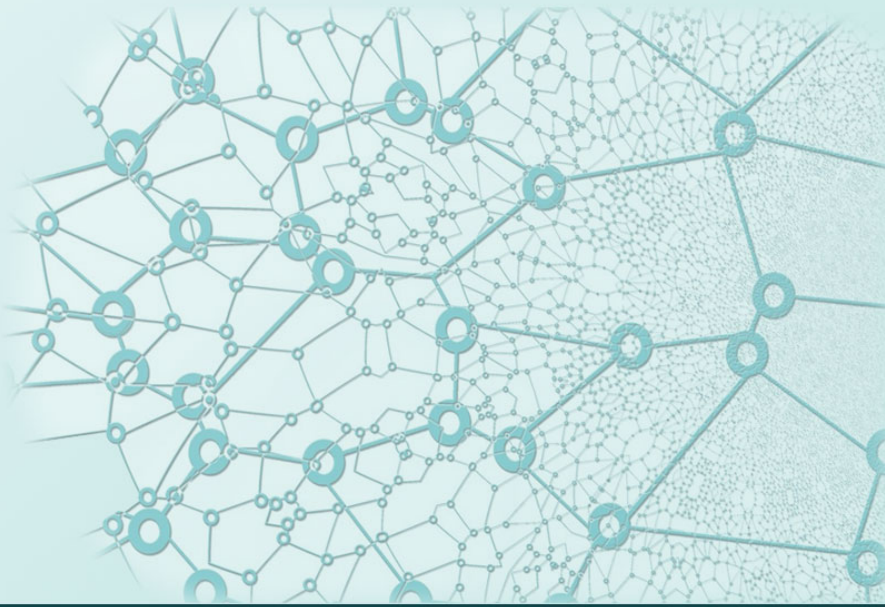
KNOM Review | 29권 1호, 2026년 8월 | ISSN 2287-1543 (Online)

KNOM REVIEW

Network Operations & Management

Vol. 29 No.1 | AUG. 2026



한국통신학회 통신망운용관리연구회

The Committee on Korean Network Operations and Management (KNOM)
Korean Istitute of Communication Sciences (KICS)
http://www.knom.or.kr

목 차

제29권 제1호, 2026년 8월

LLM 기반 초기 해집단 생성을 활용한 서비스 배치 및 트래픽 라우팅 최적화
..... 이도영 1

적응형 세그먼트 길이 기반의 DASH 볼류메트릭 영상 스트리밍
..... 유상우, 홍원기 13

자율 Kubernetes 관리를 위한 에이전트형 AI: LLM 기반 의사결정 사이클을 활용한 멀티 에이전트 아키텍처
..... Kiran Batool, 송왕철 24

PRISM-2026:엔드포인트 기반 자동 라벨링 다계층 네트워크 트래픽 데이터셋
..... 김지민,유경민,장윤성,남승우,김주성,김명섭 34

사물인터넷 환경에서 효과적인 분할 연합 학습을 위한 통신 오버헤드 최적화 연구 동향
..... 정한길, 김종욱, 최미정 45

LLM 기반 네트워크 기능 설치 명령어 생성 및 자가 수정을 위한 RAG 기법 연구
..... 남석현, 홍원기 59

한국통신학회 통신망운용관리연구회

KNOM Review '26-01 Vol.29 No.01
https://doi.org/10.22670/knom.2026.29.1.1

LLM 기반 초기 해집단 생성을 활용한 서비스 배치 및 트래픽 라우팅 최적화

이 도 영◆°

Service Placement and Traffic Routing Optimization Using LLM-Generated Initial Population

Doyoung Lee◆°

요 약

6G 네트워크 시대를 준비하기 위해서는 5G에서 정의한 성능 기준치의 상향 뿐만 아니라, 새롭게 등장할 서비스나 시나리오 요구사항에 맞춘 네트워크 측면의 발전이 요구되고 있다. 특히, 차세대 네트워크에서는 엄격한 서비스 요구사항과 더불어 서비스를 제공함에 있어서 고려되어야 하는 추가적인 제약 사항들로 인해 기존 네트워크에서 운용되는 서비스 배치 및 트래픽 라우팅 기법의 개선을 촉진하고 있다. 따라서, 동적인 6G 네트워크 최적화를 위한 연구들이 진행되고 있으며, 그 중 메타휴리스틱 기반 기법은 최적 근사해를 도출하는데 효과적인 방법으로 주목받고 있다. 하지만, 메타휴리스틱 기반 최적화는 초기 해의 구성에 따라 최종 해의 적합도 또는 수렴 시간에 큰 영향을 준다는 한계가 존재한다. 본 논문에서는 이를 극복하기 위하여, 기존 초기 해 임의 생성 방식 대신 LLM을 활용하여 네트워크 상황 및 서비스 요구사항에 맞는 초기 해집단을 생성하는 방식을 제안한다.

Key Words : 6G, Routing optimization, Large Language Models, Metaheuristic algorithm

ABSTRACT

To prepare for the upcoming 6G era beyond 5G, network technologies must evolve not only to achieve enhanced performance targets but also to support emerging services and communication scenarios. In particular, next-generation networks demand improvements in existing traffic routing and service placement techniques under increasingly stringent service requirements and additional operational constraints. Accordingly, metaheuristic-based optimization techniques have attracted attention as effective approaches for deriving near-optimal solutions in dynamic 6G environments. However, their performance and convergence speed are highly dependent on the quality of the initial solution population. To address this limitation, this paper proposes an LLM-based approach that generates high-quality initial solutions according to network conditions and service requirements, replacing conventional random initialization methods.

◆° First Author and Corresponding Author : Kongju National University, dylee@kongju.ac.kr

논문번호 : KNOM2026-01-002, Received July 22, 2026; Revised August 9, 2026; Accepted August 18, 2026

Ⅰ. 서 론

정보통신기술의 발전은 5G 네트워크 시대를 성공적으로 실현할 수 있었던 원동력이 되어왔으며, 이제는 다가오는 6G 네트워크 시대의 기술 구현을 목표로 하고 있다. 이와 함께, 6G 네트워크에서는 5G에서 정의된 용례들(Use-cases)보다 엄격한 성능 요구사항을 가지는 다양한 서비스 시나리오를 제안하고 있다 [1]. 따라서, 6G의 기술적 요구사항을 충족하는 네트워크 구현을 위해서는 지능적이고 유연한 네트워크 관리 및 운용 기술이 필수적이다. 또한, 5G를 넘어서 6G로 향하는 시대적 흐름에서는 인공지능(AI, Artificial Intelligence) 기술을 부가적으로 활용하는 것이 아니라, 네트워크 전 영역에 탑재된 형태로 적극적으로 활용하는 AI 내재형 네트워크(AI-native networks)의 실현이 필요하다 [2].

AI 내재형 네트워크에서는 6G에서 언급되는 초대규모 IoT, 초저지연 서비스, 위성-지상 통합 네트워크 등을 지원하기 위한 최적화 기법의 필요성이 강조되고 있다 [3, 4]. 특히, AI 모델의 학습 및 추론 성능 개선, 지속적인 네트워크 상태 관측, 데이터 수집 및 전처리 등의 작업이 네트워크 운용과 함께 병행되면서 네트워크 운용 및 관리를 위한 비용이 급증할 가능성이 존재한다. 이와 함께, 최근 환경 문제에 대한 관심은 5G와 6G에서는 네트워크 운용 및 서비스 제공 과정에서 발생하는 탄소 배출량을 감축하는 것도 주요 목표로 다루어지도록 하고 있다 [5, 6].

상기 서술한 기술적인 필요성과 서비스 요구사항들은 기존 네트워크 운용 및 관리 기술의 개선을 촉진시키는 원인이 되었다. 예시로, 엄격한 서비스 성능 요구사항, 즉, QoS (Quality-of-Service)를 충족함과 동시에 운용 비용과 탄소 배출량을 동시에 최소화하는 것은 서비스 배치 문제, 트래픽 라우팅 문제 등을 NP-Hard 또는 조합 최적화 문제로 진화시켰다. 한편, AI 기술은 네트워크 내 존재하는 여러 최적화 문제를 위한 해결 방안으로 활용될 수 있지만 [7], 다목적 최적화 문제들을 해결하는 과정에서 최적해를 보장할 수가 없다는 한계가 존재한다 [8]. 따라서 이러한 문제를 효과적으로 해결하기 위해 네트워크 최적화를 위한 연구가 수행되고 있으며, 그 중에서 메타휴리스틱(Metaheuristic) 기반 방식은 허용 가능한 시간 내 최적 근사해를 도출할 수 있는 특성으로 인해 주목받고 있다.

일반적으로 메타휴리스틱 기반 최적화 기법에서는 우선 초기 해를 생성한 후, 각 해에 대한 평가와 새로운 해 탐색, 해 개선 등을 종료 조건에 도달할 때까지 반복하며 최적 근사해를 도출한다 [9]. 즉, 초기 해집단을 구성하는 단계부터 시작하기 때문에 초기 해들의 분포에 따라 최종적으로 산출되는 해의 적합도(Fitness)나 수렴 시간에 영향을 받을 수 있다. 따라서, 해 공간을 효율적으로 탐색함과 동시에 한정된 시간 내 가장 높은 적합도를 가진 해를 도출하기 위해서는 다양성과 품질을 고려한 초기 해들을 생성하는 것이 중요하다. 하지만, 대부분의 메타휴리스틱 기반 최적화 기법에서는 임의로 초기 해집단을 구성하는 방법을 사용하고 있다는 한계가 존재한다.

