Establishing Trust for Using Natural Language for Intent-Based Networking

Arthur Selle Jacobs, Ricardo José Pfitscher, Rafael Hengen Ribeiro, Lisandro Zambenedetti Granville, *Senior Member, IEEE*, Ronaldo Alves Ferreira, Walter Willinger, *Fellow, IEEE*, and Sanjay G. Rao

***Abstract*—Today's enterprise networks wrestle with accommodating an ever-growing number of devices of different types, supporting increasingly demanding applications and ever more complex services, and protecting their users from sophisticated and disrupting cyber threats. In response, a proposed architectural approach for improving network management, referred to as Intent-Based Networking (IBN), has attracted significant attention. It is built on the premise that network operators specify network policies in natural language and the network correctly translates these spoken intents (*e.g.,* policies) into proper device-specific configurations that are then deployed across the network to reliably act on the operators' expressed intents. Unfortunately, IBN has not yet fully delivered on its promise of automated, fast, and reliable policy deployment, mainly due to the significant challenges that the reliance on methods from Natural Language Processing (NLP) or more recent techniques from Machine Learning (ML) and Artificial Intelligence (AI) poses for unambiguously and accurately translating the myriad of intents that operators can express in natural language into "trustworthy" device configurations. This paper uses LUMI, a recently designed end-to-end prototype of a system that allows operators "to manage their network by talking to the network," as an illustrative case study. In particular, we use it to elaborate on the different functionalities such systems should have to realize IBN's vision of automating the fast deployment of policies. At the same time, we leverage LUMI to highlight the extra efforts that are required to ensure that the deployed policies can be entrusted to accurately express and execute the operators' original intents.**

Received 7 September 2023; revised 10 March 2024 and 17 October 2024; accepted 1 May 2025. Date of publication 6 June 2025; date of current version 7 October 2025. The authors also acknowledge the financial support received from the Brazilian National Council for Scientific and Technological Development (CNPq) Procs. 420934/2023-5, 308101/2022-7, 465446/2014-0, 142089/2018-4; Brazilian Coordination for the Improvement of Higher Education (CAPES) Finance Code 001; Sao Paulo Research Foundation (FAPESP) Procs. 2020/05183-0, 2023/00811-0, and 2023/00812-7; and National Science Foundation (NSF) Award Number 2319425. The associate editor coordinating the review of this article and approving it for publication was F. Paganelli. *(Corresponding author: Ronaldo Alves Ferreira.)*

Arthur Selle Jacobs, Rafael Hengen Ribeiro, and Lisandro Zambenedetti Granville are with the Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre 90040-060, Brazil (e-mail: asjacobs@inf.ufrgs.br; rhribeiro@inf.ufrgs.br; granville@inf.ufrgs.br).

Ricardo José Pfitscher is with the Mobility Engineering Department, Federal University of Santa Catarina (Joinville), Joinville 89219-600, Brazil (e-mail: ricardo.pfitscher@ufsc.br).

Ronaldo Alves Ferreira is with the College of Computing, Federal University of Mato Grosso do Sul, Campo Grande 79070900, Brazil (e-mail: raf@facom.ufms.br).

Walter Willinger is with NIKSUN Inc., Princeton, NJ 08540 USA (e-mail: wwillinger@niksun.com).

Sanjay G. Rao is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: sanjay@purdue.edu).

Digital Object Identifier 10.1109/TNSM.2025.3574626

***Index Terms*—Intent-based networking, natural language processing, self-driving networks, machine learning.**

I. INTRODUCTION

AS MODERN networks grow in size and complexity, their operation becomes increasingly prone to human errors [1]. This trend has driven both industry and academia to try to automate network management and control tasks, avoiding having humans in the loop whenever and wherever possible [2], [3]. In recent years, researchers have envisioned managing and controlling networks that can automatically understand, refine, deploy, and execute human instructions for their own operation. This concept is commonly referred to as *Intent-based Networking (IBN)* and describes the ability of a network to interpret and implement high-level objectives or business goals (*e.g.,* specifying desired levels of quality of service, security, or performance for particular applications or services) without the operators having to worry about how the network will achieve them [4].

Since this vision of IBN promises to enable the automated and fast deployment of network intents that are expressed in natural language, it has attracted significant interest from the largest tech companies [3], [5] and service providers [6], [7]. Although they may differ in their motivation to pursue IBN, these companies view IBN as a means of reducing operational cost, especially at a time when they face a shortage of operators with the necessary domain expertise (*e.g.,* technical knowledge of network programming languages or vendor-specific Command-Line Interfaces (CLI)).

Given the growing interest from industry and the general appeal of IBN, a number of studies have attempted to leverage methods from NLP or the latest AI/ML-based techniques (*e.g.,* Large Language Models, or LLMs) to solve networking-related problems (*e.g.,* see [8], [9], [10], [11] and references therein). However, while IBN's general appeal is partly because it allows operators to express the same intent using different phrasings, this flexibility also creates enormous challenges for generating suitable configurations. In particular, to realize IBN in practice by deploying these generated configurations in real-world production networks, operators must be able to "trust" these generated configurations in the sense that the latter have to capture the operators' expressed intents unambiguously and accurately. Unfortunately, providing such guarantees that would make operators comfortable deploying the generated configurations in safety-critical settings and

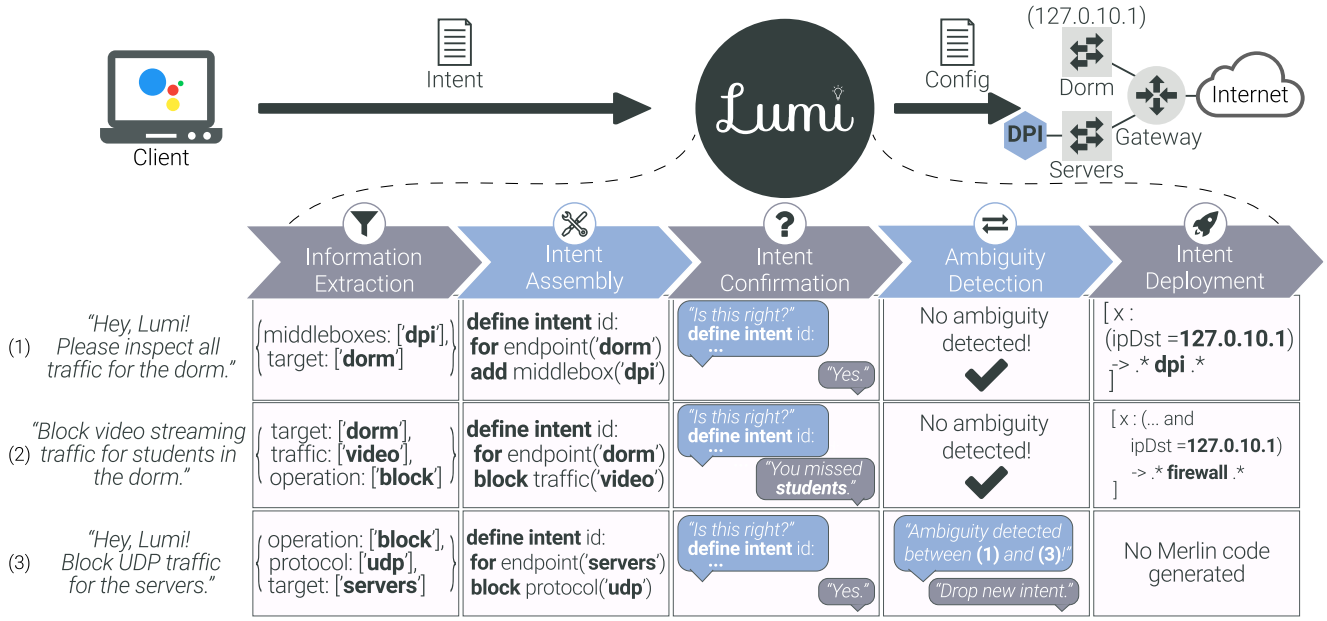


Fig. 1. The (extended) LUMI pipeline at a glance.

using them for high-stakes decision-making in their production network environments is currently not part of the state-of-the-art NLP or AI/ML methods used in this area [12].

These limitations notwithstanding, recent advances in AI/ML have made it possible to use NLP-based methods for trying to solve specific problems in the networking domain [8], [11], [13]. For example, by taking advantage of some of these advances, we designed and implemented LUMI [14], a prototype of a system with a conversational interface that enables an operator "to talk to the network." Given that LUMI is highly modular, with each of the modules enabling well-defined functionalities, having their own set of requirements and resulting in their own design choices, its design allows for easy plug-and-play, where existing modules can be replaced with alternative solutions, or new modules can be included. As a result, LUMI's architecture is extensible and evolvable and can easily accommodate further extensions or enhancements.

