

LLM 기반 네트워크 기능 설치 명령어 생성 및 자가 수정을 위한 RAG 기법 연구

남 석 현*, 홍 원 기°

A Study of RAG for LLM-based NF Configuration Generation and Self-healing

Sukhyun Nam*, James Won-Ki Hong°

요 약

본 연구는 네트워크 운용자가 자연어로 구성 요청을 입력하면 대규모 언어 모델(Large Language Model, LLM)을 통해 CNF (Container-Native Network Function) 및 VNF (Virtualized Network Function)를 자동으로 설치·구성하는 NFV (Network Function Virtualization) 자동화 시스템을 제안한다. 제안된 시스템은 NDT (Network Digital Twin)를 활용한 사전 검증, 로그 기반 오류 탐지, RAG (Retrieval-Augmented Generation)를 통한 오류 수정 기능을 포함하며, 특히 Log-TF-IDF 기반 Input filtering과 Judge LLM 기반 Output filtering을 적용하여 LLM의 자가 수정 능력을 크게 향상시킨다. Kubernetes 환경에서 구현된 본 시스템은 자연어 기반 네트워크 구성 자동화 프레임워크가 실제 NFV 운영 환경에서도 실현 가능성을 보여준다.

Key Words : IBN, AI based Network Management, NFV, LLM, RAG

ABSTRACT

This study proposes an NFV (Network Function Virtualization) automation system that enables network operators to input configuration requests in natural language, which are then processed by a Large Language Model (LLM) to automatically install and configure CNFs (Container-Native Network Functions) and VNFs (Virtualized Network Functions). The proposed system incorporates NDT (Network Digital Twin)-based validation, log-based error detection, and RAG (Retrieval-Augmented Generation)-based error correction. In particular, it significantly enhances the self-healing capability of the LLM through Log-TF-IDF-based input filtering and Judge LLM-based output filtering. Implemented in a Kubernetes environment, the system demonstrates the feasibility of a natural-language-driven automated configuration framework for NFV.

*이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(RS-2024-00392332, 6G 네트워크 통합 지능형
면 기술 개발, 50%)과 삼성전자의 지원(50%)을 받아 수행되었습니다.

• First Author : Pohang University of Science and Technology Institute of Artificial Intelligence, obiwon96@postech.ac.kr

° Corresponding Author : Pohang University of Science and Technology Department of Computer Science Engineering,
jwkhong@postech.ac.kr

논문번호 : KNOM2025-02-008, Received November 5, 2025; Revised December 15, 2025; Accepted December 26, 2025

I. 서론

4G/5G 상용화 이후 네트워크에 연결된 디바이스 수와 트래픽 양이 급증함에 따라 네트워크의 규모와 복잡도는 지속적으로 증가하고 있다. 기존 네트워크 관리는 전문지식을 갖춘 운용자의 경험과 수작업에 의존해왔으나, 복잡한 토폴로지와 다양한 프로토콜을 장치별 명령어로 수동 관리하는 방식에는 본질적인 한계가 존재한다.

수작업 기반 관리 방식은 인적 오류(human error)로 인한 장애 위험을 증가시키며, 네트워크 환경 변화에 신속하게 대응하기 어렵다는 문제가 있다. 이에 따라 네트워크 구성 및 운용 자동화의 필요성이 크게 대두되고 있으며, 특히 대규모 언어 모델(LLM)을 활용하여 자연어 기반 구성 요청을 자동으로 해석·실행하는 연구가 활발히 진행되고 있다. 이러한 접근은 의도 기반 네트워킹(Intent-Based Networking, IBN)으로 불리며, LLM의 발전과 함께 연구 범위가 확장되고 있다[1].

현재 IBN 연구는 링크 단절, IP 설정, 인테트 번역과 같은 기본 기능 자동화에 집중되어 있으며, NF (Network Function) 설치 자동화는 아직 초기 단계에 머물러 있다[2-5]. 그러나 최근 LLM의 성능 향상으로 보다 복잡한 구성 명령의 이해 및 자동 실행 가능성이 증가하고 있다.

그림 1은 LLM 기반 네트워크 자동 구성 시스템의 개요이다. 자연어 명령은 LLM을 통해 분석되며, LLM은 분석 결과 및 매뉴얼을 기반으로 설정 명령어를 생성한다. 생성된 명령어는 실제 네트워크 또는 시뮬레이션 환경에서 검증되며, 필요 시 자동 수정이 반복되는 폐쇄 루프(closed-loop) 구조를 구성할 수 있다.

본 논문에서는 동일한 접근을 CNF (Container-Native Network Function) 설치 자동화

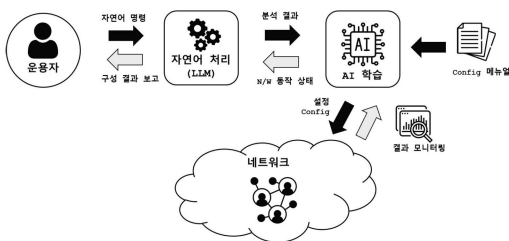


그림 1. 의도 기반 네트워킹-네트워크 설정 개요
Fig. 1. IBN-Configuration Outline

에 적용한다. 사용자가 자연어로 CNF 설치 명령을 입력하면 LLM이 MOP를 참고하여 설정 명령어 (Python/Ansible)를 생성하고, 그 결과를 자가 수정 (self-healing)하는 구조의 AI Agent를 설계·검증한다. 본 연구는 Kubernetes 환경에서 CNF 배포 (deployment)를 자동화하는 것을 목표로 하며, 여기서 배포란 컨테이너 생성 (create), NF 설치 및 구성 (configuration) 작업 전체를 포함한다. 본 연구에서는 실험환경이 Kubernetes 기반이므로 CNF 중심이지만, 동일한 방법론으로 VNF (Virtualized Network Function)도 포괄적으로 다룰 수 있는 연구이다.