본 논문에서는 이와 같은 문제를 해결하기 위해 대형 언어 모델(LLM, Large Language Model)을 초기 해 생성에 사용하는 메타휴리스틱 기반 최적화 기법을 제안한다. 메타휴리스틱 알고리즘으로는 유전 알고리즘을 활용하며, 이는 동적인 네트워크 환경에서 운용 비용을 최소화하고 서비스 요구사항을 만족하는 다목적 최적화 문제인 네트워크 서비스 배치 및 트래픽 라우팅에 적용된다. 또한, 일반적으로 특정 용도를 위한 LLM을 확보하기 위해 수행되는 미세조정(Fine-tuning)은 큰 연산 비용이 요구되기 때문에, 본 논문에서 제안하는 방법은 초기 해 생성을 위해 맥락 내 학습(ICL, In-Context Learning) 기법을 활용한다.

본 논문의 나머지 내용은 다음과 같이 구성된다. 2장에서는 네트워크 서비스 배치와 트래픽 라우팅 관련된 연구 및 기술 동향에 대해 서술한다. 3장에서는 서비스 배치 및 트래픽 라우팅을 다루기 위한 시스템 모델을 정의하고, 이어지는 4장에서는 제안하는 LLM 기반 초기 해 생성 방법을 적용한 유전 알고리즘 기반 서비스 배치 및 트래픽 라우팅 방법에 대해 서술한다. 5장에서는 구현된 최적화 기능에 대한 성능 평가를 수행하며, 마지막 6장에서는 논문 내용을 정리하고 향후 연구 방향에 대해 논의한다.

Ⅱ. 연구 배경 및 관련 연구

네트워크 운용 및 관리 기술 중, 서비스 배치는 사용자에게 제공되어야 하는 네트워크 서비스를 클라우드, 엣지(Edge) 등 물리적으로 어떤 위치에 배포되어야 하는지 결정하는 최적화 기술이다. 서비스 배치에서는 사용자에게 충족되어야 하는 서비스별

QoS 및 배치, 운용 비용 등을 고려하여 효율적인 배치가 될 수 있도록 하는 것이 중요하다.

네트워크 서비스 배치 문제는 대표적인 NP-Hard 문제이며, 이를 해결하기 위한 다양한 연구가 수행되어왔다. 선행연구 [10]에서는 서비스 지연시간을 최소화하기 위해 사용자와 서비스 노드 간 홉(Hop) 수를 최소화하는 정수 선형 계획법(ILP, Integer Linear Programming) 모델을 정의하고, 이를 해결하기 위한 네트워크 상태 예측 기반 서비스 배치 방법을 제안하였다. 또한, 선행연구 [11]에서는 서비스 배치와 사용자의 서비스 요청이 어떤 엣지에서 처리되어야 하는지 스케줄링하는 문제를 탐욕적 알고리즘 기반으로 해결하였다. 이 외에도 기계학습이나 강화학습 기법을 활용하여 동적인 네트워크 환경에서의 서비스 배치 문제를 최적화하려는 연구가 수행되었다 [12, 13].

한편, 서비스 트래픽 라우팅은 사용자가 특정 서비스를 제공받는 과정에서 발생하는 서비스 트래픽을 네트워크의 어떤 경로를 통해 전달되어야 하는지를 결정하는 최적화 기술이다. 서비스 배치와 유사하게, 트래픽 라우팅에서도 네트워크 운용 비용과 각 서비스별 QoS 보장을 위한 최단 경로 결정이 중요한 조건으로 작용한다.

일반적으로 트래픽 라우팅은 서비스 종류에 따라 트래픽을 최적 경로를 통해 목적지로 전달하는 기술로 인지된다 [14]. 즉, 운용 비용이나 홉 수를 최소화하는 최단 경로를 결정하는 알고리즘들이 주로 연구되었으나, 5G와 6G에서는 위성-지상 통합 네트워크로 범위가 확대되어 동적으로 변화하는 네트워크 토폴로지와 불균형한 트래픽 부하를 처리해야 하는 복잡한 문제로 진화하였다. 예시로, 선행연구 [15]에서는 강화학습을 통해 각 위성 노드를 독립적인 에이전트로 취급하고, 기계학습을 활용하여 트래픽 부하를 예측하는 방안을 제안하였다. 이 외에도 트래픽 라우팅에서 운용 비용과 함께 탄소 배출량을 절감하는 것이 중요시되며 탄소 배출량 절감을 목적으로 하는 라우팅 기법이 제안되었다 [16].

상기 서술한 연구들 외에도, 최근에는 생성형 인공지능 모델을 활용한 네트워크 최적화 연구가 주목을 받고 있다. 특히, LLM은 네트워크 운용 정책을 자연어로 표현된 의도(Intent)로 입력받아 추론 결과를 도출할 수 있는 특성으로 인해 다양한 분야에 적용되고 있다 [17]. 그 예시로, 선행연구 [18]에서는 LLM을 활용하여 VM (Virtual Machine) 생성 및 VNF (Virtual Network Function) 설정을 포함하는 VNF 배치 자동화 방안을 제안하였고, 선행연구 [19]에서는 사용자 의도를 소프트웨어 정의 네트워크(SDN, Software-Defined Network)의 제어평면(CP, Control Plane)에서 인식할 수 있는 라우팅 결정으로 변환하는 방안을 제시하였다. 이 외에도 LLM의 범용적인 추론 능력을 활용하여, LLM을 최적화 과정의 다양한 단계를 자동화할 수 있는 최적화 도구로써 사용되는 연구가 수행되고 있다 [20, 21, 22].

이처럼 5G와 6G 네트워크에서의 서비스 배치, 트래픽 라우팅과 같은 네트워크 최적화 문제는 엄격한 QoS 조건 달성과 함께 운용 비용, 탄소 배출량 절감 등을 최소화하는 다목적 최적화 문제로 진화하고 있다. 따라서, LLM 적용 등을 통해 동적으로 변화하는 네트워크 환경에서 효과적으로 서비스 배치 및 트래픽 라우팅 결정을 내리는 기술에 대한 수요가 증가하고 있으며, 서비스 배치 및 트래픽 라우팅을 동시에 고려하는 최적 의사 결정 방안이 필요할 것으로 예상된다.

Ⅲ. 시스템 디자인

3.1. 시스템 모델

네트워크 소프트웨어화(Network softwarization)로 인해 네트워크 운용 및 관리 기능과 서비스를 동적이고 유연하게 배포하는 것이 가능하게 되었다. 특히, SDN은 이를 실현한 대표적인 예로, 중앙 집중화된 제어 평면을 통해 네트워크 상태를 관측하고, 서비스 배치 또는 트래픽 라우팅 결정을 내린 후 이를 데이터 평면에 적용하는 구조를 가진다. 5G와 6G에서도 요구사항에 맞는 최적 의사 결정을 내리기 위해서는 해당 구조가 적용될 것으로 예상된다. 그림 1은 제어 평면을 통해 서비스 배치 및 트래픽 라우팅을 수행하는 시스템 모델의 개념을 보인다.

본 논문에서 다루는 서비스 배치 및 트래픽 라우팅 문제는 선행연구 [23]을 기반으로 정의된다. 구체적으로는 서비스 트래픽이 전달되는 네트워크 도메인(Domain)과 서비스가 배치되어 트래픽을 처리하는 연산 도메인이 포함된다. 따라서, 네트워크를 구성하는 노드(Node)들은 역할에 따라서 네트워크 노드 N^d와 연산 노드 N^C로 구분할 수 있다. 각 노드는 서로 다른 노드와 링크(Link)로 연결될 수 있다. 즉, 서비스 배치와 트래픽 라우팅 환경인 네

트워크는 노드들의 집합 N과 링크 집합 L로 표현되는 그래프 $G=(N, L)$로 정의된다.

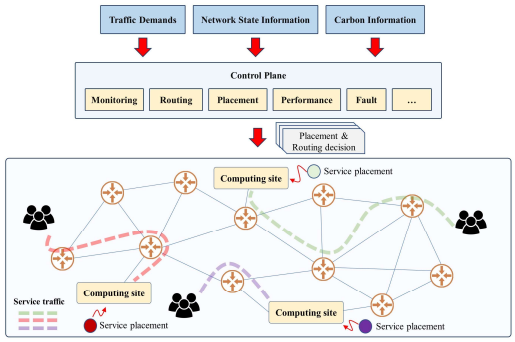


그림 1. 서비스 배치 및 트래픽 라우팅 개념
Fig. 1. Concept of service placement and traffic routing

주어진 네트워크에서 서비스 배치와 트래픽 라우팅은 각 서비스 트래픽의 요구사항 $r \in R$이 존재할 때, 사용자로부터 요청받은 서비스가 배치되어야 하는 연산 노드를 N^C 중에서 선택한 후 해당 연산 노드까지의 경로를 구성하는 네트워크 노드들을 N^d에서 선택하는 것이다. 최종적으로는 선택된 네트워크 노드들을 연결하는 링크들이 경로를 구성하는 요소들이 된다. 즉, 서비스 트래픽 요구에 명시되는 목적지는 고정된 단일 목적지 형태로 표현되는 것이 아니라, 해당 서비스가 배치될 수 있는 위치까지 고려한 여러 선택지가 존재한다. 따라서, 서비스 배치 및 트래픽 라우팅 문제는 서비스 배치 위치와 최적 경로를 결정하는 다목적 최적화 문제이다.