We use LUMI as a case study to present an in-depth account of how AI/ML methods, in conjunction with recent progress in other relevant scientific areas, can be leveraged to realize IBN. In the process, we elaborate on how LUMI can ensure that as the operators' original intents traverse the LUMI pipeline, they are correctly interpreted so that only "trusted" configurations are deployed. Here, the LUMI pipeline refers to the workflow that starts with receiving as input an operator's intent expressed in natural language, translating these natural language utterances into configuration commands, and finally deploying the generated configurations in the network to carry out the operator's intent.

In particular, in this paper, we describe two main contributions. First, we demonstrate the benefits of LUMI's modular design by presenting and evaluating an extension of LUMI's initial design for detecting ambiguities between newly considered intents and already deployed configurations (Section IV). Second, we show that our extended version of LUMI is one of the first examples of an AI/ML-based approach to IBN that provides the critical features for operators to trust the generated configurations (Section V).

Figure 1 shows the high-level design of the proposed extension of the original version of LUMI and uses the intent examples *"Hey, Lumi! Inspect traffic for the dorm."*, *"Block video streaming traffic for students in the dorm."*, and *"Hey Lumi! Block UDP traffic for the server."* to illustrate the breakdown of the workflow by which this extended version of LUMI accomplishes the stated objectives. The figure also serves as a roadmap for providing in Section III a brief summary of the original LUMI pipeline (*i.e.,* without the Ambiguity Detection module) and describing in Section IV the design and evaluation of the newly developed module for detecting ambiguous intents. These sections are preceded by a discussion of related work in Section II and are followed by Section V that gives a detailed account of how the extended version of LUMI engenders operators' trust. The paper concludes with a discussion of the limitations of our extended version of LUMI and a description of open challenges that constitute a rich agenda for future research in this area.

The extended LUMI pipeline described in this paper represents a promising step towards realizing the vision of IBN of achieving fast, automated, and *reliable* policy deployment. By letting operators express intents in natural language, LUMI not only simplifies the jobs of network operators (*i.e.,* deploying policies) and saves them time, but it may also help network providers reduce their operational costs. Importantly, LUMI's design highlights the extra efforts that are needed when relying on error-prone AI/ML-based decision-making to solve network management problems that tolerate no mistakes and require solutions that are unambiguous, accurate, and correct at all times. In this sense, LUMI also serves as a general guide for how to build systems that network operators can trust and use

"to talk to the network with confidence" (*i.e.,* with guarantees that intents are correctly interpreted and deployed).

II. Related Work

Given LUMI's primary goal of refining and deploying intents expressed in natural language, the five-stage processing pipeline shown in Figure 1 represents a logical "division of labor" and is an intuitive design choice for the system. However, designing each of the five modules comes with design choices that require careful consideration for the set of desired features and requirements we want each module to satisfy. Before discussing the various design choices we made for LUMI, we first review related existing work in this area.

A. Intent-Based Networking

Recent survey papers, such as [15], [16], summarize the latest efforts to realize the IBN vision for different networking domains, including enterprise and cellular networks. We briefly describe some of these prior efforts, but note that they all differ from LUMI in the sense that they are not concerned with using natural language for engaging network operators in a purposeful dialogue.

One category of previous efforts deals with the problem of *middlebox chaining*. Works such as [17], [18] focus on determining which virtual network functions should be chained to fulfill network and security intents from network operators. Another category is concerned with *policies related to access control lists (ACL) and privacy*. Several works [19], [20], [21], [22] propose automatic processes for refining, deploying, and enforcing ACL policies, leveraging distinct abstractions for describing intents, such as graphs and special-purpose intent representation language. A third category consists of prior, more architectural IBN-related efforts. One example is [23] that describes an extension of the *Nile* language and uses it to declare intents that express how to program P4 switches. Another example is [24], [25], where the authors focus on designing and modeling end-to-end IBN-based network architectures but are not concerned with practical aspects like implementing the proposed system.

B. Dealing With Conflicts and Ambiguities

Recent interest and proposals in advancing the use of IBN have spurred efforts to address conflict detection and automated resolution. Most of the recent work in this area focuses on strict algorithmic solutions for analyzing the static properties of network intents and identifying conflicting behaviors. However, these solutions often fall short of detecting dynamic interactions between intents once deployed on the network substrate.

In [26], [27], the authors propose a Prolog-based declarative model to describe intents for chaining Virtual Network Functions (VNFs), with the capability of automatically identifying and resolving certain types of conflicts between two intents. A different approach is considered in [28] where the authors describe a GUI-based system for users to express their intents that is also capable of detecting conflicts and makes up to ten attempts to automatically deal with detected conflicts while making the least amount of changes to previously deployed intents. Their algorithmic approach relies on pairwise comparison of incoming intents with previously deployed intent, relying mostly on analyzing intents types, affected endpoints and chosen parameters. To contrast, the authors of [29] rely on a method that is based on a Satisfiability Modulo Theory (SMT) solver to detecting conflicts between networks intents that compose a network policy set. This method leverages the commonly-used five-tuple definition to uniquely identify incoming packets and determine if they match more than one expressed intents.

Despite providing interesting stepping stones for dealing with conflicts, these works still do not cover the extent of LUMI-supported operations and the ambiguities that may arise from combining them. In fact, as highlighted in [27], very few works have even attempted to classify the types of conflicts that may occur between network intents, an effort that further depends on the representation used to express those intents. Given LUMI's support of natural language intents, scrutinizing the possible interactions between intents for conflicts and ambiguities remains an open problem.

C. Intent-Based Networking and AI/ML

From its conception, the proposal of IBN has been intrinsically tied to advances in AI/ML to fulfill it. Earlier works often relied on recurrent neural network models with an established track record in NLP techniques, such as LSTMs, for intent refinement [30]. With the recent advances in transformer-based Large Language Models (LLMs) such as OpenAI's ChatGPT, Google's Gemini and Meta's LLaMA, a number of research efforts have attempted to reap the benefits of those models and apply them to IBN [31].

In [32], the authors propose AppleSeed, an intent-based end-to-end system that leverages ChatGPT to translate natural language intents directly into Python scripts to manage networking infrastructure. Similarly, the approach described in [10] relies on LLMs to iteratively generate network policies and execute them through a series of API calls based on which action was identified by the LLM (*e.g.,* create, delete, validate). In [33], the authors extend their proposal to use another LLM to analyze collected healthcheck metrics from deployed intents to identify "intent drift" (*i.e.,* checking if the original intent is still true) and suggest corrective actions to fulfill the original intent again. In [34], the authors also leverage LLMs to propose an end-to-end IBN architecture, translating natural language intents into a standardized intent notation, such as ETSI's Network Service Descriptor.

Of particular interest and perhaps closest to our work is the recent paper [35] that presents NAIL, a network management architecture that relies on NLP and AI/ML techniques to translate network intents expressed in natural language into data-plane programming code, such as P4 or DPDK, that satisfies the described intent. However, it is noteworthy that the authors of NAIL forgo the use of modern LLMs because of the unreliability of the code generated by such models. Instead, the authors rely on more traditional and well-established NLP

techniques to identify key objectives and network components described in the network intents.

While not strictly related to IBN, other works enable operators to use natural language for some network configuration tasks by relying on LLMs as well. In [11], the authors use LLMs to generate task-specific code to analyze and manipulate a graph structure that models the underlying network infrastructure. In [36], the authors propose NETBUDDY, an LLM-powered system that uses a multi-stage refinement process to translate natural language policies and requirements into low-level network configurations in the form of P4 programs. Some of these works acknowledge the mistakes from LLMs and provide basic safeguards to mitigate them, such as leveraging sandbox environments for deployment [10], [11], [33] or relying on intermediate representations instead of directly generating configuration code [36]. However, aside from [34], these studies lack the means to prevent LLMs from making mistakes and producing erroneous network configuration, and most of them are more concerned with ensuring that LLMs achieve a correct syntax than warranting that the underlying semantic meaning of the original intent is fulfilled. In fact, none of these works support LUMI-like capabilities such as verifying and validating generated configurations through direct interactions with and feedback from the operator.

III. THE ORIGINALLUMI PIPELINE: A RECAP

The original design and implementation of LUMI as described in [14] resulted in a prototype of a system with a conversational interface that enables operators "to talk to the network." The modular nature of this design defines a workflow (*i.e.,* the LUMI pipeline) that starts with receiving as input an operator's intent expressed in natural language, translating these natural language utterances into configuration commands, and finally deploying the generated configurations in the network to carry out the operator's intent. Formally, the original version of this pipeline is comprised of the Information Extraction, Intent Assembly, Intent Confirmation (and Feedback), and Intent Deployment modules. In this section, we briefly summarize the purpose and functionality of each of these four different modules. For a more detailed account of the original LUMI pipeline and its design, we refer to [14].