설정 명령어를 자동 생성하고 이를 기반으로 NF 배포를 정확히 수행하는 작업은 결코 단순하지 않다. 코드 생성 기능을 갖춘 LLM이라 하더라도 복잡한 배포 시나리오에서는 오류 없이 정확한 구성을 보장하기 어렵고, 오류 발생 시 스스로 원인을 분석하고 수정하는 능력 또한 제한적이다. 이를 해결하기 위해 본 연구는 관련 문서 제공을 위한 Retrieval-Augmented Generation (RAG) [6], 그리고 RAG 정확도를 높이기 위해 Input/Output Filtering 기법을 적용하였다.

본 논문의 구성은 다음과 같다. 2장에서는 연구 배경 및 관련 연구를 소개한다. 3장 시스템 설계를 설명하며, 4장에서는 실험 및 평가 결과를 제시한다. 마지막으로 5장에서 결론을 제시한다.

II. 연구 배경 및 관련 연구

2.1. 의도 기반 네트워킹 (IBN)

IBN (Intent-Based Networking)은 운용자가 자연어 또는 고수준 정책 형태로 입력한 의도(intent)를 네트워크 구성 명령으로 자동 변환하는 방식의 네트워크 관리 패러다임이다[1]. 기존의 장치 중심 명령 기반 방식과 달리, IBN은 요구사항을 추상적 정책으로 표현하여 관리 복잡성을 줄이고 자동화 수준을 높일 수 있는 장점이 있다.

현재 LLM을 직접 활용하여 NF 배포를 완전 자동화한 사례는 존재하지 않지만, 네트워크 관리의 각 분야에서 LLM을 활용한 다양한 연구들이 진행되고 있다. D. Donadel 외 [2]는 LLM이 네트워크 토폴로지 정보를 이해하고 시스템 관리자 수준의 질의에 응답할 수 있는지를 평가하였다. 해당 연구는 LLM이 토폴로지 구조를 해석하고, 경로를 계산

하는 데 초점을 맞추었으며, LLM이 복잡한 네트워크 구조를 완전히 이해하기에는 아직 한계가 있음을 시사하였다. 해당 연구는 특히 프롬프트 엔지니어링(prompt engineering)의 중요성을 강조하였다.

J. Lin 외 [3]는 자연어로 정의된 다중 도메인 네트워크 요구사항을 상세 구성 명령어로 변환하기 위해 LLM을 활용하는 실험을 수행하였다. 연구에서는 특정 네트워크 설정 예시를 few-shot 학습 형태로 제공했을 때 구성 변환 정확도가 크게 향상됨을 확인하였다.

E. Jeong 외 [4]는 스위치 설정을 보조하기 위해 LLM 기반 도구 “S-Witch”를 제안하였다. S-Witch는 사용자 자연어 입력을 해석하여 정교한 스위치 구성 명령을 생성하며, 설계 단계와 구성 단계를 담당하는 LLM을 분리한 계층형 구조를 활용한다.

VNF/SFC (Service Function Chain) 관리를 자동화하기 위한 LLM 활용 연구도 일부 존재한다. K. Dzeperaska 외 [5]는 LLM을 이용하여 애플리케이션 구성 및 배포, SFC 동작을 제어하는 관리 정책(policy)을 자동 생성하는 방법을 제안하였다. N. V. Tu 외 [7]는 LLM을 활용한 의도 기반 NFV 구성 프레임워크를 제안하였다. 연구에서는 VNF 및 SFC와 관련된 6개의 동작(action)을 정의하고, 이러한 동작을 포함한 자연어 문장을 미리 정의한 NFV-Intent 템플릿으로 변환하는 실험을 수행하였다.

Nephio [8]는 Google과 Linux Foundation이 2022년에 시작한 오픈소스 프로젝트로, 분산형 멀티벤더 텔코 네트워크에서 CNF 자동화와 오케스트레이션을 단순화하는 것을 목표로 한다. Nephio는 Kubernetes [9] 기반 IBN과 GitOps 개념을 결합하여 네트워크 기능 및 서비스를 선언적 방식으로 구성할 수 있도록 한다. 현재 Nephio는 5G RAN 오케스트레이션, 엣지 네트워크 자동화, 멀티도메인 서비스 프로비저닝 등에서 그 가능성이 검증되고 있으며, 배포 후 지속적 모니터링, 자가 수정, 자동 롤백 기능을 중점적으로 개발 중이다. 그러나 2025년 현재 선언형 CRD + 템플릿 기반 워크플로우를 주류로 유지하고 있으며, 자연어 기반 NF 설치 자동화—본 논문의 핵심 주제—는 GenAI 연계 가능성으로 제안된 초기 실험 단계에 머물러 있다.

범용 LLM은 네트워크 구성에 대한 전문적인 지식을 충분히 내재하고 있지 않아, 실제 환경을 정확히 구성하기 위해서는 매우 상세한 구성 맥락을 LLM에 제공해야 하거나, 범용 LLM을 파인튜닝하

여 사용해야 한다. 하지만 이를 통해서도 네트워크 관리는 아직 복잡한 목표이며, LLM이 옳은 응답을 내도록 하는 것이 현재 IBN 연구에서 해결해야 할 핵심 과제이다.

2.2. Retrieval-Augmented Generation (RAG)

일반 목적의 지식을 기반으로 학습된 LLM은 특정 분야의 전문성을 요구하는 도메인 특화 작업에 적용될 때 본질적인 한계를 보인다. 해당 문제를 해결하기 위해 P. Lewis 외 [6]는 Retrieval-Augmented Generation (RAG) 프레임워크를 제안하였다. RAG는 기존의 sequence-to-sequence 생성 과정에 검색 모듈을 통합함으로써, 모델이 추론 과정에서 대규모 코퍼스에서 관련 문서를 동적으로 검색하고, 해당 외부 텍스트 근거를 바탕으로 출력 결과를 생성할 수 있도록 한다.

RAG의 핵심 구성 요소는 밀집 벡터(dense vector) 기반의 검색 메커니즘이다. 대표적으로 DPR (Dense Passage Retrieval)과 같은 사전학습 임베딩 모델을 사용하여 질의(query)와 코퍼스 문서를 동일한 벡터 공간으로 변환하며, dot product 또는 cosine similarity와 같은 유사도 합수를 통해 관련성이 높은 상위 k개의 문서를 효율적으로 검색한다.

