to help automate network diagnosis and troubleshooting as a new technology. Our experience shows that LLM is good at learning, extracting and paraphrasing static information from diverse sources. For example, it can accurately and helpfully answer questions like "How does our QoS system work" and "What is the standard operating procedure for a network change". However, we find it challenging to use LLMs to answer network diagnosis questions due to their limitations in understanding and learning diagnostic logic, as well as processing real-time monitoring data. Therefore, in this project, we leverage LLM for intent understanding but still rely on pre-defined diagnosis workflows.

Secondly, the manual effort involved in building workflows hinders the complete automation of network incident detection and mitigation. Achieving full automation remains a non-trivial task. We plan to fully automate the ability to convert workflows from natural language and past incident cases/tickets. Furthermore, we expect the workflows to be self-adjusting based on feedback from users and network engineers.

Finally, network monitoring primitives form the foundation of our tool, and the quality of the diagnoses is highly dependent on the input monitoring data. The diversity of data center network monitoring data and the different implementations from numerous hardware vendors make data screening and pre-processing very arduous. In the future, we plan to enhance our tool's capabilities with selective data source selection and noise filtering to improve the quality of the results.

6 Related Work

Natural Language Processing: Our work has been heavily inspired by the dialogue systems in the NLP domain [14, 21]. A dialogue system can be categorized into task-oriented or non-task-oriented (also known as open-domain). There have been several task-oriented dialogue system products in industry [1, 3, 4, 10]. Our work attempts to leverage the task-oriented dialogue technique into the data center networking domain and addresses several unique challenges. Besides, a significant breakthrough has recently been made in the field of LLM. Products like ChatGPT [5], LLaMa [6], and PaLM [7] demonstrate strong ability in language understanding and generation. Our work leverages the understanding ability of LLM, and we are still exploring its generation ability.

Querying: Many works have improved the efficiency and effectiveness of querying and summarizing network traffic. Sonata [19] partitions each query across the stream processor and the data plane and dynamically refines each query to ensure it focuses only on traffic satisfying it. BeauCoup [15] is another programmable switch-based system. Using a coupon collector approach, it supports multiple distinct counting queries simultaneously while making only a small constant number of memory accesses per packet. Net2Text [13] uses NLP to format natural language operator queries to database SQL-like queries, executes the query, summarizes the results, and translates the summary back to natural language. We are inspired a lot from Net2Text and further leverage the dialogue system to build a general-purpose querying framework.

Diagnosis: There have been numerous efforts to develop tools that can automate fault localization. One such tool is OmegaGen [20], which combines static and dynamic analysis to track the control and data flow through a program to localize partial software failures at runtime. Another focus of automated fault localization tools is identifying the cause of packet drops. For instance, 007 [12] ranks links based on their relative drop rates using the path of TCP connections suffering from one or more retransmissions. On the other hand, Drift-Bottle [28] takes a more distributed approach, with each switch using the status of flows to infer suspicious links and add lightweight inference headers to packets sent to the operator. VTrace [16] focuses on detecting persistent packet loss over the cloud-scale overlay network. It utilizes the "fast path-slow path" structure of virtual forwarding devices to track and inspect the packets of interest in depth selectively. Zeno [23] generates a causal graph to capture the temporal dependencies between events in a system, such as requests and their corresponding responses, to diagnose performance problems. CloudCanary [25], on the other hand, uses fault graphs to perform real-time audits on service updates to identify the root causes of correlated failure risks and generate improvement plans with increased reliability. Tools like Scouts [17] use machine learning to analyze complex relationships and route incidents to the most likely responsible team. We benefit from these works by allowing network engineers to autonomously select and customize diagnostic algorithms to build workflows.

7 Conclusion

We propose NETASSISTANT, a virtual assistant tool to answer network diagnosis questions and help both data center network users and network engineers. The tool is motivated by our measurement study about network user daily queries and the network monitoring primitives used by network engineers in our production network. NETASSISTANT abstracts the whole diagnosis process into the dialogue layer, workflow layer, and data layer. Accordingly, we design three functional modules to realize these layers and provide dialogue-based network diagnosis to our users. NETASSISTANT has been deployed and used in our company for over three years and the evaluation results show that this tool can significantly reduce the workload of both network users and network engineers and provide excellent user experience and performance.