A. Information Extraction

This module is responsible for considering intents such as those shown on the left-hand side of Figure 1, correctly extracting named entities, and correctly labeling them by their type. In the case of the example intents, the extracted named entities and their types are given in the column "Information Extraction" in Figure 1.

The main building blocks for LUMI's Information Extraction module are a chatbot interface as its entry point and the use of *Named Entity Recognition (NER)* [37] to extract and label entities from the operators' natural language intents. Given the popularity of personal assistants, such as Google Assistant, Amazon Alexa, or Apple Siri, LUMI's goal in providing a natural language interface goes beyond facilitating the lives of traditional network operators to possibly include technologically less savvy users such as homeowners who may want to manage their home networks.

Solving the NER problem involves applying machine learning (for extracting named entities in unstructured text) and using a probabilistic graphical model (for labeling the identified entities with their types). Although, in theory, NER is largely believed to be a solved problem [38], in practice, to ensure that NER achieves its purpose with acceptable accuracy, some challenges remain, including careful "entity engineering" (*i.e.,* selecting entities appropriate for the problem at hand), a general paucity of tagged or labeled training data, and a lack of trust in the correctness of the extracted and labeled entities.

LUMI addresses these challenges by defining the set of entities hierarchically and organizing them into three different categories: *common*, *composite*, and *immutable* entities. Here, common entities form the bottom of the hierarchy, comprise raw textual values, and largely determine what LUMI can understand. For example, the textual values in the common entity class *@middlebox* are network functions such as firewalls, packet inspection, and traffic shaping. The hierarchy's intermediate level consists of composite entities. The entities in this class do not have any inherent nouns, verbs, or even synonyms associated with them; they only establish a relationship between common entities through the aggregation of prepositions and help with avoiding ambiguities (see Section IV). For example, the composite entity class *@origin* consists of composite values such as "from *@location*" and "from *@service*". Finally, immutable entities make up the top of the hierarchy and form the core of LUMI. In particular, while the entity class *@operation* expresses the operations that *Nile* supports (*e.g.,* "allow", "block", "set"), the entity class *@entity* consists of a list of LUMI-supported common entities (*e.g.,* "group", "location", "service").

To extract and label *entities* (*i.e.,* actions and targets) from the intent the operator expresses in natural language, LUMI leverages the machine learning-based NER approach. LUMI'S NER architecture is described in detail in [14] and uses an enhanced version of the Long Short-Term Memory (LSTM) model (*i.e.,* the so-called Bi-LSTM model [39]). Then, LUMI employs a statistical modeling method known as Conditional Random Field (CRF) for entity labeling [40] using Inside-Outside-Beginning (IOB) tagging for identifying entity values with multiple words [37]. LUMI's NER algorithm is trained with domain-specific entities to recognize network intents concerned with common features such as middleboxes, service-level agreement requirements, temporal restrictions, and network endpoints.

Since the initial design of LUMI, new advances with Transformer recurrent neural networks [41] have led to improvements for solving the NER task [42] with models such as BERT [43], DIET [44] and GPT-4 [45]. Although Transformer-based classifiers achieve better performance for general-purpose natural language processing, we found the simpler LSTM architecture to be competitive for extracting networking-related information from natural language. At the same time, because of LUMI's fully modular

TABLE I
OVERVIEW OF *Nile*-SUPPORTED OPERATIONS

Operation	Function	Required	Policy Type
from/to	endpoint	Yes	All
for	group/endpoint/service/traffic	Yes	All
allow/block	traffic/service/protocol	No	ACL
set/unset	quota/bandwidth	No	QoS
add/remove	middlebox	No	Service Chaining
start/end	hour/date/datetime	No	Temporal

design, nothing prevents a researcher from replacing LUMI's existing information extraction module with a Transformer-based module without affecting the system's end-to-end functionality.

B. Intent Assembly

Using a chatbot interface with NER capabilities as the front end of LUMI solves only part of the problem of defining and deploying network intents. For example, if a network operator asks a chatbot *"Please add a firewall for the dorm."*, the output of the extraction module could be the following entities: {*middleboxes: 'firewall'*}, {*target: 'dorm'*}. These two pairs of key-values do not translate immediately into meaningful network configuration commands. Assembling the extracted entities into a structured and well-defined intent such that it can be interpreted and checked for correctness by the operator before being deployed in the network calls for an abstraction layer between natural language intents and network configuration demands.

LUMI relies on an extended version of the Network Intent Language (*Nile*) to serve as abstraction layer language. In previous work [46], we proposed an initial version of *Nile*, which provided minimal support for intent definition and focused primarily on service-chaining capabilities. However, LUMI uses an extended version of *Nile* which provides support for crucial features for network management in real-world environments (*e.g.,* usage quotas and rate-limiting). Closely resembling natural language, this extended version of *Nile* has a high level of legibility, reduces the need for operators to learn new policy languages for different types of networks, and supports different configuration commands in heterogeneous network environments. Using this extended version of *Nile* as abstraction layer language, the purpose of the Intent Assembly module is to ingest as input the output of the Information Extraction module (*i.e.,* entities extracted from the operator's utterances), assemble this unstructured extracted information into syntactically correct *Nile* intents, and then output them.

Table I shows the main operations supported by the extended version of *Nile*. The full grammar of *Nile*, in EBNF notation, is available on LUMI's website [47]. Some operations have opposites (*e.g.,* allow/block) to undo previously deployed intents and enable capturing incremental behaviors stated by the operators. Other operations in a *Nile* intent are mandatory, such as **from/to** or **for**. More specifically, an operator cannot write an intent in *Nile* without stating a clear target (using **for**) or an origin and a destination (using **from/to**) when the direction of the specified traffic flow matters.

C. Intent Confirmation and Feedback

Using natural language to operate networks (*e.g.,* as part of IBN) is subject to inherent vagueness that arises from the different ways in which operators can express the same intent in different network environments in natural language. To address this problem, LUMI is designed with a learning-based solution and leverages the chatbot interface implemented as part of its first module.

In particular, the purpose of LUMI's Intent Confirmation module is to make sure that the output of the Intent Assembly module (*i.e.,* syntactically correct *Nile* intents) is confirmed by the operator before being deployed. Importantly, when presented with assembled intents that result in false positives or false negatives, the operator is asked to provide feedback that LUMI subsequently uses to augment the original training dataset with new labeled data; that is, LUMI learns new constructs over time, gradually reducing the chances of making mistakes. Although this interaction may slow down initial intent deployments until LUMI adapts to the operator's usage domain, it is essential for guaranteeing reliable intent refinement and deployment.

Extracting pertinent information from user feedback also requires identifying specific entities in the feedback text, similar to extracting entities from an input network intent. To this end, LUMI uses the same NER model for both tasks, relying primarily on the immutable *@entity* to extract which entity class is being augmented and what value is being added. Relying on the same NER model also requires retraining of the model to identify and extract entities in the received user feedback. However, since LUMI's interactions with the user are limited to answering simple questions, processing user feedback does not require a large set of training samples.

D. Intent Deployment

The fourth and last module of the original LUMI pipeline compiles the operator-confirmed *Nile* intents into code that can be deployed on the appropriate network devices and then executes the original network intent expressed by the operator in natural language.

To this end, the abstraction layer provided by *Nile* enables compilations to a number of different existing network configurations, including other policy abstraction languages such as *Merlin* [48], *Janus* [21], *PGA* [20], and *Kinetic* [49]. In view of the set of supported features and code availability, we chose to compile structured *Nile* intents into *Merlin* programs for the LUMI's Intent Deployment module. The *Merlin* language was chosen over these alternative frameworks because of its good fit with *Nile*, the network features it supports, its performance, and the availability of source code. The following examples illustrate a number of practical aspects of mapping *Merlin* features to the set of operations supported by *Nile* intents.

Resolving logical handles. Logical handles in *Nile* intents are decoupled from low-level IP addresses, VLAN IDs and IP prefixes, which LUMI resolves (*e.g.,* `dorm` $\rightarrow$ `10.1.2.0/24`) using information provided during the bootstrap process. ACLs rules are resolved similarly. Once LUMI produces *Merlin* programs with resolved identifiers (*i.e.,*

VLAN IDs, IPs, and prefixes), the compilation to the corresponding OpenFlow rules is handled by *Merlin*.

Temporal constraints and QoS. As *Merlin* does not support temporal policies, LUMI stores every confirmed *Nile* intent so that it can install or remove device configurations according to times and dates defined by the operator. Another feature defined in *Nile* that is not natively supported by *Merlin* are quota restrictions. Quota restrictions are achieved by relaying all traffic marked with a quota requirement to a traffic-shaping middlebox, taking advantage of *Merlin*'s support for middlebox chaining. Other QoS policies, such as rate limiting, are already supported in *Merlin*.