RAG의 주요 장점은 출력이 외부 근거에 기반하여 사실적으로 정확하며 문맥적으로 일관된다는 점이다. 또한 모델 자체를 재학습할 필요 없이 최신 정보를 활용할 수 있어, 의료, 기술 분야 등 전문 지식이 요구되는 영역에서 특히 효과적이다.

III. 시스템 디자인

3.1. AI Agent 디자인

본 장에서는 본 연구에서 설계한 LLM 기반 NFV 자동 구성 AI Agent의 전체 구조와, Input/Output Filtering을 포함한 RAG 설계에 대해 설명한다. 전체 구조는 오픈스택 (OpenStack)에서 VNF 설정 자동화 시스템을 제안했던 사전 연구 [10]의 구조에 기반하며, RAG 모듈이 추가된 구조이다. 시스템 전체 구조는 그림 2와 같다.

우리는 범용 목적으로 개발된 LLM이 NF 설정과 관련된 사전 지식이 없을 수 있기 때문에 NF 설정

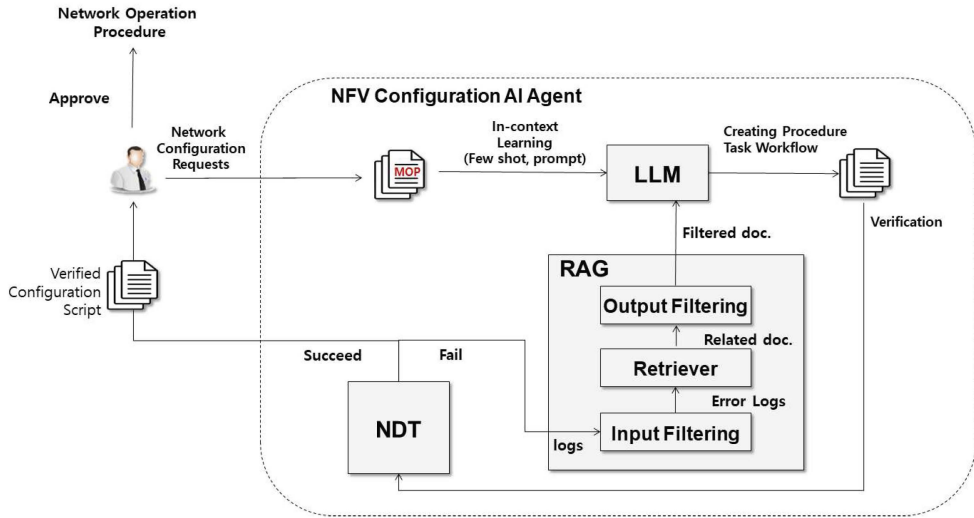


그림 2. LLM 기반 NFV 구성 에이전트 구조도
Fig. 2. LLM-based NFV Configuration AI Agent Structure

에 관한 문서 (MOP)를 입력으로 함께 사용하는 구조를 사용하였다. LLM은 자연어와 MOP를 입력받아 사용자의 요청에 따라 설치를 자동화 하는 Python/Ansible 코드를 생성한다. 생성된 코드는 실제 네트워크에 적용되기 전에 가상 네트워크 환경인 Network Digital Twin (NDT)에서 검증 과정을 거친다. 이 때 파싱 (parsing) 및 테스트 모듈이 LLM의 응답에서 코드를 추출하고 NDT와 통신하여 실행함으로써 설치가 정상적으로 완료되는지 검증한다. 검증 과정은 사전에 규칙 기반(rule-based)으로 설치하고자 하는 NF의 동작을 검증할 수 있는 프로그램을 포함한다.

NDT에서 정상적으로 NF가 설정되었다면 해당 워크플로우 코드를 사용자에게 전달함으로써 실제 네트워크에 적용될 수 있도록 하면 되지만, LLM의 실수로 원하는대로 설정이 안되었을 수가 있다. 원인은 다양할 수 있으며, NF가 제대로 설치가 안되었을 수도 있고, 설치는 되었지만 사용자가 설정하고자 하는 기능이 설정되지 않을 수도 있다. 이러한 경우에 제한하는 AI Agent는 LLM이 문제를 인지하고 자가 수정할 수 있도록 추가적인 동작을 지시함으로써 폐쇄 루프를 수행하도록 한다. 테스트 모듈은 NDT와 통신하며 NF의 설치 및 설정을 확인하며 만약 에러가 발생한 경우, NF 설치 과정에서 발생하는 로그 등을 수집하여 LLM에 전달함으로써 이를 고칠 수 있도록 한다.

우리는 해당 과정을 위해 사전 연구 [10]에서 테

스트 모듈을 설계하였으며, Github 레포지토리 [11]를 통해 공개하였다. 우리는 사전 연구에서 NDT에서 발생한 에러 로그를 단순히 LLM에게 전달해주는 것으로는 LLM이 잘못된 원인 분석을 제대로 못하고 이전 응답과 유사한 응답을 반환하는 경우가 많으며, 이 때문에 재시도를 통한 성공률이 굉장히 낮은 것을 확인하였다. 이 때문에 이번 연구에서 제안하는 AI Agent는 NDT에서 발생한 로그와 연관된 문서를 반환하도록 하는 RAG 모듈을 사용한다.

3.2. RAG 모듈

RAG 모듈은 NDT에서 발생한 에러를 고치기 위하여 사용한다. RAG는 2.2장에서 설명하였듯이 관련된 문서를 찾기 위해 지식 베이스(Knowledge Base), 벡터 데이터베이스 (Vector DB), 리트리버 (Retriever)로 구성된다.