Acknowledgments

We would like to thank many ByteDance colleagues who have contributed to this work. These include Jiale Feng, He Zhang, Anjian Chen, Dong Zhao, Chengcai Yao, Jing He, Chenxingyu Zhao, Shravan Jagadish Kumar, Chuanxiong Guo, and many others. We also thank our anonymous reviewers and shepherd Arpit Gupta for their valuable comments.

References

[1] Amazon alexa/echo. https://alexa.amazon.com/.

[2] Apache kafka. https://kafka.apache.org/.

[3] Apple siri. https://www.apple.com/siri/.

[4] Google Now. https://assistant.google.com/.

[5] Introducing chatgpt. https://openai.com/blog/chatgpt.

[6] Introducing llama: A foundational, 65-billion-parameter large language model. https://ai.meta.com/blog/large-language-model-llama-meta-ai/.

[7] Introducing palm 2. https://blog.google/technology/ai/google-palm-2-ai-large-language-model/.

[8] Introduction to syslog in cisco routers and switches. https://www.omnisecu.com/ccna-security/introduction-to-syslog-in-cisco-routers-and-switches.php.

[9] Logging - basic syslog and beyond. https://arista.my.site.com/AristaCommunity/s/article/logging-basic-syslog-and-beyond.

[10] Microsoft cortana. https://www.microsoft.com/en-us/cortana.

[11] Network switches, past, present – and an exciting future. https://gblogs.cisco.com/uki/network-switches-past-present-and-an-exciting-future/.

[12] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. Liu, J. Padhye, B. T. Loo, and G. Outhred. 007: Democratically finding the cause of packet drops. In *USENIX NSDI*, pages 419–435, 2018.

[13] R. Birkner, D. Drachlser-Cohen, L. Vanbever, and M. Vechev. Net2text: Query-guided summarization of network forwarding behaviors. In *USENIX NSDI*, pages 609–623, 2018.

[14] H. Chen, X. Liu, D. Yin, and J. Tang. A survey on dialogue systems: Recent advances and new frontiers. *ACM SIGKDD Explorations Newsletter*, 2017.

[15] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. In *ACM SIGCOMM*, pages 226–239, 2020.

[16] C. Fang, H. Liu, M. Miao, J. Ye, L. Wang, W. Zhang, D. Kang, B. Lyv, P. Cheng, and J. Chen. Vtrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network. In *ACM SIGCOMM*, pages 31–43, 2020.

[17] J. Gao, N. Yaseen, R. MacDavid, F. Vieira Frujeri, V. Liu, R. Bianchini, R. Aditya, X. Wang, H. Lee, D. Maltz, M. Yu, and B. Arzani. Scouts: Improving the diagnosis process through domain-customized incident routing. In *ACM SIGCOMM*, pages 253–269, 2020.

[18] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *SIGCOMM*, pages 139–152, 2015.

[19] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. Sonata: Query-driven streaming network telemetry. In *ACM SIGCOMM*, pages 357–371, 2018.

[20] C. Lou, P. Huang, and S. Smith. Understanding, detecting and localizing partial failures in large system software. In *USENIX NSDI*, pages 559–574, 2020.

[21] J. Ni, T. Young, V. Pandelea, F. Xue, and E. Cambria. Recent advances in deep learning based dialogue systems: a systematic survey. In *Artificial Intelligence Review*, pages 3055–3155, 2022.

[22] B. Qin, B. Hui, L. Wang, M. Yang, J. Li, B. Li, R. Geng, R. Cao, J. Sun, L. Si, F. Huang, and Y. Li. A survey on text-to-sql parsing: Concepts, methods, and future directions. In *arxiv.org/abs/2208.13629*, 2022.

[23] Y. Wu, A. Chen, and L. T. X. Phan. Zeno: Diagnosing performance problems with temporal provenance. In *USENIX NSDI*, pages 395–420, 2019.

[24] M. Yu. Network telemetry: Towards a top-down approach. *ACM SIGCOMM CCR*, pages 11–17, 2019.

[25] E. Zhai, A. Chen, R. Piskac, M. Balakrishnan, B. Tian, B. Song, and H. Zhang. Check before you change: Preventing correlated failures in service updates. In *USENIX NSDI*, pages 575–589, 2020.

[26] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, 2015.

[27] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng. Packet-level telemetry in large datacenter networks. In *SIGCOMM*, pages 479–491, 2015.

[28] X. Zuo, Q. Li, J. Xiao, D. Zhao, and J. Yong. Driftbottle: A lightweight and distributed approach to failure localization in general networks. In *CoNEXT*, 2022.