Middlebox chaining. LUMI focuses on middlebox chaining, *i.e.,* correctly relaying traffic specified in the intents through a sequence of middleboxes. Since the actual configuration of each middlebox is done outside of LUMI, LUMI can handle chaining policies associated with *any* middlebox type, virtual or physical, compilation for which is straightforward since *Merlin* natively supports middlebox chaining.

IV. THE EXTENDEDLUMI PIPELINE: AMBIGUITY DETECTION MODULE

By enabling network operators to express network intents in natural language and then translates these natural language intents into device configurations, LUMI's original design lacks the means for detecting, mitigating, or ruling out the creation of ambiguous forwarding rules. In this section, we define what constitutes an ambiguity between two network intents and demonstrate the benefits of the modular design of the original LUMI pipeline by presenting the design, implementation, and evaluation of a new Ambiguity Detection module. This new module enriches the original LUMI workflow and can be easily inserted between the existing Intent Confirmation and Intent Deployment modules to form an *extended* LUMI pipeline capable of detecting ambiguities between newly considered network intents and already deployed configurations.

A. Ambiguous Network Intents: Definition

By allowing operators to express network intents with temporal restrictions, quotas, QoS parameters, or high-level ACL rules, LUMI is prone to result in potentially conflicting intents that are outside the purview of commonly considered conflicts in the data plane [50] or control plane [1], [51] (*e.g.,* opposing forwarding behavior in distinct switches and routers). To distinguish conflicts that arise due to LUMI's use of higher levels of abstractions (*i.e.,* natural language) from these more traditional configuration conflicts, we refer to the former as *ambiguities* and use the following formal definition of what constitutes an ambiguity between two network intents.

Definition 1: We define intent ambiguity as a directional relationship between pairs of network intents, denoted by "Text" (T) and "Hypothesis" (H). Here, T refers to the ground truth intent which is already deployed and H is the new intent to be introduced. We say that H and T are ambiguous if there is no possible network state in which both intents can be asserted. Note that in the textual entailment framework [52], Text and Hypothesis are termed the entailing and entailed text fragments, respectively.

TABLE II
TYPES AND EXAMPLES OF AMBIGUOUS INTENTS

Type	Ambiguity
Path overlap (Fig. 2)	T: Inspect traffic for the dorm. H: Block udp traffic in the servers.
Temporal overlap	T: Prioritize Netflix from 12 PM to 1 PM. H: Block Netflix from 8 AM to 8 PM.
Negation	T: Allow HTTPS traffic from backend to database. H: Block all traffic to database servers.
QoS	T: Set maximum 10 Mbps bandwidth for campus dorms. H: Set minimum 30 Mbps bandwidth for students.
Synonym	T: Block all streaming traffic for students dormitories. H: Dorm must be able to access YouTube.
Hierarchy	T: Set 10 Gb per week usage quota for Prof. John Doe. H: Set 1 Gb per week usage quota for professors.

In effect, we require that before deploying any device configurations that LUMI created in response to an operator's expressed network intent, the system checks for any possible intent ambiguities, identifies them and informs the network operator.

B. Detecting Intent Ambiguities

Given that the stated goal of LUMI's extended pipeline is to prevent inconsistent device configurations resulting from LUMI's use of natural language and higher-level abstractions, designing an new Ambiguity Detection Module for LUMI boils down to implementing a practical approach to identifying the type of intent ambiguities that can arise from the operations *Nile* supports.

To this end, we consider the operations supported by the *Nile* language (see Table I) and map the ambiguities that may occur between intents to ramifications of those ambiguities in case the intents were deployed in the network. In view of the limitations of simpler approaches to detecting ambiguities discussed earlier (see Section II), LUMI's Ambiguity Detection module uses supervised learning to label pairs of intents as ambiguities or as ambiguity-free. In the following, we first discuss the key task of feature engineering (*i.e.,* extracting relevant features from each pair of intents) that is informed by the effort of mapping different types of ambiguities and then describe LUMI's choice of learning model for detecting intent ambiguities. Note, however, that intent ambiguities are in general a mixture of different types, and accurately identifying them requires considering their entire feature set and not their individual features in isolation.

1) Feature Engineering: When considering the operations supported by *Nile*, Table II lists the six main types of ambiguities that can occur in a pair of *Nile* intents. Other types of ambiguities may arise with the addition of new operations and should be addressed accordingly. In the following, we discuss some of these ambiguity types in more detail and list the features that we selected as relevant for each type.

Path overlap. Figure 2(d) shows the two seemingly unrelated intents defined in the first example in Table II. However, trying to deploy the new intent (left) with another intent

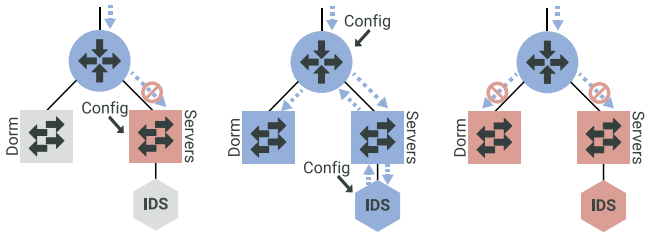


Fig. 2. Path overlap ambiguity example: "Block UDP traffic in the servers." (intent 1 on left); "Inspect traffic for the dorm." (intent 2 in middle); trying to deploy new intent 1 with intent 2 already deployed will result in an ambiguity (right).

(middle) already in place creates an ambiguity, as UDP traffic for the dorm will be inadvertently dropped. To address this type of ambiguity, we note that the paths on which the example intents act overlap in the underlying network (*i.e.,* links between Gateway and IDS nodes). This observation motivates computing the network path in which a new intent will be deployed, comparing it with the path of the already deployed intent, counting the number of switches and routers the two paths have in common, and using the resulting count as the feature. The longer the path the two intents share, the higher the chances that the two intents contradict each other. In turn, regardless of the configurations expressed in the two intents, if the two paths do not overlap, the two network intents are likely ambiguity-free.

Temporal overlap. Since *Nile* allows operators to specify intents that take place at specific times during the day, it is possible that two intents do not define an ambiguity, even if they dictate opposite behavior for the same targets (*e.g.,* see the second example in Table II). If not for the temporal constraints, the two intents would contradict each other. To detect this type of ambiguity, we can use as a feature the temporal overlap that the two intents have (expressed in minutes), where intents without any temporal settings are considered active at all times.

QoS. QoS constraints are familiar sources of ambiguities, and detecting them is a well-known hard problem (see, for example, [21], [48]). We can select as the QoS feature a simple binary marker of QoS feasibility that returns one if the QoS metrics, constraints, and values in both intents do not oppose each other, and zero if both intents are obviously surpassing the capacity of their computed paths.

Negation. Since each *Nile* operation has a direct opposite that negates it (*e.g.,* *add/remove*, *allow/block* and *set/unset*), the presence of one or more negations is one of the best indicators of contradictory behavior. The feature we can use to detect this type of ambiguity is computed as the number of occurrences of negations in both intents.

Synonyms. Since operators are free to refer to operation targets (*e.g.,* service, location, or type of traffic) using different words, LUMI relies on WordNet [53] to obtain synonyms of operation targets in both intents and also uses a dictionary of known "handles" for network devices and groups (*i.e.,* different ways of addressing the same network elements). The feature LUMI uses to deal with this type of ambiguity is the number of matches it finds as it considers each synonym of an operation target in the new intent and searches the operation targets of the previous intent.

Hierarchy. As shown in the last example in Table II, ambiguities can sometimes occur due to a hierarchical relationship between intent targets, services, or traffic, respectively (*e.g.,* Prof. Doe vs. professors, BitTorrent vs. peer-to-peer traffic). LUMI uses two features for this type of ambiguity; one for intent endpoints and groups, and another for types of traffic, services, and protocols. Both are binary indicators of ancestry that encode whether any operation target of the new intent is a descendant or ancestor of any operation target of the previously deployed intent.

In addition to the features directly related to each of the ambiguity types listed in Table II, LUMI also extracts several features related to the **similarity** of both intents. It measures the similarity of all *Nile* operation targets (*i.e.,* endpoints, services, traffic, etc.), generates a similarity score per target, and also counts the number of occurrences of each operation target. This process results in twelve features, representing the lexical similarity scores for each of the three *Nile* operation targets in each of the two intents. The similarity score is computed using the SpaCy [54] built-in function of the Python NLP library, which converts words into vector representations and computes cosine similarity based on generated vectors.