기존 LLM은 주로 질의응답(QA) 시스템에서 사용되며, RAG 또한 주로 QA 시나리오를 지원하기 위해 특정 엔티티(entity)와 관련된 사실 정보를 추출 하는 형태로 설계되어 왔다. 그러나 본 연구의 목적은 사실 정보를 찾는 것이 아니라, 로그 기반 오류를 해석하고 해결하는 데 필요한 문맥적 정보만을 제공하는 것이다. 로그 그 자체는 문서에 없을 수 있으며, 로그와 가장 유사한 문맥적 정보를 찾는 것이 목표이다. 이 때문에 기존 QA 중심 RAG를 그대로 적용하기에는 한계가 있으며, 본 연구에서는 오류 해결을 위해 문서 구조를 유지하면서 의

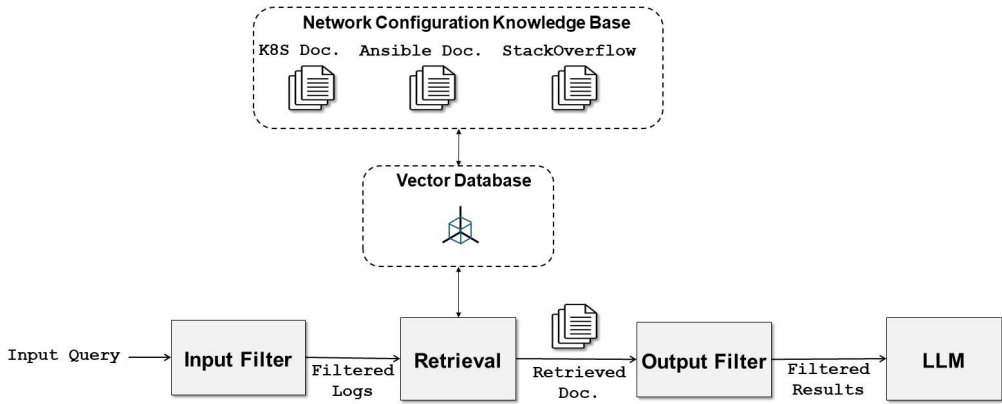


그림 3. 입/출력 필터 포함 RAG 구조도
Fig. 3. RAG Structure with Input/Output Filter

미적으로 관련된 문맥만을 추출할 수 있도록 맞춤형 RAG 프레임워크를 설계하였다. 전체 구조는 그림 3과 같으며, 리트리버의 앞뒤의 내용을 필터링하는 Input/Output 필터 과정을 포함한다.

1) 지식 베이스

NFV 구성과 관련한 지식을 저장한 데이터베이스이다. 본 연구에서는 Kubernetes 환경에서 Python/Ansible로 CNF를 설치하는 것을 목표로 하였기 때문에 Kubernetes의 공식 홈페이지의 도큐먼트, Ansible의 공식 홈페이지 도큐먼트를 크롤링하여서 각 API 별로 별개의 문서를 저장하였다. 또한, 프로그래밍 관련 질문 사이트인 StackOverflow에서 Kubernetes와 각 NF 관련 질문글을 StackAPI [12]를 통해 저장하였다. 각각의 문서는 API 이름과 질문 제목을 문서의 제목으로 저장하였다.

2) 벡터 데이터베이스 및 리트리버

사전 학습된 문장 임베딩 모델을 이용하여 지식 베이스 내 모든 문서의 제목을 벡터화한다. StackOverflow 데이터의 경우 질문 본문도 추가로 임베딩 생성에 사용된다. RAG 사용 시에는 입력된 오류 로그 역시 동일한 임베딩 모델을 통해 벡터로 변환되며, 입력 로그와 가장 높은 벡터 유사도를 보이는 문서(또는 제목 - 질문 쌍)를 검색하여 RAG 결과로 반환한다.

그러나 해당 과정을 실행하는 과정에서 두 가지 주요 문제가 발생하였다.

i. 입력 로그에서 불필요한 정보까지 벡터화될 경우, 전혀 관련 없는 문서가 검색됨

ii. 지식베이스 내에 실제 오류와 관련된 문서가 존재하지 않는 경우, 유사도 기반 검색 특성상 거리

가 가장 가까운 문서를 반환하게 되어 비관련 문서가 임의로 선택

두 경우 모두 LLM이 오류 수정에 도움이 되지 않는 잘못된 문서를 입력받게 되어 결과적으로 수정 정확도가 오히려 떨어지는 문제가 발생한다. 본 연구에서는 각각 입력 필터링, 출력 필터링을 통해 문제점을 해결하고자 하였다.

3) Input 필터

앞서 언급한 첫 번째 문제를 해결하기 위해 로그에서 실제 오류 로그만 분리하여 추출하기 위한 기능이다. 우리는 이전 연구[13,14]에서 정상·비정상 네트워크 상태에서 로그 패턴 분포를 비교 분석하여 스위치 이상 탐지 가능성을 연구하였다. 이 과정에서 NLP 분야의 대표적 통계 기법인 TF-IDF [15]에서 영감을 받아, 로그 패턴의 회귀도를 정량화하기 위한 **Log-TF-IDF**를 제안하였다. 전통적 TF-IDF가 문서 내 단어의 중요도를 측정한다면, Log-TF-IDF는 이를 **정상 로그 내 이벤트 회귀도**로 변환한 개념이다.

TF-IDF는 TF와 IDF 두 구성요소의 곱으로 이루어지며, 문서(doc.)와 개수(num.) 표기를 간결하게 사용하였다.

i. Term Frequency (TF): 문서 d 별로 단어 w 가 얼마나 자주 등장하는지를 수치화

$$TF(w, d) = \frac{\text{Num. of word } w \text{ appears in doc. } d}{\text{Total num. of words in doc. } d}$$

ii. Inverse Document Frequency (IDF): 전체 문서 중 단어 w 가 등장한 빈도의 역수로서, 적은 문서에 등장한 단어를 강조

$$IDF(w) = \log\left(\frac{\text{Total num. of doc.}}{\text{Num. of doc. containing word } w+1}\right)$$

TF-IDF는 정보 검색, 텍스트 분류, 군집화, 키워드 추출 등에서 널리 사용되며, 문서 수가 많을 때는 log-scaling을 통해 값의 스케일을 조절하고 분모에 1을 더해 0으로 나누는 문제를 방지한다.