3.2 서비스 배치 및 트래픽 라우팅 모델

서비스 배치 및 트래픽 라우팅 문제는 서비스에 대한 트래픽 요구사항 $r \in R$이 생성되었을 때, 사용자에게 제공되어야 하는 서비스가 배치된 연산 노드와 해당 노드로 트래픽이 전달되는 경로에 포함되는 네트워크 노드들에서 발생하는 운용 비용을 최소화하는 문제로 정의할 수 있다. 본 논문에서는 서비스 배치 및 트래픽 라우팅 문제를 다루기 위해 선행 연구 [23]에서 정의된 수식 (1)의 최적화 문제를 활용한다. 해당 수식에서는 트래픽 요구사항 $r \in R$을 충족하는 서비스 배치 및 트래픽 라우팅에서 발생하는 운용 비용을 최소화하는 목적을 가진다. 이 때, 운용 비용은 서비스 배치 결정으로 인해 선택된 연산 노드 $n \in N^C$에서 발생하는 운용 비용 m_n과 해당 연산 노드까지 경로에 포함되는 링크들 $l \in L$에서의 운용 비용 e_l을 합한 값으로 표현된다.

$$\min \sum_{r \in R}\left(\sum_{l \in L} x_{rl} e_l + \sum_{n \in N^C} y_{rn} m_n\right) \tag{1}$$

$$\text{s.t.,} \quad \sum_{r \in R} x_{rl} t_r \le b_l, \ \forall l \in L \tag{2}$$

$$\sum_{l \in L} x_{rl} d_l \le g_r, \ \forall r \in R \tag{3}$$

$$\sum_{l \in L} x_{rl} \log a_l \ge \log k_r \ \forall r \in R \tag{4}$$

$$\sum_{l \in L} x_{rl} \le h_r \ \forall r \in R \tag{5}$$

즉, 네트워크 내에서 해당 트래픽 요구사항을 처리하기 위해 선택된 연산 노드와 링크들을 식별할 방법이 필요하며, 이를 위해 수식 (6)과 같이 각각 이진 변수 x_{rl}와 y_{rn}을 사용한다.

$$x_{rl}, y_{rn} \in \{0, 1\}, \ \forall r \in R, \ l \in L, \ n \in N^C \tag{6}$$

해당 수식에서 x_{rl}은 트래픽 요구사항 r을 목적지로 전달할 때 링크 l이 경로에 포함될 경우 1의 값을 가지며, y_{rn}은 연산 노드 n이 트래픽 요구사항 r을 처리할 목적지로 선택되는 경우 1의 값을 갖는다. 한편, 해당 트래픽 요구사항에 관련된 트래픽 경로에 포함되지 않은 링크이거나, 서비스 배치에 선택된 연산 노드가 아닌 경우는 0의 값을 갖는다.

그 외 수식 (2)~(5)는 목적 함수인 수식 (1)과 관련된 제약 조건을 나타낸 것이다. 수식 (2)는 네트워크 전송 용량의 한도를 나타내는 제약 사항으로서, 각 트래픽 요구사항 r이 필요로 하는 전송 용량 t_r의 합이 링크 l에서 보장하는 최대 전송 용량인 b_l 이내여야 하는 것을 명시한다. 또한, 수식 (3)은 각 r이 허용하는 최대 지연 시간인 g_r가 r을 위해 결정된 라우팅 경로에 포함되는 각 링크 l의 지연 시간 d_l의 합보다 같거나 작아야 하는 조건을 의미한다. 다음으로 수식 (4)와 (5)는 각각 경로에 포함된 링크 l의 가용성(Availability) a_l이 트래픽이 요구하는 최소 가용성 k_r 이상이어야 하고, 각 경로는 최대 홉수 h_r을 넘을 수 없는 것을 제약하는 조건들이다.

한편, 각 트래픽 요구사항 r을 처리하기 위한 서비스가 배치되는 연산 노드 n과 관련된 수식 (7)~(9)와 같은 제약 조건들도 존재한다. 수식 (7)은 특정 트래픽 요구사항 r은 단일 연산 노드에서 처리되어야 한다는 것을 나타내는 조건이다. 다음으로

수식 (8)은 두 종류의 노드 중, 네트워크 노드에서는 트래픽이 처리될 수 없음을 나타내며, 수식 (9)는 선택된 연산 노드 n에서 트래픽을 처리하는 과정에서 요구되는 자원 $j \in J$에 대한 수요량인 f_{rj}보다 해당 노드의 가용 자원량 c_{nj}가 크다는 조건을 명시한다.

$$\sum_{n \in N^C} y_{rn} = 1, \forall r \in R \tag{7}$$

$$\sum_{n \in (N - N^C)} y_{rn} = 0, \forall r \in R \tag{8}$$

$$\sum_{r \in R} y_{rn} f_{rj} \leq c_{nj}, \forall n \in N^C, j \in J \tag{9}$$

각 연산 노드에서 트래픽 처리에 사용할 수 있는 자원의 종류로는 CPU, GPU, 메모리가 존재하며 $J = \{\text{CPU}, \text{GPU}, \text{RAM}\}$으로 정의할 수 있다.

Ⅳ. 제안 방법

일반적으로 동적 네트워크 환경에서의 서비스 배치는 NP-Hard 문제이며, 트래픽 라우팅 또한 제약 조건이 존재하는 최단 경로 문제(CSP, Constrained Shortest Path)인 경우 NP-Hard로 간주된다 [24]. 따라서, 본 논문의 서비스 배치 및 트래픽 라우팅 문제는 NP-Hard이기 때문에 다항 시간 내 최적해 도출을 보장하기 어렵다. 본 논문에서는 이러한 문제를 해결하기 위해 메타휴리스틱 기법 중 유전 알고리즘(GA, Genetic Algorithm)을 활용한다. 그림 2는 본 논문에서 제안하는 유전 알고리즘 기반 최적화 방식의 절차를 보인다.

4.1 GA 기반 서비스 배치 및 라우팅 방법

유전 알고리즘을 활용한 서비스 배치 및 트래픽 라우팅에서 각 염색체(Chromosome)는 해(Solution)를 표현한다. 여기서 단일 염색체는 목적지가 되는 연산 노드를 포함하는 트래픽 경로를 나타낸다. 즉, 염색체를 구성하는 유전자(Gene)를 살펴보면, 첫 번째 유전자는 항상 트래픽의 출발 지점이 되는 네트워크 노드이고, 서비스 배치로 인해 결정되는 연산 노드는 트래픽의 목적 지점으로서 마지막 유전자에 위치한다. 한편, 제안하는 방법에서는 수식 (5)와 같이 각 트래픽 경로가 허용하는 최대 홉 수가 제한되어 있다. 따라서, 트래픽 경로를 표현하는 염색체의 길이 또한 한계가 존재한다. 이는 곧 유전 알고리즘에서 초기 해를 생성할 때, 출발 노드로부터 목적지가 될 수 있는 연산 노드를 식별하는 조건이 된다.

제안하는 유전 알고리즘 기반 서비스 배치 및 트래픽 라우팅 방식에서는 트래픽 요구사항이 있을 때, 기본적인 초기 해집단을 생성하기 위해 상기 언급된 제약 조건을 활용하여 너비 우선 탐색(BFS, Breadth-First Search)을 수행한다. 이를 통해 한도 홉 수 내에 도달할 수 있는 연산 노드를 목적지로 고려할 수 있다. 이후, 초기 해집단은 너비 우선 탐색을 통해 식별된 경로들을 초기 해집단 크기에 맞는 개수를 임의로 선택하여 생성한다.

이 외에도 제안하는 방법에서는 LLM을 활용하여 초기 해집단을 생성하는 기능을 제공한다. 초기 해집단을 생성하는 단계에서 LLM을 사용하는 경우, LLM은 네트워크 토폴로지 정보, 트래픽 요구사항, 링크 정보, 운용 비용 등이 포함된 입력값을 프롬프트(Prompt) 형태로 받는다. 이에 대한 결과로 LLM은 초기 해집단에 포함되어야 하는 초기 해들을 생성하며, 이들 중 초기 해집단 크기에 맞는 개수 또는 초기 해집단의 일정 비율을 만족하는 범위에서 선택되어 초기 해집단에 포함된다. LLM을 활용하는 과정에서 불완전한 추론 결과가 도출되는 것을 방지하기 위해 본 논문에서 제안하는 방법에서는 맥락 내 학습 방식을 활용한다. 즉, 입력으로 받는 프롬프트에서는 특정 네트워크 상황에서 생성되었던 초기 해집단을 나타내는 예시가 포함된다.

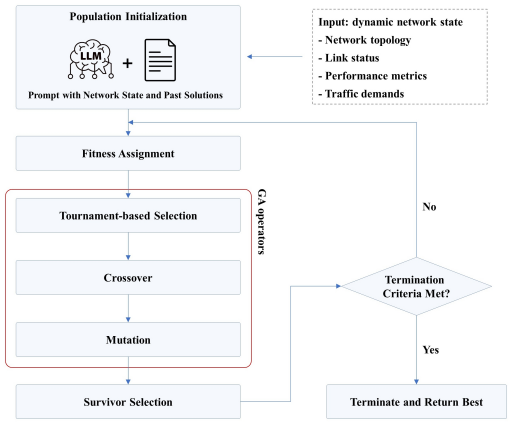


그림 2. 유전 알고리즘 기반 서비스 배치 및 라우팅 순서도
Fig. 2. Procedure of GA-based Service placement & routing

초기 해집단을 생성한 후에는 토너먼트 선택 방식(Tournament-based selection)을 활용하여 해집단 내

에서 부모 염색체 쌍을 선정한 후, 교차(Crossover)와 돌연변이(Mutation)와 같은 유전 연산을 확률적으로 발생시킨다. 이 과정에서 유전 연산으로 인해 단절된 경로나 순환 경로 등 불완전한 경로가 새로운 해로 포함되는 것을 피하기 위해 교차 연산에서는 부모 염색체 간 동일한 유전자, 즉, 두 경로 상 공통으로 존재하는 네트워크 노드 중 하나를 임의로 선택한 후 교차가 발생하는 지점으로 활용한다. 마찬가지로, 돌연변이 연산에서는 돌연변이가 발생하는 지점의 노드는 해당 노드의 전후에 위치한 노드의 공통 이웃 노드들 중 하나로 변경된다.