2) Learning Model: Our implementation of LUMI's Ambiguity Detection module uses a pre-trained Random Forest Classifier. We arrive at that choice by standardizing the input feature vectors (*i.e.,* subtracting their mean and dividing by their standard deviation to ensure all features are within a common range) and using the transformed feature vectors to train different learning models that also perform feature selection (*i.e.,* use PCA-like methods to analyze the importance of features and consider only the most important feature for the subsequent classification task). We then selected the best-performing classification model out of three popular candidates, *i.e.,* Logistic Regression, Support Vector Machines (SVM), and Random Forests (see below for more details).

C. Evaluation of the New Ambiguity Detection Module

In view of the detailed evaluation of the original LUMI workflow and its four modules provided in [14], we focus in the following on evaluating the newly-designed Ambiguity Detection module of the extended LUMI pipeline. Also note that the full implementation of the extended LUMI pipeline, a working demo, and all datasets used in the course of developing, testing, and deploying LUMI's different modules are publicly available at [47].

To train and validate our Ambiguity Detection module, we need an ambiguity corpus, *i.e.,* a dataset of natural language sentences annotated as ambiguous or non-ambiguous. However, to the best of our knowledge, no such publicly available corpus exists. To circumvent this problem, we generated three datasets of different sizes (with 100, 1,000 and 5,000 entries) with pairs of *Nile* intents that we annotated "by hand" as being ambiguous or not. We use a standard campus network topology [55] and known services, groups, and traffic to create random pairs of *Nile* intents with and without ambiguities.

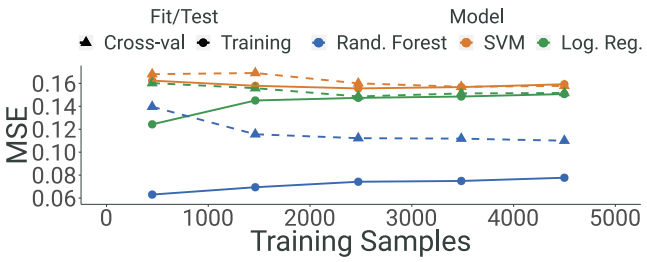


Fig. 3. Learning curves for each model, trained with a 5,000 entries dataset.

TABLE III
RESULTS OF RANDOM FOREST CLASSIFIER IN *campi* DATASET, BY AMBIGUITY TYPE, RESTRICTED BY UNIVERSITY

Type	TP	TN	FP	FN	Precision	Recall	F1
Path overlap	0	43	0	0	0	0	0
Temporal overlap	0	0	0	0	0	0	0
Negation	0	154	0	0	0	0	0
QoS	8	3	0	1	1	0.888	0.941
Synonym	0	0	0	0	0	0	0
Hierarchy	1	3	0	0	1	1	1
Total	9	203	0	1	1	0.9	0.947

We rely mostly on the principle of path overlap to generate targets for the intents that do not share any network element, or that do share network elements and have obvious opposite operations.

1) Model Selection: For each dataset, we used a 75%-25% train-test split, 10-fold cross validation, to evaluate the three classification models: Logistic Regression, Support Vector Machine (SVM) and Random Forest Classifier. Among the three evaluated models, the Random Forest Classifier achieved the highest Precision=**0.819**, Recall=**0.819** and F1-score=**0.819**, with the highest scores resulting from using the 5,000 entries dataset. Note that Precision and Recall have similar values. This happens because the datasets are balanced, with a similar number of ambiguous and non-ambiguous cases. We also analyzed the learning curves for each model, shown in Figure 3, where we plot the Mean Square Error (MSE) obtained during training and during cross-validation with the 5,000 entries dataset.

The learning curves show that the MSEs for Logistic Regression and SVM models are significantly higher than for the Random Forest model. We also observe that for the former, the MSEs for cross-validation converge to the training MSEs, indicating a very low variance and a high bias. This means that increasing the number of training cases will most likely not increase precision and that we have reached the limit of generalization of these models. In comparison, we note that there is a large gap between cross-validation and training MSEs in the case of the Random Forest model, indicating higher variance and lower bias. Random Forest models tend to work better with a large number of features, as is the case for our problem. Therefore, if we increase the number of training examples, we might be able to achieve higher precision. With that in mind, we generated a new dataset with 10,000 entries to train the Random Forest model to see if we could achieve higher Precision and Recall rates. As expected, using the 10,000 entries dataset, we achieved our highest results for Precision=**0.893**, Recall=**0.889** and F1-score=**0.888**. We rely on this Random Forest Classifier, trained with 10,000 samples, as our final model for LUMI's Ambiguity Detection module and use it for the remainder of this evaluation section.

2) Model Evaluation: To evaluate the accuracy of the pre-trained Random Forest Classifier in our ambiguities detection module, we used it to analyze the campus network intents from the *campi dataset* (expressed in *Nile*), trying to identify ambiguities in the policies published by the different universities. Universities with only one intent were discarded because we require at least two intents to check for ambiguities, and we ended up with 213 unique pairs of intents when only pairing intents from the same university. Using the same generic campus network we relied on for generating our synthetic datasets, we manually labeled each pair of intents as having an ambiguity or being ambiguity-free, identified its ambiguity type, and compared our results against this ground truth.

Given that these published policies can be assumed to be currently deployed in their respective universities, we did not expect to encounter any ambiguities in this dataset. To our surprise, we found nine of the 213 pairs of intents to be ambiguous. The ambiguous intents were from four universities and resulted from contradicting ACL and QoS policies that were published on the universities' websites. However, since these intents were extracted from different published documents on the universities' websites, these ambiguities could result from inconsistencies across the university's different websites and might not exist in the universities' actual networks.

We used our labeled dataset of ambiguous intents as input to our pre-trained Random Forest Classifier to classify each example. The results in Table III show that despite being trained with synthetic data only, our model is able to achieve very good Precision and Recall in evaluating real-world intents (*i.e.,* F1 score of 0.947).

Noting the relatively small number of network intents that we extracted from campus policies, we repeated the evaluation of our ambiguity detection module with a larger dataset that we generated by applying the same intent pairing method as before (*i.e.,* using the campi dataset), but without separating intents by university. It allows us to examine the different ways that universities handle network policies. The resulting dataset consists of 1,221 unique intent pairs (including the 213 intent pairs from the smaller dataset used before) that we also manually labeled as having an ambiguity (71) or being ambiguity-free (1114) and categorized by type.

We again used our pre-trained Random Forest model to classify each pair in this dataset and the results are shown in Table VI. We observe that while our model remains moderately successful in detecting most of the ambiguous intents (*i.e.,* reaching an F1 score of almost 0.8), it has difficulties with some types (*e.g.,* QoS intents and intents with negating operations where it results in 18 False Positives and 10 False Negatives). At the same time, in light of the fact that our model was trained with entirely synthetic data,

TABLE IV
RESULTS OF RANDOM FOREST CLASSIFIER IN *campi* DATASET, BY AMBIGUITY TYPE, REGARDLESS OF UNIVERSITY

Type	TP	TN	FP	FN	Precision	Recall	F1
Path overlap	1	634	3	0	0.25	1	0.4
Temporal overlap	2	0	0	0	1	1	1
Negation	14	285	2	10	0.875	0.583	0.7
QoS	53	74	18	2	0.746	0.963	0.841
Synonym	0	118	0	0	0	0	0
Hierarchy	1	3	0	1	1	0.5	0.666
Total	71	1114	23	13	0.755	0.845	0.797

we are encouraged by these results. Even though the intents were originally deployed in different networks and represent therefore no actual ambiguities, our evaluation sheds light on the potential of the classification model for detecting nuances in intents expressed in natural language and accurately classifying pairs of intents from real university policies. While our approach helps mitigate the lack of annotated corpora of intent ambiguities, it argues for special future efforts towards the collection of datasets of real-world network intents in support of learning-based approaches to detecting ambiguous network intents.

3) Resource Usage: To assess the feasibility of LUMI's Ambiguity Detection module to be deployed in practice, we also analyzed the resource consumption of our chosen ML model, as well as training and prediction times when deploying a new intent. For this evaluation, we used a 16-inch MacBook Pro, running MacOS Ventura 13.5 (22G74), with an M1 Pro Processor, 32GB of RAM and 500GB of storage. First, we measured training times for our Random Forest Classifier model using the 10,000 entries dataset, repeating the process 30 times, which resulted in an average of 0.19 seconds ($\pm$ 0.002 s) to train the model. Being an ensemble model, Random Forest Classifiers are extremely efficient when training with large amounts of data, given it can be parallelized with multiple jobs. This explains the fast training times, even on a standard laptop. As a result, the consumed CPU and RAM during this training times were negligible (around 1% on average).