Log-TF-IDF는 각 로그 이벤트를 “term”, 날짜별로 수집된 로그를 “document”로 간주한다. 특정 문서 내 중요도가 아니라 정상 로그 전체 데이터에서 해당 이벤트의 희귀도를 측정하는 방식이다. 이는 TF-IDF의 IDF가 “전체 코퍼스에서 반복적으로 등장하는 단어에 패널티를 부여”하는 방식과 동일한 개념적 기반을 갖는다. 예를 들어, 특정 로그는 하루에 특정 시점마다 꾸준히 발생할 수 있다. 이는 전체 출현 빈도는 낮을 수 있으나 지속적으로 나타나는 정상 패턴이므로 이상 탐지와 무관하다. IDF는 해당 이벤트에 낮은 비중을 부여하여 오류 패턴과의 혼동을 방지한다.

본 연구에서는 실험 환경에서 LLM이 생성한 워크플로우를 지속적으로 실행하여 로그를 수집하고 정상 로그와 오류 로그를 수동으로 레이블링하여 정상 로그 베이스라인을 구성하였다. 이 데이터는 Log-TF-IDF 계산의 기준으로 활용되었다. 정상 데이터셋 (Normal Dataset, ND로 표기함)에 대해 Log-TF-IDF는 다음과 같이 정의된다. 1)

i. Log-TF: 전체 발생 빈도가 낮은 이벤트에 대해 더 높은 Log-TF 값을 반환. ND의 전체 로그 수가 매우 클 경우가 많기 때문에 log-scaling을 적용하여 규모를 맞춘다.

$$\text{Log-TF}(t) = \log\left(\frac{\text{Total num. of Logs in ND}}{\text{Num. of event } t \text{ appears in ND}}\right)$$

ii. Inverse Document Frequency (IDF): 전체 데이터에서 등장 횟수가 적을지라도 꾸준히 발생하는 이벤트에 대해 낮은 값 반환. 시스템 로그 데이터셋은 상대적으로 데이터 수가 적기 때문에 IDF에는 log-scaling을 취하지 않는다. 1보다 큰 값이 되도록 하기 위해 분자에 2를, 분모에 1을 더한다.

$$\text{Log-IDF}(t) = \frac{\text{Total num. of distinct datas in ND} + 2}{\text{Num. of datas that event } t \text{ appears} + 1}$$

이를 통해 계산되는 Log-TF-IDF는 다음과 같다.

$$\text{Log-TF-IDF}(t) = \log(\text{Log-TF}(t) \times \text{Log-IDF}(t))$$

이 수식은 이벤트의 총 발생 횟수가 낮거나 날짜별

발생 패턴이 불규칙한 경우 높은 점수를 부여하여, 희귀 이벤트를 효과적으로 식별할 수 있게 한다.

또한 로그 이벤트 추출을 위해 이전 연구[14]에서 개발한 로그 파서 및 이벤트 분류기를 사용하였다. 로그 파서는 입력 로그 내 중요하지 않은 비핵심 정보를 제거하고, 의미적으로 유사한 로그를 패턴 단위로 묶을 수 있도록 설계되었다. 파서는 사전에 구축된 단어 사전에 포함되지 않는 숫자, 파일 경로 등은 [num], [loc] 등 일반화된 토큰으로 대체한다. 이는 로그가 템플릿 기반으로 생성된다는 특성을 활용하여 대표 패턴을 추출하기 위함이다. 이후 파서는 향상된 최소 편집 거리(Minimum Edit Distance) 알고리즘을 기반으로 한 패턴 분류기를 통해 유사 패턴을 하나의 이벤트로 묶는다.

정상 로그 데이터셋을 기반으로 모든 이벤트에 대해 Log-TF-IDF 값을 계산한 후, 정상 로그에서 관측된 최대 Log-TF-IDF 값을 기준으로 입력 필터링의 임계값을 설정하였다. RAG 모듈은 입력된 로그를 동일한 방식으로 파싱하여 Log-TF-IDF 값을 계산한 뒤, 임계값을 초과한 경우 오류 로그로 판단한다. 이후 필터링된 오류 로그만 RAG 모듈의 입력으로 전달된다.

4) Output 필터

2절에서 언급한 두 번째 오류를 해결하기 위해 검색된 문서 중 오류 맥락과 관련성이 낮은 문서를 제거하는 기능이다. RAG 과정의 출력 단계에서도 필터링이 이루어진다. 검색된 문서의 관련성을 평가하기 위해 본 연구는 AI Agent의 주 LLM과 별개로 동작하는 Judge LLM을 도입하였다. Judge LLM은 사전에 수집한 로그 데이터와 해당 오류를 설명하는 문서 쌍 데이터 셋을 이용해 파인튜닝함으로써, Kubernetes 기반 환경에서 로그와 문서 간 문맥적 연결성을 학습하도록 구성하였다.

이를 통해 Judge LLM은 RAG가 반환한 문서가 입력 오류 로그와 실제로 관련이 있는지를 판단할 수 있다. 또한 문서가 유의미하다고 판단될 경우, 오류 해결에 필요한 핵심 정보만 추출하도록 설계하였다. 반면, 관련성이 낮다고 판단되면 해당 문서를 Main LLM에 전달하지 않는다. 이러한 선택적 필터링 메커니즘은 Main LLM이 비관련 정보로 인해 혼동되거나 잘못된 수정 결과를 생성하지 않도록 하는 데 중요한 역할을 한다.

다음 수도코드는 본 연구에서 제안하는 RAG 알고리즘의 전체 과정을 나타낸다.