상기 과정은 유전 알고리즘이 종결 조건을 달성할 때까지 반복 수행함과 동시에 수식 (10)의 적합도 함수(Fitness function)를 통해 해의 최적화 정도를 측정하여 최적해를 산출한다. 제안하는 방법에서는 일정 세대 수가 반복되는 동안 최적해가 변경되지 않거나 미리 정의된 종결 세대수에 도달하면 알고리즘을 종료하는 것으로 종결 조건을 설정한다.

$$F = C_{total} + P_{total} \tag{10}$$

$$C_{total} = C_{links} + C_{nodes} \tag{11}$$

$$P_{total} = \sum_{z \in Z} P_z^{nw} + \sum_{j \in J} P_j^{cp} \tag{12}$$

$$P_j = \max(0, f_{rj} - c_{nj}) \times w_j \tag{13}$$

앞서 언급된 수식 (10)의 적합도 함수는 각 해의 최적화 정도를 나타낸 값으로서, 트래픽 요구사항을 충족하는 것과 동시에 최소 운용 비용을 가질 수 있도록 설계되었다. 예시로, 해당 적합도 함수를 통해 작은 값이 도출될수록 그 해는 적합한 서비스 배치 및 트래픽 라우팅 결과라는 것을 나타낸다.

구체적으로, 각 해의 적합도는 서비스 배치가 되는 연산 노드에서 트래픽 처리 과정 중 발생하는 운용 비용인 C_{nodes}와 트래픽 라우팅 경로를 구성하는 각 링크에서의 운용 비용 C_{links}을 합한 값인 C_{total}와 함께 트래픽 요구사항의 충족 여부에 따라 부여되는 벌점 P_{total}을 기반으로 계산된다. 해당 벌점은 수식 (12)의 정의에 따라 네트워크 도메인과 연산 도메인에서의 벌점을 합한 값으로 산출된다. 이 때, 네트워크의 벌점은 성능 지표 $z \in Z$에 관련된 충족 여부에 따라 부여되는 P_z^{nw}로 표현되며, 연산 노드의 벌점은 자원 종류 $j \in J$에 해당하는 가용 자원량이 트래픽 처리에 요구되는 정도보다 작을 경우 부여되는 벌점 P_j^{cp}로 표현된다. 또한, 각 벌점은 요구사항과 대비하여 위반된 수준을 나타내기 위해 가중치 w를 곱하여 산출한다. 수식 (13)은 자원 종류 j에 대한 벌점을 계산하는 예시이다. 각 자원 종류별 벌점에 활용되는 가중치는 CPU, GPU, 메모리 순으로 1500, 2000, 1000을 할당하여 활용한다.

4.2 LLM 기반 초기 해집단 생성

LLM 기반 초기 해집단 생성 방법에서는 네트워크 토폴로지, 노드 및 링크 정보, 서비스 요구사항 등이 주어졌을 때, 유전 알고리즘 기반 서비스 배치 및 트래픽 라우팅 수행에 필요한 초기 해집단을 생성하여 제공한다. 일반적인 유전 알고리즘에서 임의로 해집단을 생성하는 것과는 달리, LLM을 통해 다양성을 지닌 높은 품질의 초기 해집단 생성 역할을 수행하기 위해서는 미세조정을 통한 모델의 갱신이 요구된다. 하지만, 미세조정은 큰 연산 비용과 방대한 학습 데이터를 요구할 수 있다는 단점이 존재한다.

본 논문에서 제안하는 방법은 상기 문제에 대응하기 위하여 LLM의 미세조정 대신 프롬프트 내부에 몇 가지 예시를 포함하는 맥락 내 학습 방식을 적용한다. 이러한 방법에서는 프롬프트 내부에 포함되어야 하는 예시를 미리 준비해야 하는 과정이 요구되기 때문에 제안하는 방법에서는 선행연구 [25]의 그래프 신경망(GNN, Graph Neural Network) 기반 해 품질 예측 방법을 통한 초기 해 구성을 활용하였다. 구체적으로는 사전에 특정 네트워크 환경에서 GNN 기반 방식을 통해 고품질 해들을 산출한 후, 각 네트워크 환경과 이에 맞춰 산출된 고품질 해들을 연관지어 저장하여 이들 중 일부를 맥락 내 학습의 예시로 활용하는 것이며, GNN 모델 자체를 초기 해 생성 과정의 구성 요소로 포함하지 않는다. 따라서, 제안하는 방식은 초기 해 생성 단계에서 GNN 모델의 학습 또는 추론은 수행되지 않으며, GNN을 통해 학습 기반으로 수행하던 초기 해 생성을 LLM의 맥락 내 학습 기반 추론으로 대체하는 것을 목적으로 한다.

이와 같이 준비되는 예시는 주어진 네트워크 환경에서 특정 서비스 요구사항을 만족하는 트래픽 라우팅 경로를 의미하는 초기 해집단들 중, GNN을 통해 높은 품질로 판별되는 해집단이다. 즉, 예시는 네트워크 환경, 서비스 요구사항, 그에 대한 고품질 초기 해집단으로 구성되며, 이를 통해 LLM이 임의 생성 방법 대비 고품질의 초기 해를 구성할 수 있도록 유도한다.

맥락 내 학습 방식을 이용한 LLM 기반 서비스

```
# Task
Generate an initial population for a genetic algorithm, optimizing jointly a service placement and traffic routing.
Each example contains adjacency, computing node resources, edge performance metrics, demand, and population.

# Examples
Adjacency:{"1":[2,3],"2":[1,4],"3":[1,4,5],"4":[2,3,6],"5":[3,6],"6":[4,5]}
ComputingNodes: { "2": {"mem":100000000,"cpu":10000000,"gpu":1000000000,"rmem":100000000,"rcpu":10000000,"rgpu":1000000000,"cost":12.34}, …}
Edges: [{"s":1,"t":2,"cap":100,"avail":1.0,"delay":3.49,"rcap":100,"cost":5.11}, …]
Demand: {"source":"1","destination":"6","transmission_capacity":8.39,"maximum_delay":61.46,"minimum_availability":0.91,"maximum_hops":8,"memory":14,"cpu":6,"gpu":181}
Population: [[1,2,4,6],[1,3,5,6]]

# Output Format
Return only JSON:
{ "population": [[1, "...", 6]] }

# New Task
Source: 1
Destination: N
Adjacency: {"1":[2,5,8,...],"2":[1,3,...],"N":[7,10,...],...}
ComputingNodes: {"5":{"mem":100000000,"cpu":10000000,"gpu":1000000000,"cost":12.34}, 8":{"mem":100000,"cpu":10000,"gpu":1000000,"cost":5.64},"…}
Edges: [{"s":1,"t":2,"cap":200,"avail":1.0,"delay":0.24,"rcap":200,"cost":8.93}, {"s":7,"t":"3","cap":50,"avail":1.0,"delay":2.0,"rcap":50,"cost":5.33}, ...]
Demand: {"source":"1","destination":"N","transmission_capacity":4.52,"maximum_delay":45.3,"minimum_availability":0.95,"maximum_hops":8,"memory":42,"cpu":5,"gpu":512}
```

그림 3. 유전 알고리즘 기반 서비스 배치 및 트래픽 라우팅을 위한 LLM 프롬프트 예시
Fig. 3. Example of LLM prompt for GA-based service placement & traffic routing

배치 및 트래픽 라우팅을 위한 초기 해집단 생성에서는 LLM에게 이를 요청하는 프롬프트의 설계가 필수적이다. 제안하는 방법에서 사용하는 프롬프트의 구조는 LLM이 수행해야하는 서비스 배치 및 트래픽을 라우팅을 위한 초기 해집단을 정의하는 역할의 작업(Task), 맥락 내 학습 방식을 위한 예시(Examples), 결과로 도출되어야 하는 초기 해집단 출력 형식(Output Format), 그리고 초기 해집단이 생성되어야 하는 네트워크 정보와 서비스 요구사항을 명시한 새로운 작업(New Task)으로 구성된다. 그림 3은 일부 내용은 축약된 형태로 표현된 프롬프트의 예시를 보인다.

한편, 제안하는 방법에서는 추론 비용 절감과 초기 다양성 확보를 위해 한 번의 LLM 호출 당 미리 정의된 개수의 초기 해를 생성하는 배치(Batch) 방식을 채택하였다. 즉, 각 LLM 호출이 초기 해집단의 크기에 해당하는 개수의 해를 단일 호출로 생성하지 않고, 복수의 호출을 통해 생성한다. 예를 들어, 10개의 해가 초기 해집단을 구성할 경우, 제안하는 배치 방식에서는 5번의 LLM 호출을 통해 각각 2개의 해를 생성하여 초기 해집단에 포함시킬 수 있다. 이러한 배치 방식은 프롬프트와 추론 결과의 길이 제한으로 인해 정상적으로 해가 반환되지 않는 경우를 방지하기 위해 적용되었다.