Second, we analyzed the time it takes to predict if a new intent is ambiguous before deploying it. We used the Random Forest model to predict if each of the 50 intents in the *campi* dataset were ambiguous with the previously deployed intents (pair-wise), as they were deployed into the network. We repeated this experiment 30 times and noticed that, as expected, the time our Random Forest Classifier takes to process a new intent increases linearly with each new intent deployed in the network. While the first intents had prediction times close to 0, as there were few intents to analyze, the 50th intent took roughly 0.7 seconds, on average, for the model to process it (*i.e.,* to compare the new intent with 49 previously deployed intents). However, this naive pair-wise process could be greatly optimized, for instance, by parallelizing the pair-wise predictions, or by constraining the number of intents to include only the ones that affect the same subnets of the network. In the process we measured negligible CPU and RAM consumption throughout this experiment and open-source all results on LUMI's GitHub repository.

V. ENGENDERING TRUST IN LUMI

For IBN-generated configurations to be deployable in real-world production networks, operators require that they can "trust" them; that is, capture the operators' expressed intents unambiguously, accurately, and reliably. In this sense, IBN defines an application domain of AI/ML for high-stakes decision making that does not tolerate any mistakes, be they false-positives or false-negatives. However, since state-of-the-art AI/ML methods used in the networking area are in general incapable of providing the type of guarantees that would make operators comfortable with relying on IBN-generated configurations to operate their production networks [12], LUMI has been designed with special features that provide the means for operators to gain confidence in the derived intents and trust the generated configurations.

In this context, it is important to note that the considered version of the LUMI pipeline errs on the side of caution and asks for operator feedback and confirmation repeatedly and at different stages within the pipeline. While this constant interaction with the operator may appear to contradict the concept of IBN that strives for reducing the need of humans in the loop, with increased familiarity with LUMI, operators are free to minimize the interactions to only those they deem critical. To this end, it is easy to fine-tune the existing LUMI pipeline to any desired level of operator-system interactions.

In the following, we discuss and explain the various module-specific features that LUMI supports to help operators engender trust in LUMI-generated configurations.

A. On the Use of NER for Information Extraction

The NER method described in Section III is an instance of a supervised learning algorithm. The primary training step consists of both adapting the weights of the Bi-LSTM model and re-calculating the conditional distribution of the CRF model to infer the correct NER tags, and this step may have to be repeated until convergence.

Note that retraining can be done each time the existing corpus of training data is augmented with new key-value pairs or with existing entities used in a novel context (*i.e.,* requiring a new tag). A basic mechanism for obtaining such new key-value pairs or for benefiting from the use of existing entities in a novel context is to engage users of LUMI and entice their feedback in real-time, especially if this feedback points to missing or incorrect pieces of information in the existing training data. By enticing and incorporating such user-provided feedback as part of the Intent Confirmation stage (see Section III for more details), LUMI's design leverages readily available user expertise as a critical resource for constantly augmenting and updating its existing training dataset with new and correctly-labeled training examples that are otherwise difficult to generate or obtain. After each new set of key-value pairs is obtained through user feedback, LUMI augments the training corpus and retrains the NER model from scratch.

Thus, LUMI's approach improves upon commonly-used NER implementations by incorporating user feedback that enables LUMI to both learn over time (*i.e.,* thus reducing the problem caused by a paucity of labeled training data) and gain the operators' trust (*i.e.,* by providing a means for

operators to correct mistakes that LUMI may make as part of the information extraction stage).

B. On the Use of Nile for Intent Assembly

To enforce the correct syntax of the assembled intent, LUMI leverages the *Nile* grammar to guarantee that the intent contains the required information for correct deployment. If the grammar enforcement fails due to the lack of information, the system prompts the operator via the chatbot interface to provide the missing information. Assume, for example, that the operator's original intent stated *"Please add a firewall."*, without providing the target of the intent. Since specifying a target is required according to the *Nile* grammar, the module will not attempt to construct a Nile program but will instead interact with the operator to obtain the missing information.

C. On the Use of Feedback for Intent Confirmation

LUMI's Intent Confirmation module requires the output of the Intent Assembly module (*i.e.,* syntactically-correct *Nile* intents) to be confirmed by the operator. If the assembled intents result in false positives or false negatives, the operator is asked to provide feedback that LUMI subsequently uses to augment the original training dataset with new labeled data. To this end, LUMI is designed to be able to learn new constructs over time, gradually reducing the chances of making mistakes.

To reduce an operator's need for technical knowledge during this intent confirmation stage, LUMI supports feedback interactions that induce the operator to provide the information that LUMI needs. This is achieved by verifying if the required entities for valid feedback have been filled, and asking complementary questions in case they have not. A suggestion capability was also implemented to enable operators to select a button with the entities that LUMI did not extract.

To provide operators with more flexibility and not insist that they have to use specific keywords, this module compares the entities and values provided by the operators with synonyms already in the training dataset. Figure 4 illustrates a case where, due to a lack of training, LUMI misses an entity in the input text, and the confirmation mechanism lets operators easily catch this mistake and provide the necessary feedback for LUMI to learn. Also, note that this module's design is conservative in that operator feedback is requested for each assembled intent, irrespective of LUMI's confidence level concerning its accuracy.

D. On using Learning for Ambiguity Detection

Operationally, the Ambiguity Detection module in LUMI pairs a new *Nile* intent with each of the previously deployed intents to extract a set of features and uses a pre-trained Random Forest Classifier to produce a numeric result indicating if the new intent contradicts any of the intents deployed in the past. If the predicted outcome classifies an intent as ambiguous, LUMI uses the chatbot interface to ask for operator feedback to decide whether it should keep the new intent or discard the new or the old one. The output of this module consists of a syntactically-correct *Nile* intent that LUMI also considers to be ambiguity-free.

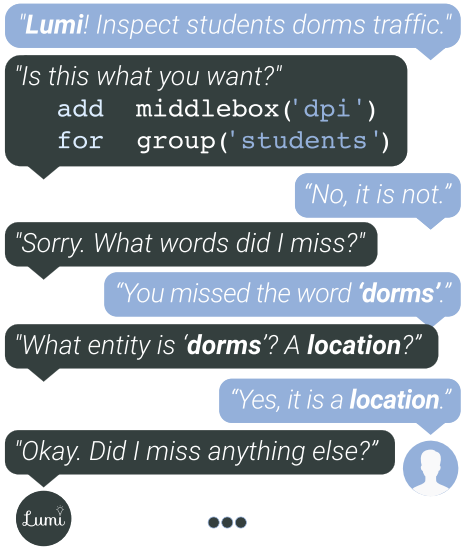


Fig. 4. LUMI's feedback mechanism in action (simplified).

Note that we lose some precision in the process by not using a deterministic ambiguity detection method, but we can cover every type of possible ambiguities between *Nile* intents. Despite the precision loss and the training requirements, this approach enables real-time ambiguity detection and limits the complexity of the problem to the number of previously deployed intents. However, as it is impossible for a machine-learned classifier to achieve 100% classification accuracy, having this ambiguity module does not eliminate the need for formal verification of network properties, perhaps as a separate module running post-deployment of confirmed intents. Still, being able to alert operators of possible mistakes in real time is an appealing feature in any networking management system.

VI. OPEN CHALLENGES

We consider LUMI to be a promising first step toward fully realizing IBN, but several engineering and research challenges remain. Among the engineering challenges, we mention the need to get LUMI "production-ready" (*i.e.,* to be used in real-world production networks that consist of legacy equipment from a variety of different vendors); the requirement to extend LUMI for other use-cases (*e.g.,* data-center networks that rely on technologies such as VXLANs or NVGRE to share network infrastructure between tenants); and the desire to accommodate out-of-vocabulary natural language intents (*e.g.,* referencing to VLANs by group name vs IDs or dealing with misspellings and grammatical errors).

In addition to the engineering challenges that a production deployment of LUMI entail, the current version of LUMI also identifies a number of research challenges that arise from attempts at designing a more robust or "future-proof" version of LUMI and include the following problem areas:

Using formal network verification techniques. Recent efforts on formal network verification [1], [56], [57] provide sub-second verification of waypointing, reachability, and isolation properties. However, the potential of these existing efforts for verifying configurations generated by LUMI-compiled

intents notwithstanding, their adaptation to LUMI-specific settings remains an open problem. At the same time, approaches such as automated testing of intents in emulated environments [58] or methods for detecting and resolving conflicting intents [21], [28] could benefit the validation of intent behavior prior to deployment.

Ensuring the correctness of LUMI-generated configurations. Information extraction from natural language is inherently prone to errors. LUMI alleviates this problem by asking operators for feedback and using their responses to check if the extracted information is correct. On the one hand, leveraging the emerging generation of LLMs does not solve this problem. Without knowing the root causes of the mistakes that these LLMs make, operators will object to deploying LMM-generated configurations in security- and safety-critical settings like their production network environments. On the other hand, by applying state-of-the-art ML pipelines such as TRUSTEE [12] that explain ML model-based decisions by means of purposefully-extracted decision trees, operators could examine in detail how a particular NER-based model extracts information from natural language, thus gaining trust in how LUMI parses their intents.