1) 수식에서는 로그 메시지와 logarithm을 구별하기 위해 로그 메시지를 뜻하는 로그는 첫글자를 대문자로 표기함.

```

1: ErrorList  $\leftarrow$  []
2: for Log  $\in$  InputLogs do
3:   LogEvent, IsNew  $\leftarrow$  LogParser(Log)
4:   if IsNew then
5:     LogTFIDF  $\leftarrow$  CalculateTFIDF(LogEvent)
6:   else
7:     LogTFIDF  $\leftarrow$  TFIDF_Dict[LogEvent]
8:   end if
9:   if LogTFIDF  $\geq$  TFIDF_Threshold then
10:    Append [Log] to ErrorList
11:   end if
12: end for
13: if ErrorList is Empty then
14:   ErrorList  $\leftarrow$  InputLogs
15: end if
16: InputVector  $\leftarrow$  EmbeddingModel(ErrorList)
17: MaxSim  $\leftarrow$  0
18: for (Vector, Doc)  $\in$  VectorKB do
19:   sim  $\leftarrow$  VectorSim(Vector, InputVector)
20:   if sim  $>$  MaxSim then
21:     MaxSim  $\leftarrow$  sim
22:     QueriedDoc  $\leftarrow$  Doc
23:   end if
24: end for
25: Related, FilteredOutput  $\leftarrow$  JudgeLLM(QueriedDoc)
26: if Related and MaxSim  $>$  Sim_Threshold then
27:   return FilteredOutput
28: end if
29: return False

```

IV. 결과 및 검증

4.1. 실험 환경

본 연구에서 제안한 LLM 기반 AI Agent 및 RAG 시스템을 실험하기 위해 LLM 및 RAG를 동작시킬 GPU 서버와 Kubernetes NDT 환경을 동작시키는 서버를 별개로 구동하였다. GPU 서버는 NVIDIA RTX A6000 GPU와 Intel Xeon Silver 4216 CPU를 포함한 서버이다. NDT는 Kubernetes 1.29.15 버전을 활용하였으며, Ubuntu 22.04 LTS를 설치한 두 개의 서버를 활용하여 각각 Master 노드와 Worker 노드로 활용하였다. 테스트 모듈이 실행되는 서버는 kubectl을 통해 Kubernetes 클러스터에 접근 권한을 부여받았으며, 이를 통해 Python 및 Ansible 기반 테스트 절차 모두에서 클러스터와 직접 상호작용할 수 있도록 구성하였다. 해당 구조는 Kubernetes 환경 내에서 Pod 실행 및 CNF 설정 작업을 직접 수행할 수 있게 한다.

RAG를 구현하기 위해 먼저 포괄적인 지식베이스를 구축하였다. 구체적으로, Kubernetes 공식 문서에서 538개, Ansible 문서에서 2,676개, Stack Overflow로부터 1,552개의 질문 스레드를 크롤링하

여 총 4,766개의 문서로 구성된 코퍼스를 구축하였다. 공식 문서의 경우 하나의 문서에 여러 기능 설명이 포함된 경우 이를 세분화하여 별도의 문서 항목으로 분리함으로써 보다 정밀한 검색이 가능하도록 하였다. Stack Overflow는 “Kubernetes”, “Ansible”, “nDPI”, “Firewall” 등의 키워드를 기반으로 관련성이 높은 질문을 선별하였다. 또한 실험 과정에서 오류 메시지에 대해 유용한 문서가 추가로 식별되는 경우, 해당 문서를 지식베이스에 수동으로 보강하였다.

지식베이스 구축 후에는 벡터 기반 검색 시스템을 개발하였다. 각 문서는 제목, URL, 본문 내용, 그리고 질문 본문(존재하는 경우)을 포함한 필드와 함께 SQLite 데이터베이스에 저장하였다. 벡터 기반 검색을 지원하기 위해 RAG 구현에서 널리 활용되는 문장 임베딩 모델인 all-MiniLM-L6-v2 [16]를 사용하였다. 또한 동일한 모델을 기반으로 본 연구에서 수집한 로그 데이터로 파인튜닝한 모델을 추가로 구성하여 비교 실험을 수행하였다. 각 문서는 제목과 질문 본문(StackOverflow 한정)을 입력으로 하여 벡터 표현을 생성하였고, 생성된 벡터는 Chroma 벡터 데이터베이스에 저장하였다.

RAG 시스템의 성능을 평가하기 위하여 실제 시나리오에서 AI Agent를 실행하고, 정상적으로 설치된 경우와 오류가 발생한 경우 모두에서 로그를 수집하였다. 정상 로그 데이터셋을 바탕으로 Log-TF-IDF 값을 계산하여 로그 시퀀스 내 이벤트의 중요도를 정량화하였다. 이를 활용해 패턴 사전(pattern dictionary)과 이벤트 사전(event dictionary)을 구축하였으며, 각 이벤트에 대해 Log-TF-IDF 값을 산출하였다. 분석 결과 평균 Log-TF-IDF 값은 3.54였으며, 최대값은 5.46으로 나타났다. 이를 기반으로 본 연구에서는 RAG 파이프라인의 입력 로그 필터링 기준을 5.46으로 설정하여, 기준을 초과하는 로그를 오류로 판단하도록 하였다.

오류 상황에서 수집한 129가지의 다른 로그에 대해 각각 수동으로 로그와 관련된 문서를 태깅하여 RAG의 검증에 사용하였다. 구축된 로그 데이터셋을 바탕으로 Judge LLM을 학습시켰다. 문서 제목과 로그 시퀀스를 함께 Judge LLM에 제공하여 지도학습 기반 파인튜닝을 수행하였다. 구체적으로, PEFT 라이브러리와 LoRA 기법을 활용하여 Phi-4 모델 [17]을 파인튜닝함으로써 오류 패턴과 관련 문서 간의 매핑을 학습할 수 있도록 하였다.

4.2. 실험 결과

1) RAG 자체 성능 검증

RAG의 Retriever 성능, 즉 Judge LLM 이전까지의 과정을 독립적으로 평가하기 위한 실험을 수행하였다. 실험은 직접 태깅한 129개의 로그-정답 문서 데이터셋을 활용하였으며, 각 로그에 대해 RAG 시스템이 해당하는 정답 문서를 정확히 검색해낼 수 있는지를 확인하였다. 검색된 문서의 제목과 정답 문서의 제목 사이의 코사인 유사도가 0.8을 초과할 경우 성공으로 판단하여 유사한 문서가 복수 존재하는 경우에는 이를 모두 정답으로 인정하였다.