마지막으로, LLM을 통해 생성된 초기 해는 해집단에 포함하기 전에 유효성 검사를 수행한다. 정상적인 출력 형식을 만족하지 않거나 네트워크 토폴로지상 유효하지 않은 경로가 생성된 경우, 해당 초기 해는 별도의 수정 없이 폐기(Discard)한다. 이로 인해 유효한 초기 해의 수가 설정된 초기 해집단의 크기에 미달하는 경우에는 기존 GA에서 사용하는 임의 초기화 방식을 통해 부족한 수의 초기 해를 추가로 생성한다. 이를 통해 LLM의 출력 오류가 발생하는 경우에도 사전에 설정된 초기 해집단의 크기를 유지한 상태에서 이후의 유전 알고리즘을 수행할 수 있도록 구성하였다.

V. 구현 및 성능 평가

5.1. 실험 환경 구성 및 성능 평가 시나리오

LLM을 통해 초기 해집단을 구성하는 유전 알고리즘 기반 서비스 배치 및 트래픽 라우팅 기능은 파이썬으로 구현되었다.

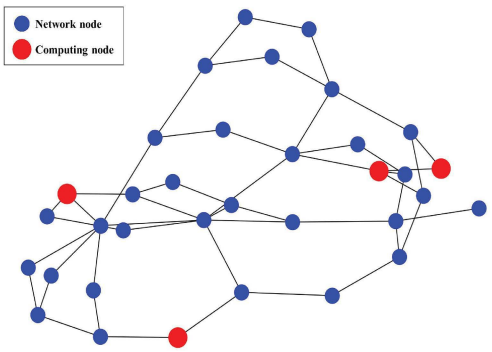


그림 4. Xspedius 네트워크 토폴로지 예시
Fig. 4. Example of Network topology for Xspedius

해당 기능 내 활용된 LLM은 Ollama [26]에서 제공되는 모델 중, 8B 매개변수를 가진 gemma4:e4b를 활용하였으며, 배치 방식의 LLM 호출에서 각

호출 당 반환되는 해의 개수는 2개로 설정하였다. 또한, LLM의 프롬프트와 응답 메시지의 한도를 고려하여 맥락 내 학습으로 인해 포함되는 예시는 1개로 제한하였다.

서비스 배치 및 트래픽 라우팅이 수행되어야 하는 네트워크는 Internet Topology Zoo [27]에서 제공되는 통신 네트워크 토폴로지 중 Xspedius를 사용하였다. 해당 네트워크 토폴로지는 총 34개의 노드가 98개의 유방향 링크를 통해 연결된 형태를 가진다. 이 때, 제안하는 방법의 성능 평가를 위하여 전체 노드 개수의 1/10보다 작은 임의 개수의 노드를 선택하여 연산 노드로 분류하며, 나머지는 네트워크 노드의 역할을 수행한다. 그림 4는 성능 평가에 사용된 Xspedius 토폴로지의 예시를 보인다.

표 1. 시뮬레이션 및 환경 설정
Table 1. Simulation and environment configuration

Category	Type	Value
Environment	Link capacity (Gbps)	{50, 100, 200}
	Link availability	[0.9999, 0.999999]
	Link operating cost	[1, 10]
	Computing operating cost	[1, 20]
Service req.	Transmission capacity (Gbps)	[0.1, 10]
	Maximum delay (ms)	[10, 100]
	Minimum availability	[0.9, 0.999]
	Required resource - CPU	[1, 10]
	Required resource - RAM	[1, 100]
	Required resource - GPU	[100, 1000]
GA	Population size	10
	Generations	100
	Mutation rate	0.1
	Crossover rate	0.8
LLM	Model	Gemma4:e4b
	Parameters, Quantization	8B, Q4_K_M
	Temperature	1
	Top_k, Top_p	64, 0.95

주어진 네트워크에서 서비스 배치 및 트래픽 라우팅은 필요 연산 자원(CPU, GPU, 메모리), 최소 가용성, 허용 가능한 최대 지연 시간, 최소 전송 용량 등을 포함하는 서비스 요청(Service request)이 발생할 때 수행된다. 이 때, 서비스 요청은 임의의 개수가 생성되며, 각 요청은 성능 요구사항 항목별로 임의의 범위 내 값이 할당된다. 또한, 제안하는 방법은 유전 알고리즘을 기반으로 해당 최적화를 수행하기 때문에 세대 수(Generation), 해집단의 크기(Population size), 유전 연산 확률(Mutation and crossover rate) 등이 매개변수로 설정되어야 한다. 특히, 이들 중 해집단의 크기는 초기해의 다양성을 위해 세밀한 설정이 필요한 매개변수이며, 네트워크 토폴로지에서 구성될 수 있는 라우팅 경로의 개수를 고려하여 10으로 설정되었다. 이와 관련된 상세 실험 환경 구성 정보는 표 1에 정리되어 있다.

5.2. 성능 분석

성능 평가는 서비스 배치와 트래픽 라우팅이 요구되는 서비스 요청을 25, 50, 75, 100개 생성한 후, 이를 대상으로 유전 알고리즘 기반 서비스 배치 및 트래픽 라우팅을 수행하여 산출한 평균값을 비교 분석하였다. 이 때, 평균값은 각 서비스 요청 개수의 경우마다 20회를 반복 수행하여 계산하였다. 대조군으로는 유전 알고리즘 기반 최적화 방법들을 구현하여 활용하였으며, 구체적으로는 임의로 해를 생성하는 기존 유전 알고리즘 방식(즉, BFS 경로 중 임의 선택)과 선행연구 [25]에서 초기 해집단의 품질을 GNN을 통해 예측한 후 고품질의 초기 해집단을 선택하여 활용하는 유전 알고리즘 방식을 사용하였다. GNN 기반 GA 방식에서는 Xspedius 토폴로지에서 10,000개의 데이터셋을 통해 학습을 수행한 GNN을 활용하였다. 모델 학습 시에는 은닉층의 개수와 미니배치(Mini-batch) 크기는 각각 64와 32로 설정하였으며, 학습률 0.001을 적용하여 총 100 에포크(Epoch) 동안 학습을 수행하였다. 모델의 파라미터 최적화에는 Adam 최적화를 사용하였다. 각 대조군들은 GA (Base)와 GA (GNN-based)로 표기되며, 제안하는 방식은 GA (LLM-based)로 명시된다.

첫 번째로, 그림 5는 서비스 요청 수의 증가에 따른 총 운용 비용(Total operational cost)을 비교한 결과이다. 전체적으로 서비스 요청 수가 증가할수록 모든 유전 알고리즘 기반 최적화 방법에서 총 운용 비용이 증가하는 것으로 관측되었다. 이는 서비스 요청 개수가 증가할수록 서비스 배치와 트래픽 라우팅에 더 많은 네트워크 자원이 할당되고 링크가 활성화되기 때문인 것으로 해석할 수 있다. 한편, GNN 기반 방식과 LLM 기반 방식에서는 서비스 요청 개수가 100개 미만인 경우 유사한 총 운용 비용이 발생하였지만 100개의 서비스 요청에서는 LLM 기반 방식이 가장 낮은 운용 비용을 달성하였다. 이는 LLM 기반 방법이 대규모 서비스 환경에서 다른 방식 대비 효율적인 초기 해집단을 생성할 수 있다는 가능성을 보인다.

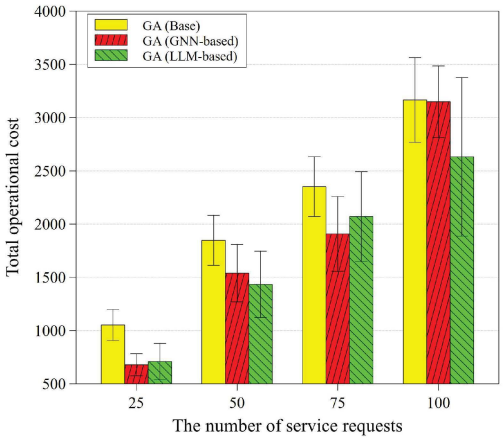


그림 5. 유전 알고리즘 기반 최적화의 평균 총 운용 비용
Fig. 5. Avg. total operational cost of GA-based optimization

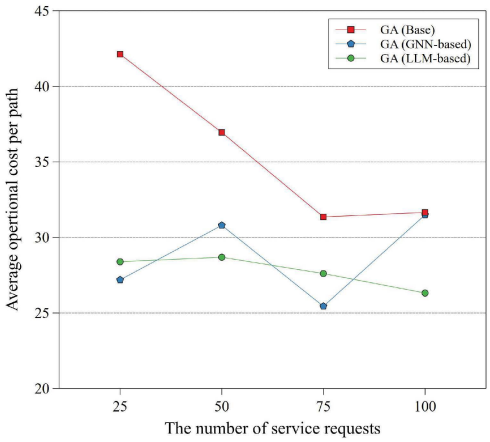


그림 6. 유전 알고리즘 기반 최적화의 평균 운용 비용
Fig. 6. Avg. operational cost of GA-based optimization

다음으로, 그림 6은 각 서비스 요청에 의해 생성되는 경로 당 평균 운용 비용(Average operational cost per path)을 비교한 결과를 나타낸다. 기존 유전 알고리즘 방식에서는 모든 서비스 요청 수에서 가장 높은 평균 운용 비용을 보였으며, 특히 서비스 요청 수가 적은 경우에는 다른 방식과의 차이가 크게 나타났다. 반면, GNN 기반과 LLM 기반 방식에서는 기존 방식보다 낮은 평균 비용을 달성하였다. 그 중에서도 제안하는 방식은 서비스 요청 수가 증가하여도 평균 비용의 변동이 비교적 작게 유지되었으며, 전체적으로 가장 안정적인 비용을 산출하는 것으로 확인되었다. 이는 LLM 기반 초기 해 생성이 특정 네트워크 환경에 과도하게 의존하지 않고 다양한 초기 해를 생성함으로써 보다 안정적인 탐색을 수행할 수 있기 때문으로 분석된다.