Post-deployment behavior monitoring. To further improve the trust of operators in LUMI-generated configurations, we envision a LUMI pipeline that will also include a monitoring module for collecting data that can be used to check if the deployed configurations respect the intents produced by the refinement process and achieve the objective(s) that the operator expressed in the first place. However, deciding which traffic, devices, and properties to monitor and which data to collect will require instrumenting networks with an unprecedented level of control that is only possible by leveraging emerging programmable data plane technologies [59], [60].

The need for data. The LUMI use case highlights the scarcity of intent-related data, particularly labeled data. This shortage is partly due to the diversity of intent representation languages in the IBN literature. At the same time, publicly available datasets for general-purpose NLP research are an intrinsic aspect of the NLP research community [61], [62] and one of the main reasons for its success and the rapid advances the field has experienced. Moreover, the NLP community has developed tools to automate part of the tedious and manual process of curating an annotated dataset of natural language snippets [63]. We argue that the networking community would be wise to take advantage of these advances and tools to create rich and privacy-safe intent corpora to enable and ensure further progress in the field.

VII. CONCLUSION

In this paper, we present an in-depth account of how NLP, in conjunction with emerging technologies and recent advances in other relevant scientific areas, can be leveraged to allow operators to use natural language to configure a network automatically. In particular, building on our previously proposed LUMI conversational interface [14], we augment it with a new module that is designed to detect ambiguities between newly considered intents and already deployed intents and detail how this extended LUMI pipeline provides the means for operators to gain confidence that the LUMI-generated intents correctly and accurately reflect their original intents so they can trust the resulting configurations and deploy them in their production networks. However, despite addressing and resolving many of the challenges that realizing the vision of IBN with natural language poses, this paper also shows that much work remains. For one, while the design choices for LUMI's different modules resulted in a working prototype, developing a production-ready version of LUMI will require incorporating additional or improving existing features. However, LUMI's modular design can readily accommodate such extensions and improvements. Moreover, since LUMI in its current form is mainly intended for use in campus networks, supporting other environments (*e.g.,* home or enterprise networks) will most likely require that the set of *Nile* operations and functions (and in turn the set of LUMI entities) be judiciously extended.

ACKNOWLEDGMENT

The authors thank the editors and anonymous reviewers of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT for providing them with valuable feedback.

REFERENCES

[1] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proc. ACM SIGCOMM*, 2017, pp. 155–168. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098834

[2] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proc. ACM SIGCOMM*, 2016, pp. 328–341. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934909

[3] J. Apostolopoulos. "Improving networks with artificial intelligence." Oct 2020. [Online]. Available: https://blogs.cisco.com/networking/improving-networks-with-ai

[4] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-based networking—Concepts and definitions," IETF, Internet-Draft draft-IRTF-NMRG-IBN-concepts-definitions-00, Dec. 2019. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-ibn-concepts-definitions-00

[5] G. Narcisi. "Juniper networks CEO: 'The goal now is a self-driving network,"' May 2020. Accessed: May 26, 2025. [Online]. Available: https://www.crn.com/news/networking/juniper-networks-ceo-the-goal-now-is-a-self-driving-network-

[6] H. H. Liu, "The practice of network verification in Alibaba's global WAN," May 2021. [Online]. Available: https://netverify.fun/the-practice-of-network-verification-in-alibaba-global-wan/

[7] B. Koley, "The zero touch network," presented at 12th Int. Conf. Netw. Service Manag., 2016.

[8] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Net2Text: Query-guided summarization of network forwarding behaviors," in *Proc. USENIX NSDI*, 2018, pp. 609–623. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/birkner

[9] A. Alsudais and E. Keller, "Hey network, can you understand me?" in *Proc. IEEE INFOCOM Workshops*, May 2017, pp. 193–198.

[10] K. Dzeparoska, J. Lin, A. Tizghadam, and A. Leon-Garcia, "LLM-based policy generation for intent-based management of applications," in *Proc. 19th Int. Conf. Netw. Service Manage. (CNSM)*, 2023, pp. 1–7.

[11] S. K. Mani et al., "Enhancing network management using code generated by large language models," in *Proc. 22nd ACM Workshop Hot Topics Netw.*, 2023, pp. 196–204. [Online]. Available: https://doi.org/10.1145/3626111.3628183

[12] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "AI/ML and network security: The emperor has no clothes," in *Proc. ACM CCS*, 2022, pp. 1537–1551

[13] M. F. Franco et al., "SecBot: A business-driven conversational agent for cybersecurity planning and management," in *Proc. CNSM*, 2020, pp. 1–7.

[14] A. S. Jacobs et al., "Hey, Lumi! Using natural language for intent-based network management," in *Proc. USENIX ATC*, Jul. 2021, pp. 625–639. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/jacobs

[15] A. Leivadeas and M. Falkner, "A survey on intent based networking," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 625–655, 1st Quart., 2023.

[16] K. Mehmood, K. Kralevska, and D. Palma, "Intent-driven autonomous network and service management in future cellular networks: A structured literature review," *Comput. Netw.*, vol. 220, Jan. 2023, Art. no. 109477. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622005114

[17] E. J. Scheid et al., "INSpIRE: Integrated NFV-based intent refinement environment," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag.*, 2017, pp. 186–194.

[18] J. Kim et al., "IBCS: Intent-based cloud services for security applications," *IEEE Commun. Mag.*, vol. 58, no. 4, pp. 45–51, Apr. 2020.

[19] M. Cheminod, L. Durante, L. Seno, F. Valenza, and A. Valenzano, "A comprehensive approach to the automatic refinement and verification of access control policies," *Comput. Security*, vol. 80, pp. 186–199, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404818303870

[20] C. Prakash et al., "PGA: Using graphs to express and automatically reconcile network policies," in *Proc. ACM SIGCOMM*, 2015, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787506

[21] A. Abhashkumar, J. Kang, S. Banerjee, A. Akella, Y. Zhang, and W. Wu, "Supporting diverse dynamic intent-based policies using janus," in *Proc. ACM CoNEXT*, 2017, pp. 296–309. [Online]. Available: http://doi.acm.org/10.1145/3143361.3143380

[22] B. Tian et al., "Safely and automatically updating in-network ACL configurations with intent language," in *Proc. ACM SIGCOMM*, 2019, pp. 214–226. [Online]. Available: https://doi.org/10.1145/3341302.3342088

[23] M. Riftadi and F. Kuipers, "P4I/O: Intent-based networking with P4," in *Proc. IEEE NetSoft*, 2019, pp. 438–443.

[24] K. Dzeparoska, N. Beigi-Mohammadi, A. Tizghadam, and A. Leon-Garcia, "Towards a self-driving management system for the automated realization of intents," *IEEE Access*, vol. 9, pp. 159882–159907, 2021.

[25] L. Velasco et al., "End-to-end intent-based networking," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 106–112, Oct. 2021.

[26] J. Massa, S. Forti, F. Paganelli, P. Dazzi, and A. Brogi, "Towards declarative intent processing and conflict resolution in IBN," in *Proc. 16th IEEE/ACM Int. Conf. Utility Cloud Comput.*, 2023, pp. 1–2.

[27] J. Massa, S. Forti, F. Paganelli, P. Dazzi, and A. Brogi, "A declarative reasoning approach to conflict management in intent-based networking," in *Proc. 27th Conf. Innov. Clouds, Internet Netw.*, 2024, pp. 228–233.

[28] Z. Xiaoang, A. Leivadeas, and M. Falkner, "Intent based networking management with conflict detection and policy resolution in an enterprise network," *Comput. Netw.*, vol. 219, Dec. 2022, Art. no. 109457.

[29] B. Li, X. Deng, and P. Zhang, "A policy conflict detection mechanism for intent-based networking," in *Proc. 9th IEEE Int. Conf. Cloud Comput. Intell. Syst.*, 2023, pp. 164–175.

[30] J. Huang, C. Yang, S. Kou, and Y. Song, "A brief survey and implementation on AI for intent-driven network," in *Proc. 27th Asia–Pacific Conf. Commun. (APCC)*, 2022, pp. 413–418.

[31] Y. Huang et al., "Large language models for networking: Applications, enabling techniques, and challenges," *IEEE Netw.*, vol. 39, no. 1, pp. 235–242, Jan. 2025.

[32] J. Lin, K. Dzeparoska, A. Tizghadam, and A. Leon-Garcia, "AppleSeed: Intent-based multi-domain infrastructure management via few-shot learning," in *Proc. IEEE 9th Int. Conf. Netw. Softwarization (NetSoft)*, 2023, pp. 539–544.