성능 비교를 위해 기본 임베딩 모델(baseline embedding model)과 본 연구에서 구축한 로그 데이터셋으로 파인튜닝한 임베딩 모델을 모두 평가하였다. 또한 제안한 Log-TF-IDF 기반 입력 필터링 기법의 효과를 검증하기 위해 ‘error’ 또는 ‘Err’ 키워드를 포함한 로그를 그대로 RAG 입력으로 사용하는 규칙 기반(rule-based) 필터링 방식과 비교하였다.

그림 4는 실험 결과를 나타낸다. 실험 결과, 파인튜닝된 임베딩 모델에 Log-TF-IDF 입력 필터링을 결합한 방식이 가장 높은 정확도로 정답 문서를 검색함을 확인하였다. 특히 Log-TF-IDF 기반 필터링은 규칙 기반 필터링 접근보다 관련성 높은 입력만 효과적으로 추출하는 데 있어 월등한 성능을 보였다. 이를 통해 입력 필터링을 통해서 에러와 연관된 로그를 RAG의 입력으로 사용하는 것이 유의미한 효과를 가지는 것을 입증하였다.

그러나 가장 우수한 설정에서도 검색 정확도는 90%를 넘지 못하였다. 추가 분석 결과, 과도하게

긴 입력 로그가 주요 원인임을 확인하였다. 입력 필터링은 로그 한 줄을 하나의 이벤트로 분류하며, 여러 줄의 로그 중 에러로 의심되는 이벤트를 필터링하는데, 한 줄의 로그가 엄청나게 긴 경우가 원인이었다. 해당 문제는 특히 Ansible이 생성한 로그에서 특히 두드러진다. 해당 로그는 실행 명령, 노드 식별자, 컨테이너 정보, 이미지 참조 등 과도한 메타 데이터를 포함하는 경향이 있다. 해당 경우 로그 파서를 적용하더라도 한 줄이 평균보다 크게 길어졌으며, 대부분의 단일 로그가 20단어 미만인 반면 Ansible 로그는 한 줄이 70단어를 초과하는 경우가 일반적이었다. 이와 같은 지나치게 장황한 내용은 출력 패턴을 왜곡하며 벡터 유사도 기반 검색 성능을 저해하는 것으로 나타났으며, 추후 복잡한 로그에 대한 전처리 과정의 개발이 필요하다.

2) RAG 적용 결과 검증

이어서 제안한 RAG 모델을 통해 AI Agent가 자가 수정이 가능한지 확인해보았다. 실험을 위해 GPT 모델[19]을 통해 5가지 NF의 설치에 관한 60개의 MOP를 AI Agent의 입력으로 사용하여 그 중 몇 %의 MOP에 대해 3번의 워크플로우 생성 요청 안에 테스트 모듈을 통과하는가에 대해 정확도를 측정하였다. RAG 없이 LLM만 사용하였을 때 가장 성능이 높은 세 가지 LLM 모델인 GPT-4o [18], Gemma3:27b [19], Qwen2.5-coder [20] 세 모델에 대해 RAG 적용 전/후의 성능을 비교하였다. 실험에서는 3-shot 러닝으로 다른 NF의 동작 코드 3개를 프롬프트에 함께 제공해주었으며, LLM은 파인튜닝 없이 오픈 소스로 공개된 모델을 그대로 사용하였다.

성능 비교 결과는 표1과 같으며, 전반적으로 RAG를 적용하였을 때, 그리고 RAG와 Output 필터를 적용하였을 때 점차 성능이 향상하는 것을 알

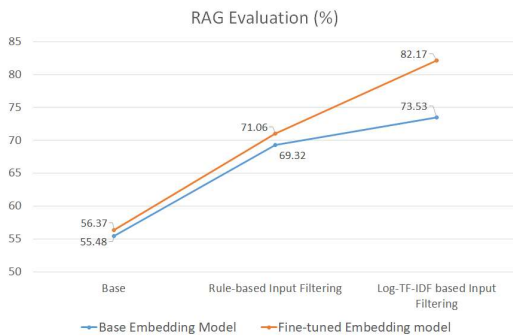


그림 4. RAG 검증 실험 결과
Fig. 4. RAG Evaluation Test Results

표 1. RAG 검증 실험 결과
Table 1. RAG Evaluation Test Results

모델	대상	LLM (%)	LLM+ RAG, Output filter X (%)	LLM+ RAG, Output filter (%)	향상도 (Δ)
GPT-4o	Python	44.3	73.3	83.3	39
GPT-4o	Ansible	51.7	68.3	63.3	11.6
Gemma3	Python	50	70	76.7	26.7
Gemma3	Ansible	56.7	65	68.3	11.6
Qwen2.5	Python	46.6	65	78.3	31.7
Qwen2.5	Ansible	53.3	55	66.7	13.4

수 있다. 특히, Judge LLM을 129개의 소규모 데이터셋을 학습시킬 때도 유의미한 효과를 가짐을 확인할 수 있다. 이를 통해 Input 필터 뿐 아니라 Output 필터도 RAG의 성능에 기여하며, 이를 통해 LLM이 자가 수정을 진행할 수 있음을 확인할 수 있다.

성능향상률이 가장 높을 때는 39%, 평균적으로는 22.3%로 정확도가 향상되는 것을 확인할 수 있다. 전반적으로 Python 환경에서 더 성능 향상이 높는데, 이는 RAG 실험 과정에서 Ansible 에러일 때 관련 도큐먼트를 잘 못 찾던 것에 기인한 것으로 해석된다. 향후 연구로서 Ansible의 복잡한 에러 로그에 대해 전처리 과정을 통해 RAG의 성능 향상이 필요하다.