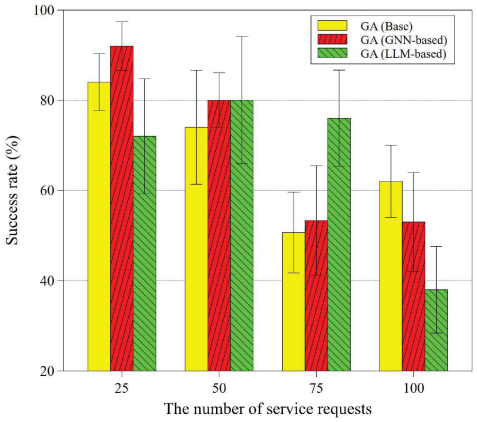


그림 7. 유전 알고리즘 기반 최적화의 서비스 성공률
Fig. 7. Service success rate of GA-based optimization

이 외에도, 그림 7은 서비스 요청 수에 따른 서비스 성공률(Success rate)을 비교한 결과를 나타낸다. 성능 평가 결과, 서비스 요청 수가 25개와 50개인 경우에는 GNN 기반 방식이 가장 높은 서비스 성공률을 보였다. 한편, 제안하는 LLM 기반 방식은 75개 이하의 서비스 요청에서 약 72~80%의 비교적 안정적인 성공률을 보였으며, 특히 75개 요청에서는 약 76%로 다른 방식보다 높은 성능을 나타냈다. 그러나 100개 요청에서는 성공률이 약 38%로 감소하여 75개 요청 대비 약 38% 감소하였다. 이는 서비스 요청 증가에 따른 자원 경합의 심화로 인해 LLM 기반 초기 해의 이점이 제한된 것으로 해석할 수 있다. 따라서 LLM 기반 초기화는 중간 수준의 네트워크 부하에서는 서비스 성공률 향상에 효과적이지만, 높은 부하 환경에서는 일부 한계가 존재함을 보인다.

마지막으로, 그림 8은 서비스 요청 수에 따른 평균 전송 용량(Average transmission capacity)과 평균 지연 시간(Average latency)을 비교한 결과를 나타낸다. 구체적으로는 각 서비스 요청에 따라 생성되는 트래픽 경로별 평균 전송 용량과 지연 시간을 의미한다. 성능 평가 결과, 평균 전송 용량 측면에서는 서비스 성공률 사례와 유사하게 LLM 기반 방식이 중간 수준의 네트워크 부하에서 가장 높은 전

송 용량을 보장하는 경로를 선정하였다. 또한, 평균 지연 시간의 경우, LLM 기반 방식이 변동폭이 가장 안정적인 지연 시간을 달성하였다. 반면, 기존 방식과 GNN 기반 방식에서는 서비스 요청 수에 따라 지연 시간의 변동폭이 크게 관측되었다. 이는 LLM 기반 초기화가 지연 시간과 네트워크 자원 활용을 동시에 고려하는 초기 해를 생성함으로써 최적화 과정에서 보다 균형 잡힌 경로를 선택하도록 유도한 것으로 분석된다. 특히, 서비스 요청 수가 증가하여도 지연 시간 측면에서는 크게 증가하지 않았다는 점은 제안하는 방법이 동적인 네트워크 환경에서도 안정적인 QoS를 제공할 수 있다는 가능성을 보여준다.

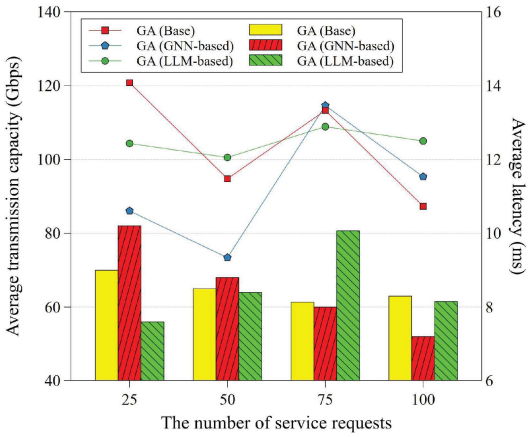


그림 8. 평균 전송량 및 지연 시간
Fig. 8. Avg. transmission capacity and latency

5.3. 논의 및 한계점

종합적인 성능 평가 결과, 제안하는 LLM 기반 초기 해 생성 방식은 모든 성능 지표에서 항상 최적의 결과를 보이지는 않았지만, 평균 운용 비용과 평균 지연 시간 측면에서 비교적 안정적인 성능을 나타냈다. 또한, 서비스 요청이 증가하는 환경에서도 운용 비용 증가를 효과적으로 억제하여, LLM을 활용한 초기 해집단 구성이 유전 알고리즘의 탐색 성능 향상에 기여할 수 있음을 확인하였다.

본 연구에서 LLM은 유전 알고리즘의 반복적인 탐색 과정에 직접 관여하지 않으며, 유전 알고리즘 수행 이전의 초기 해집단 생성 단계에서만 활용된다. 따라서, LLM 기반 초기화는 유전 알고리즘의 탐색 시작점에 영향을 미치는 방식으로 동작하며, 이후의 선택, 교차 및 돌연변이 과정은 기존 유전 알고리즘과 동일하게 수행된다. 이러한 결과는 LLM 기반 초기 해 생성 방식이 유전 알고리즘의 초기 탐색을 지원하기 위한 효과적인 방법으로 활용될 가능성을 보여준다.

다만, 초기 해집단의 품질이 유전 알고리즘의 수렴 과정과 최종 해의 품질에 미치는 영향을 보다 명확하게 규명하기 위해서는 세대별 최적 적합도(Best fitness) 변화와 수렴 세대에 대한 추가적인 분석이 필요하다. 또한, LLM 기반 초기 해 생성에는 별도의 추론 시간이 요구되므로, 실제 적용 가능성을 평가하기 위해서는 초기 해 생성에 소요되는 시간과 이후 유전 알고리즘의 최적화 수행시간을 구분하여 분석할 필요가 있다. 따라서 향후 연구에서는 다양한 네트워크 및 실험 환경을 대상으로 세대별 수렴 특성을 분석하고, LLM 추론 시간을 포함한 전체 수행시간을 정량적으로 평가함으로써 제안 방식의 일반화 가능성과 실제 적용 가능성을 보다 구체적으로 검증할 필요가 있을 것으로 사료된다.

이외에도, LLM을 통해 생성된 유효하지 않은 초기 해는 폐기하고 기존 임의 초기화 방식을 통해 부족한 해를 보완하지만, 이러한 출력 오류가 초기 해집단 구성에 미치는 영향에 대해서는 추가적인 분석이 요구된다. 향후 연구에서는 오류 유형별 발생 빈도를 정량화하고, 다양한 네트워크 및 추론 조건에 따른 LLM 출력의 신뢰성과 초기 해 생성의 안정성에 대한 분석이 필요하다.

Ⅵ. 결 론

5G와 다가오는 6G 네트워크에서는 동적 네트워크 환경에서 엄격한 성능 요구사항을 갖는 네트워크 서비스 제공이 주된 과제가 될 것으로 예상된다. 특히, 운용 비용을 최소화하기 위한 네트워크 서비스 배치 및 트래픽 라우팅 기법이 필수적이다. 이를 위해, 본 논문에서는 유전 알고리즘 기반의 서비스 배치 및 트래픽 라우팅 문제 해결 방안을 제안하였다.

제안하는 방법에서는 유전 알고리즘이 초기 해 구성에 따라 성능이 가변적이라는 한계를 극복하기 위해 대형 언어 모델과 맥락 내 학습을 활용하였다. 성능 평가 결과, 본 논문에서 제안하는 방법은 동적으로 변화하는 네트워크 환경에서 임의의 초기 해를 생성하는 기존 유전 알고리즘 방식 대비 운용 비용 측면에서 개선된 결과를 도출하는 것으로 검

증되었다. 향후 연구로는 유전 알고리즘 외 메타휴리스틱 기법에 기반한 성능 평가를 통한 범용성 검증과 메타휴리스틱 알고리즘의 최적 초매개변수 제어를 통한 성능 개선을 수행할 예정이다. 또한, 6G 시나리오를 기반으로 한 최적화 모델을 설계한 후, 제안하는 방법의 효용성을 다양한 네트워크 토폴로지에서 추가적으로 검증할 것이며, 논의 및 한계점에서 다룬 항목들에 대한 개선 및 성능 분석을 수행할 계획이다.

References

[1] Wang, Cheng-Xiang, et al. "On the road to 6G: Visions, requirements, key technologies, and testbeds." IEEE Communications Surveys & Tutorials 25.2 (2023): 905-974.

[2] Li, Chen, et al. "AI-native user-centric network for 6G." 2022 IEEE/CIC International Conference on Communications in China (ICCC Workshops). IEEE, 2022.

[3] Jeong-hwan Kim, et al. (2024). Dynamic Offloading and CPU Clock Scaling for Edge Computing in LEO Satellite Networks. The Journal of Korean Institute of Communications and Information Sciences, 49(4), 547-555.

[4] Sanghoon Lee, et al. (2025). QoS-aware Routing Algorithm in Satellite and Ground Integrated Networks. The Journal of Korean Institute of Communications and Information Sciences, 50(5), 781-789.