[33] K. Dzeparoska, A. Tizghadam, and A. Leon-Garcia, "Intent assurance using LLMs guided by intent drift," 2024, *arXiv:2402.00715*.

[34] A. Mekrache, A. Ksentini, and C. Verikoukis, "Intent-based management of next-generation networks: An LLM-centric approach," *IEEE Netw.*, vol. 38, no. 5, pp. 29–36, Sep. 2024.

[35] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "NAIL: A network management architecture for deploying intent into programmable switches," *IEEE Commun. Mag.*, vol. 62, no. 6, pp. 28–34, Jun. 2024.

[36] C. Wang, M. Scazzariello, A. Farshin, D. Kostic, and M. Chiesa, "Making network configuration human friendly," 2023, *arXiv:2309.06342v1*.

[37] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2019. [Online]. Available: https://web.stanford.edu/ jurafsky/slp3/

[38] M. Marrero, J. Urbano, S. Sánchez-Cuadrado, J. Morato, and J. M. Gómez-Berbís, "Named entity recognition: Fallacies, challenges and opportunities," *Comput. Stand. Interfaces*, vol. 35, no. 5, pp. 482–489, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548912001080

[39] J. Chiu and E. Nichols, "Named entity recognition with bidirectional LSTM-CNNs," *Trans. ACL*, vol. 4, pp. 357–370, Jul. 2016. [Online]. Available: http://aclweb.org/anthology/Q16-1026

[40] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. ICML*, 2001, pp. 282–289. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.655813

[41] A. Vaswani et al., "Attention is all you need," 2017, *arXiv:1706.03762*.

[42] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 50–70, Jan. 2022.

[43] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[44] T. Bunk, D. Varshneya, V. Vlasov, and A. Nichol, "DIET: Lightweight language understanding for dialogue systems," 2020, *arXiv:2004.09936*.

[45] J. Achiam, "Gpt-4 technical report," 2023, *arXiv:2303.08774*.

[46] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," in *Proc. ACM SIGCOMM Workshop Self-Driving Netw.*, 2018, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/3229584.3229590

[47] "Lumi supplemental material," Lumi, 2024. [Online]. Available: https://lumichatbot.github.io/

[48] R. Soulé et al., "Merlin: A language for managing network resources," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2188–2201, Oct. 2018.

[49] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable dynamic network control," in *Proc. USENIX NSDI*, 2015, pp. 59–72. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789775

[50] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. USENIX NSDI*, 2013, pp. 15–28.

[51] A. Fogel et al., "A general approach to network configuration analysis," in *Proc. USENIX NSDI*, 2015, pp. 469–483.

[52] M. C. D. Marneffe, A. N. Rafferty, and C. D. Manning, "Finding contradictions in text," in *Proc. Conf. ACL*, 2008, pp. 1039–1047. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84859888583&partnerID=40&md5=624f6d85fc589442d7af96db6236f74c

[53] G. A. Miller, "WordNet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[54] M. Honnibal and I. Montani, "SpaCy 2: Natural language understanding with bloom embeddings, convolutional neural networks, and incremental parsing," 2017. [Online]. Available: https://spacy.io/

[55] A. Amokrane, R. Langar, R. Boutaba, and G. Pujolle, "Flow-based management for energy efficient campus networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 4, pp. 565–579, Dec. 2015.

[56] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast and general network verification," in *Proc. USENIX NSDI*, Feb. 2020, pp. 201–209. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/abhashkumar

[57] S. Prabhu, K. Y. Chou, A. Kheradmand, B. Godfrey, and M. Caesar, "Plankton: Scalable network configuration verification through model checking," in *Proc. USENIX NSDI*, Feb. 2020, pp. 953–967. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/prabhu

[58] P. Alcock, B. Simms, W. Fantom, C. Rotsos, and N. Race, "Improving intent correctness with automated testing," in *Proc. IEEE NetSoft*, 2022, pp. 61–66.

[59] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proc. ACM SIGCOMM*, 2018, pp. 357–371. [Online]. Available: https://doi.org/10.1145/3230543.3230555

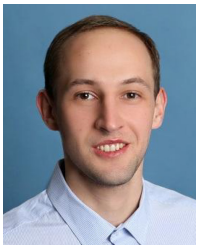
[60] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," in *Proc. ACM SIGCOMM*, 2020, pp. 662–680. [Online]. Available: https://doi.org/10.1145/3387514.3405894

[61] J. Brownlee. "Datasets for natural language processing." Aug 2020. [Online]. Available: https://machinelearningmastery.com/datasets-natural-language-processing/

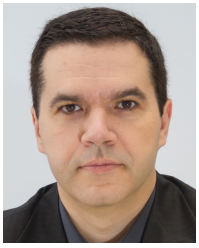
[62] Niderhoff, 2011, "Alphabetical List of free/public domain datasets with text data for use in natural language processing (NLP)," NLP-datasets. [Online]. Available: https://github.com/niderhoff/nlp-datasets
[63] "Chatito: Generate datasets for AI Chatbots, NLP tasks, named entity recognition or text classification models using a simple DSL!" Rodrigopivi. Accessed: Dec. 2022. [Online]. Available: https://github.com/rodrigopivi/Chatito

Arthur Selle Jacobs received the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Brazil. His research interests include network management, self-driving networks, and artificial intelligence. He was awarded the 2023 IETF Applied Networking Research Prize for his work on evaluating machine learning for network security, and the 2020 IBM Ph.D. Fellowship Award for academic excellence and expertise in pioneering research areas, including artificial intelligence, data science, and security.

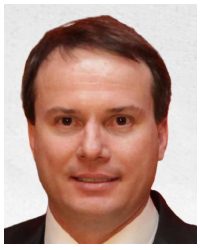
Ricardo José Pfitscher is a Professor Researcher with the Federal University of Santa Catarina, Brazil. His research interests include network management, network functions virtualization, network performance, and general applications of artificial intelligence.



Rafael Hengen Ribeiro received the master's degree under the supervision of Prof. L. Z. Granville from the Federal University of Rio Grande do Sul in 2020, with a thesis titled "A Bottom-Up Approach for Extracting Network Intents." His research interests include intent-based networking, segment routing, and network programming.



Lisandro Zambenedetti Granville (Senior Member, IEEE) is a Full Professor with the Institute of Informatics, Federal University of Rio Grande do Sul, and the CEO of the Brazilian National Research Network. His research interests include the management of network virtualization, intent-based networking, SDN, NFV, and network programmability.



Ronaldo Alves Ferreira received the B.Sc. degree in computer science from UFMS in 1992, the M.S. degree in computer science from the University of Campinas in 1998, and the Ph.D. degree in computer science from Purdue University in 2006. He is a Full Professor of Computer Science with the College of Computing, UFMS. He was a Visiting Research Scholar and a Visiting Associate Professor with Princeton University from 2014 to 2016. His research interests are in computer networks and distributed systems. He served as the Chair of the Special Interest Group on Computer Networks and Distributed Systems of the Brazilian Computing Society from 2011 to 2013. He was a member of the Board of Administration of the Brazilian National Research and Educational Network from 2011 to 2013. He was a member of the Board of Directors of the Brazilian National Laboratory of Computer Networks from 2014 to 2023.

Walter Willinger (Fellow, IEEE) received the Dipl. Math. degree from the ETH Zürich, and the M.S. and Ph.D. degrees in operations research and industrial engineering from Cornell University. He is Chief Scientist at NIKSUN Inc., a Princeton-based cybersecurity company. Before joining NIKSUN, he worked at AT&T Labs-Research, Florham Park, NJ, USA, and at Bellcore Applied Research, Morristown, NJ, USA. He is a co-recipient of the 1995 IEEE Communications Society W.R. Bennett Prize Paper Award and the 1996 IEEE W.R.G. Baker Prize Award, of the 2005 and 2016 ACM/SIGCOMM Test-of-Time Paper Awards, and of the IRTF Applied Networking Research Prize in 2023; and the recipient of the 2024 IEEE Internet Award. He is a Fellow of ACM in 2005, of AT&T in 2007, of SIAM in 2009, and of AAIA in 2023.

Sanjay G. Rao received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology at Madras, and the Ph.D. degree from the School of Computer Science, Carnegie Mellon University. He is a Professor with the Elmore Family School of Electrical and Computer Engineering, Purdue University, with research interests in computer networking, especially network verification and synthesis, and internet video. He has been a Visiting Researcher with AT&T Research, Google, and Princeton University. He is a recipient of the National Science Foundation Career Award, and has received the ACM Sigmetrics Test of Time Award for his work on End System Multicast. He has served as an Associate Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING, as an Area Technical Program Chair of IEEE Infocom, and has chaired the ACM Sigcomm Distinguished Dissertation Award Committee. He is an ACM Distinguished Member.

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - ROR identifier: 00x0ma614