V. 결 론

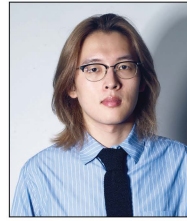
본 논문에서는 LLM 기반 NFV 구성 자동화 AI Agent를 제안하였으며, AI Agent가 자가 수정 기능을 갖도록 하기 위해 NFV 구성 에러 발생시 관련 문서를 제공하는 RAG 기능을 설계하고 실험하였다. 제안한 AI Agent는 Kubernetes NDT 환경과 통신하며 LLM의 응답을 검증하며, LLM의 응답에서 에러가 발생한 경우 제안한 입력 필터링을 통해 에러로 의심되는 로그를 RAG에 전달하고, 출력 필터링을 통해 현재 상황에 적합한 도큐먼트를 LLM에게 전달한다. 실험 결과 제안한 RAG가 LLM의 자가 수정에 유의미한 효과가 있음을 입증하였으며, 비교적 성능이 낮은 Ansible에 적합한 전처리 과정들을 포함하기 위한 추가 연구가 필요하다.

References

- [1] Alexander Clemm, Laurent Ciavaglia, Lisandro Zambenedetti Granville, and Jeff Tantsura. Intent-Based Networking Concepts and Definitions. RFC 9315, October 2022.
- [2] Denis Donadel, Francesco Marchiori, Luca Pajola, and Mauro Conti. Can llms understand computer networks? towards a virtual system administrator. In 2024 IEEE 49th Conference on Local Computer Networks (LCN), pages 1 - 10. IEEE, 2024.
- [3] Jieyu Lin, Kristina Dzevaroska, Ali Tizghadam, and Alberto Leon-Garcia. Appleseed: Intent-based multi-domain infrastructure management via few-shot learning. In 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), pages 539 - 544. IEEE, 2023.
- [4] Eui-Dong Jeong, Hee-Gon Kim, Sukhyun Nam, Jae-Hyoung Yoo, and James Won-Ki Hong. S-witch: Switch configuration assistant with llm and prompt engineering. In NOMS 2024-2024 IEEE Network Operations and Management Symposium, pages 1 - 7. IEEE, 2024.
- [5] Kristina Dzevaroska, Jieyu Lin, Ali Tizghadam, and Alberto Leon-Garcia. Llm based policy generation for intent-based management of applications. In 2023 19th International Conference on Network and Service Management (CNSM), pages 1 - 7. IEEE, 2023.
- [6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In Advances in Neural Information Processing Systems, volume 33, pages 9459 - 9474. Curran Associates, Inc., 2020.
- [7] Nguyen Van Tu, Jae-Hyoung Yoo, and James Won-Ki Hong. Towards intentbased configuration for network function virtualization using in-context learning in large language models. In NOMS 2024-2024 IEEE Network Operations and Management Symposium, pages 1 - 8. IEEE, 2024.
- [8] Linux Foundation and Google. Nephio: Automating cloud-native network functions with kubernetes. <https://nephio.org/>, 2022. White Paper.
- [9] The Kubernetes Authors. Kubernetes. <https://github.com/kubernetes/kubernetes>, 2014. Accessed: 2025-11-25
- [10] Sukhyun Nam, Nguyen Van Tu, James Won-Ki Hong, Llm-based vnf deployment automation in openstack environment, 2025 IEEE/IFIP Network Operations and Management

- Symposium (NOMS 2025), Honolulu, HI, USA, 12-16 May 2025
- [11] Sukhyun Nam. Llm based vnf github. <https://github.com/obiwan96/LLMbasedVNF>, 2024. Online; accessed 2025-11-25
- [12] Stack exchange api. <https://api.stackexchange.com/>. Accessed: 2025-11-25.
- [13] Sukhyun Nam, Jae-Hyoung Yoo, and James Won-Ki Hong. Log-tf-idf for anomaly detection in network switches. In NOMS 2024-2024 IEEE Network Operations and Management Symposium, pages 1 - 9. IEEE, 2024.
- [14] Sukhyun Nam, Eui-Dong Jeong, and James Won-Ki Hong. Log-tf-idf and netconf-based network switch anomaly detection. International Journal of Network Management, 35(1):e2322, 2025.
- [15] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. Journal of documentation, 28(1):11 - 21, 1972.
- [16] Nils Reimers and Iryna Gurevych. sentence-transformers/all-minilm16-v2. <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>, 2021. Accessed: 2025-11-25
- [17] Mojan Javaheripi, Sebastien Bubeck, Marah Abdin, Jyoti Aneja, Caio Cesar T. Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, Suriya Gunasekar, Yin Tat Lee, Yuanzhi Li, Anh Nguyen, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Adam Taumann Kalai, and many others. Phi-2: The surprising power of small language models. Microsoft Research Blog, 2023. Accessed: 2025-11-25
- [18] OpenAI. OpenAI GPT models. <https://platform.openai.com/docs/guides/gpt>. Online; accessed 2025-11-25
- [19] Google. Gemma: Open models based on gemini research and technology, 2024.
- [20] Alibaba Group. Qwen-2: A scalable foundation model from alibaba. <https://modelscope.cn/models/qwen/Qwen-2-7B>, 2024. Online; accessed 2025-11-25

남 석 현 (Suk-hyun Nam)



2025 ~ 현재 포항공과대학교, 인공지능연구원 박사후 연구원
2025 포항공과대학교, 컴퓨터공학 박사
2021 포항공과대학교, 컴퓨터공학 석사

2019 KAIST, 전산학부 학사

<관심분야> SDN, 네트워크 관리, LLM, 5G/6G

홍 원 기 (James Won-Ki Hong)



2020~현재 포스텍 크립토펙트체인연구센터 공동센터장
2012~2014 KT 종합기술원, 원장
2008~2012 포항공과대학교, 정보전자융합공학부장
2008~2010 포항공과대학교, 컴퓨터공학과 주임교수

2007~2010 포항공과대학교, 정보통신연구소 연구소장

2007~2011 포항공과대학교, 정보통신대학원장

1995~현재 포항공과대학교, 컴퓨터공학과 교수

1992~1995 Univ. of Western Ontario, 연구교수

<관심분야> 네트워크 및 시스템 관리, Gen AI, 블록체인 및 암호화폐