[5] Yang, Chien-Sheng, et al. "Toward 6G sustainable mobile communications." IEEE Wireless Communications 32.1 (2025): 44-50.

[6] Karunaharan, Vithusan, et al. "A Comprehensive Survey on 6G and Beyond for Climate Action: Toward Sustainable Connectivity." IEEE Access (2026).

[7] Benzaid, Chafika, and Tarik Taleb. "AI-driven zero touch network and service management in 5G and beyond: Challenges and research directions." Ieee Network 34.2 (2020): 186-194.

[8] Yao, Xin. "Finding approximate solutions to NP-hard problems by neural networks is hard." Information Processing Letters 41.2 (1992): 93-98.

[9] Peres, Fernando, and Mauro Castelli. "Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development." Applied sciences 11.14 (2021): 6449.

[10] Velasquez, Karima, et al. "Service placement for latency reduction in the internet of things." Annals of Telecommunications 72.1 (2017): 105-115.

[11] Farhadi, Vajiheh, et al. "Service placement and request scheduling for data-intensive applications in edge clouds." IEEE/ACM Transactions on Networking 29.2 (2021): 779-792.

[12] Rawat, Pradeep Singh, et al. "Efficient virtual machine placement in cloud computing environment using BSO-ANN based hybrid technique." Alexandria Engineering Journal 110 (2025): 145-152.

[13] Liu, Tong, et al. "Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing." IEEE Transactions on Mobile Computing 22.7 (2022): 3870-3881.

[14] Singh, Sandeep Kumar, Tamal Das, and Admela Jukan. "A survey on internet multipath routing and provisioning." IEEE Communications Surveys & Tutorials 17.4 (2015): 2157-2175.

[15] Ju, Ying, et al. "Dynamic load-balancing routing strategy for leo satellite networks based on spatio-temporal traffic prediction." IEEE Transactions on Aerospace and Electronic Systems (2025).

[16] Tabaeiaghdaei, Seyedali, et al. "Carbon-aware global routing in path-aware networks." Proceedings of the 14th ACM International Conference on Future Energy Systems. 2023.

[17] Seok Bin Son, et al. (2025). Trends in Network Optimization Using Large Language Models. The Journal of Korean Institute of Communications and Information Sciences, 50(7), 1073-1084.

[18] Nam, Sukhyun, Nguyen Van Tu, and James Won-Ki Hong. "LLM-Based AI Agent for VNF Deployment in OpenStack Environment." NOMS 2025-2025 IEEE Network Operations and

Management Symposium. IEEE, 2025.
[19] Xia, Handa, et al. "An LLM-Based Intent-Driven Framework for Automated Routing Configuration." 2025 IEEE 25th International Conference on Communication Technology (ICCT). IEEE, 2025.
[20] Lange, R.; Tian, Y.; Tang, Y. Large language models as evolution strategies. In Proceedings of the Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2024, pp. 579–582.
[21] Liu, S.; Chen, C.; Qu, X.; Tang, K.; Ong, Y.S. Large language models as evolutionary optimizers. In Proceedings of the 2024 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2024, pp. 1–8.
[22] Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q.V.; Zhou, D.; Chen, X. Large language models as optimizers. In Proceedings of the International Conference on Learning Representations, 2024, Vol. 2024, pp. 12028–12068.
[23] Ding, Hongqing, et al. "Optimal routing and heterogeneous resource allocation for computing-aware networks." ICT Express 10.3 (2024): 614-619.
[24] Garey, Michael R., and David S. Johnson. Computers and intractability. Vol. 29. New York: wh freeman, 2002
[25] Lee, Doyoung, and Taeyeon Kim. "Embedding-Assisted Genetic Algorithm for Routing Optimization in 6G Networks." Mathematics 13.21 (2025): 3536.
[26] Marcondes, Francisco S., et al. "Using ollama." Natural Language Analytics with Generative Large-Language Models: A Practical Approach with Ollama and Open-Source LLMs. Cham: Springer Nature Switzerland, 2025. 23-35.
[27] Knight, Simon, et al. "The internet topology zoo." IEEE Journal on Selected Areas in Communications 29.9 (2011): 1765-1775.

이 도 영 (Do-Young Lee)



2015 건국대학교 컴퓨터공학과 학사
2021 POSTECH 컴퓨터공학과 박사
2021~2023 삼성전자 네트워크 사업부 Staff Engineer
2023~2026 한국전자통신연구원 선임연구원
2026~ 현재 국립공주대학교 정보통신공학과 조교수
<관심분야> 6G, 네트워크 최적화, 지능형 네트워크

KNOM Review '26-01 Vol.29 No.01
https://doi.org/10.22670/knom.2026.29.1.13

적응형 세그먼트 길이 기반의 DASH 볼류메트릭 영상 스트리밍

유 상 우•, 홍 원 기°

Adaptive Segment Duration-based DASH Volumetric Video Streaming

Sangwoo Ryu•, James Won-Ki Hong°

요 약

볼류메트릭 영상은 몰입감 있는 경험을 제공하지만, 성공적인 스트리밍을 위해서는 높은 수준의 사용자 경험(QoE)을 보장하고 다양한 네트워크 환경과 서비스 및 사용자 시나리오에 유연하게 대응해야 한다. 특히 볼류메트릭 영상 서비스는 수동적 시청과 사용자 상호작용 단계가 공존하는 경우가 많아, 재생 안정성과 저지연 반응성 사이의 균형이 요구된다. 2D 영상과 다른 특성 때문에 기존의 DASH 방식으로는 해당 가변성을 효율적으로 처리하는 데 한계가 있다. 본 연구에서는 볼류메트릭 영상을 위한 DASH 시스템 내에서 세그먼트 길이를 동적으로 조절하는 적응형 전략을 제안한다. 이를 위해 다양한 세그먼트 길이를 지원하도록 확장된 MPD 구조를 도입하고, 실시간으로 세그먼트 길이를 조절하며 버퍼를 관리하여 반응성과 안정성 간의 균형을 유지하는 세그먼트 적응 엔진을 제시한다. 구현된 시스템을 다양한 시나리오에서 평가해 시스템의 가변적인 환경에 대한 적응성을 분석한다.

Key Words : Volumetric Video Streaming, Adaptive Bitrate Streaming, MPEG-DASH

ABSTRACT

While volumetric video provides an immersive experience, successful volumetric video streaming requires high quality of experience (QoE) and robust adaptation to diverse network environments and user scenarios. Moreover, volumetric video services often include both passive viewing and user interaction phases, demanding a balance between playback stability and low-latency responsiveness. Due to the differences from 2D video, existing Dynamic Adaptive Streaming over HTTP (DASH) method often struggles to address this variability efficiently. In this paper, we propose an adaptive strategy that dynamically adjusts the segment duration within the DASH system for volumetric video. To this end, we first introduce an extended Media Presentation Description (MPD) structure to support multiple segment durations. We then present a segment adaptation engine that adjusts the duration of the segment in real time, manages buffers, and pursues a balance between responsiveness and stability. We implemented our system and conducted extensive evaluations across diverse scenarios. The results illustrate that our system effectively adapts to varying conditions while balancing stability and responsiveness.

※ 이 논문은 2026년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원-대학ICT연구센터(ITRC)의 지원(IITP-2026-RS-2024-00437866, 50%)과 정보통신기획평가원의 지원을 받아 수행된 연구임 (RS-2024-00392332, 6G 네트워크 통합 지능평면 기술 개발, 50%)
• First Author : Pohang University of Science and Technology (POSTECH), Graduate School of Artificial Intelligence, rswoo@postech.ac.kr
° Corresponding Author : POSTECH, Department of Computer Science and Engineering, jwkhong@postech.ac.kr
논문번호 : KNOM2026-01-003, Received July 22, 2026; Revised August 10, 2026; Accepted August 18, 2026

Ⅰ. 서 론

확장현실(eXtended Reality)은 차세대 실감형 미디어 서비스로, 메타버스 및 네트워크 기술의 발전과 함께 성장하고 있다. 특히 볼류메트릭 영상(Volumetric Video)은 현실 세계의 물체와 장면을 3D로 재구성해 디지털 공간으로 가져와 사용자들에게 몰입감 있고 상호작용이 가능한 경험을 제공하는 확장현실의 핵심 기술이다. 이를 토대로 볼류메트릭 영상 서비스는 더욱 다양하고 복잡한 응용 시나리오로 확장되고 있다.

볼류메트릭 영상 서비스의 요구사항은 시나리오에 따라 크게 달라진다. 시청 중심의 수동적 시나리오는 재생 안정성이 중요하지만, 완전한 상호작용이 필요한 경우에는 사용자의 지속적인 움직임에 대한 반응성이 우선시된다. 두 방식이 혼재된 하이브리드 서비스도 존재하는데, 온라인 교육(강연자의 설명/3D 모델 상호작용), 스포츠 하이라이트(수동적 재생/자유 시점 탐색), 전자상거래(제품 시연/제품 탐색)가 대표적이다. 이러한 시나리오에서는 사용자의 현재 상태에 따라 시스템의 우선순위를 유연하게 조정하여 요구사항을 충족해야 한다.

일반적으로 비디오 스트리밍은 가변적인 네트워크 환경에서도 안정적인 콘텐츠 전송을 위해 MPEG-DASH [1]와 같은 적응형 비트레이트(ABR) 기술을 활용하며, 볼류메트릭 영상 스트리밍에서도 적응형 재생을 제공하기 위해 사용된다. 하지만 기존의 ABR 방식은 대부분 세그먼트 길이를 고정하기 때문에 하이브리드 서비스에서 상충하는 문제에 직면하게 된다. 예를 들어, 10초와 같은 긴 세그먼트는 HTTP 요청 오버헤드를 줄이고 영상 품질을 지속적으로 유지하여 안정성을 높이지만, 사용자 상호작용을 반영하기 위해 다음 세그먼트까지 기다려야 하므로 높은 지연 시간이 발생한다. 반면 1초 내외의 짧은 세그먼트는 즉각적인 상호작용이 가능하지만, 빈번한 HTTP 요청으로 인한 오버헤드가 발생하며 네트워크 환경이 불안정할 경우 버퍼링에 취약하다는 단점이 있다.

이러한 한계는 고정된 세그먼트 방식 대신 동적인 세그먼트 접근 방식의 필요성을 제시한다. 기존의 2D 영상용 동적 세그먼트 기법들 역시 이를 탐구한 바 있으나, 수동적인 시청 경험을 전제로 하여 안정적인 재생에만 최적화되어 있다. 따라서 상호작용이 핵심 요소인 볼류메트릭 영상 스트리밍에서는 재생 안정성과 상호작용 반응성 사이의 상충 관계를 효율적으로 관리하는 기술이 필요하다.

따라서 본 연구는 시청 중심의 수동적 환경과 사용자 상호작용이 공존하는 하이브리드 시나리오에서 안정성과 반응성을 효과적으로 확보하기 위해, 적응형 세그먼트 길이를 활용한 DASH 기반 볼류메트릭 영상 시스템을 제안한다. 이를 위해 세그먼트 길이를 결정하고 버퍼를 관리하는 세그먼트 적응 엔진을 설계한다. 해당 엔진은 네트워크 효율과 재생 안정성이 필요한 경우 긴 세그먼트를 우선 선택하며, 사용자 상호작용이 감지되면 짧은 세그먼트로 전환해 적절한 버퍼 관리를 수행하여 상호작용 지연 시간을 최소화한다. 다양한 네트워크 환경과 상호작용 시나리오를 바탕으로 제안하는 시스템의 성능을 평가하였다. 본 연구의 기여는 다음과 같다.

1) 상호작용 및 버퍼 상태에 따라 세그먼트 길이를 결정하는 적응형 엔진을 포함한 DASH 기반 볼류메트릭 영상 시스템을 제안

2) 다양한 네트워크 환경과 상호작용 시나리오를 바탕으로 시스템을 평가하여 제안 기법의 적응성을 분석

Ⅱ. 관련 연구

본 장에서는 볼류메트릭 영상 스트리밍 연구 현황과 적응형 비트레이트 스트리밍을 설명한다.

1) 볼류메트릭 영상 스트리밍

볼류메트릭 영상 스트리밍은 지난 몇년 간 꾸준히 연구되어 왔으며, 다양한 네트워크 및 서비스 요구사항을 만족시키기 위해 압축 코덱, 뷰포트 적응, 전송 최적화 등 다양한 기술들이 적용되었다. PCL [2], Draco [3], V-PCC [4] 등의 압축 코덱은 전송 효율성을 높이는 핵심 요소이며, 이를 바탕으로 전체적인 스트리밍 시스템을 발전시키기 위한 다양한 연구들이 수행되었다. ViVo [5] 및 GROOT [6]는 효율적인 사용자 시점 처리 방식을 사용했으며, YuZu [7]와 VoLUT [8]는 3D 초해상화 기반 대역폭 절감 방식을 제안하였다. MuV2 [9]는 다중 사용자 시나리오에서 엣지 기반의 콘텐츠 결합 및 공유 기술을 제안하였다. 이처럼 많은 연구들이 대역폭 절감을 위한 기술에 집중하고 있다. 본 연구는 대역폭 절감보다 스트리밍 시스템의 적응성 측면에 중점을 둔다.

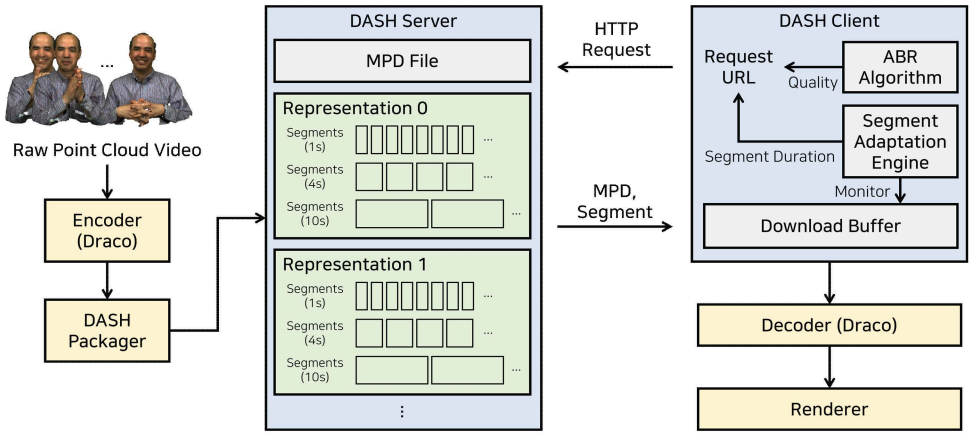


그림 1. 적응형 세그먼트 길이를 활용한 DASH 기반 볼류메트릭 영상 스트리밍 시스템 구조
Fig. 1. System architecture for DASH-based volumetric video streaming with adaptive segment duration

2) 적응형 비트레이트(ABR) 스트리밍

2D 영상 전송을 위한 적응형 비트레이트(ABR) 스트리밍 기술은 네트워크 환경 변화에 대응하기 위해 널리 연구되어 왔으며, 대부분 HLS [10]나 MPEG-DASH [1]와 같이 영상을 작은 단위로 분할해 적절한 비트레이트를 선택하는 방식을 따른다. LL-DASH [11]나 WebRTC [12]와 같이 저지연 요구사항을 만족하기 위한 방식들도 제안되었다. 볼류메트릭 영상 스트리밍 분야에서도 관련 연구들이 진행되었는데, 주로 PCC-DASH [13], DASH-PC [14], VV-DASH [15]와 같이 기존 DASH를 확장하여 포인트 클라우드 전송을 지원하거나, 다중 설명(multiple description) 기법과 WebRTC 혼잡 제어를 결합한 방식도 제안되었다 [16, 17].

2D 영상 스트리밍에서 세그먼트 길이를 가변적으로 조절하는 연구들이 존재하지만, 접근 방식과 최적화 목표는 볼류메트릭 영상의 요구사항과 차이가 있다. Schwarzmann 등 [18]은 장면 전환과 같은 콘텐츠 특성을 기반으로 세그먼트 길이를 조절하여 인코딩 효율을 개선했으나, 클라이언트의 실시간 네트워크나 버퍼 상태보다는 콘텐츠 자체에 집중한 방식이다. 또한, 다른 연구들 [19, 20, 21]은 클라이언트 상태에 따라 세그먼트 길이를 동적으로 변화시키는 방안을 제안하였으나, 최적화 목표는 재생 안정성 확보에 초점을 두고 있다. 반면, 볼류메트릭 영상은 사용자의 상호작용을 지원해야 하므로 안정성과 반응성 사이의 균형 및 버퍼 관리가 필수적이다. 본 논문에서는 수동적 시청과 능동적 상호작용 단계가 공존하는 환경에서, DASH의 세그먼트 기반 적응 방식을 확장하여 재생 안정성과 상호작용 반응성을 동시에 고려한 스트리밍 기법을 제안한다.

Ⅲ. 시스템 설계

이 장에서는 제안하는 DASH 기반 볼류메트릭 영상 스트리밍 시스템 핵심 설계들을 서술한다.

1) 볼류메트릭 영상 스트리밍 시스템

그림 1은 제안하는 시스템의 전체 구조를 나타낸다. 서버는 온디맨드(On-Demand) 형식으로 콘텐츠를 전송하도록 설계되었다. 원본 포인트 클라우드 영상은 Draco [3] 인코더로 압축된 후 다양한 길이의 세그먼트로 패키징된다. 이때 MPD(Media Presentation Description) 형식의 메타데이터 파일이 생성되는데, 본 연구에서는 다중 세그먼트 길이를 지원하도록 MPD 구조를 확장하였다. 이 세그먼트와 메타데이터 파일은 HTTP 서버를 통해 제공된다. DASH 클라이언트는 세그먼트 적응 엔진과 ABR 알고리즘을 사용하여 원하는 길이와 품질의 세그먼트를 다운로드하고, 이를 프레임 단위로 추출한 뒤 Draco 디코더와 렌더러로 전달하여 재생한다. 결과적으로 본 시스템은 온디맨드 시나리오에서 볼류메트릭 영상을 스트리밍하기 위한 구조로 설계되었다.

2) MPD 구조 및 세그먼트

DASH 기반 영상 스트리밍에서 Media Presentation Description(MPD)은 영상 메타데이터

를 XML 형식으로 저장하는 표준적인 방식이다. DASH 클라이언트는 MPD 파일에 포함된 메타데이터를 활용하여 원하는 품질의 세그먼트를 다운로드한다. 일반적으로 MPD 파일은 AdaptationSet 요소 내부에 품질을 나타내는 여러 Representation 요소를 가지며, 각 Representation 요소는 세그먼트 정보를 담고 있는 SegmentList 요소를 포함한다. 포인트 클라우드용 DASH 스트리밍 시스템을 제안한 기존 연구들도 이러한 기본 구조를 따른다.

본 연구에서는 서로 다른 길이의 세그먼트를 지원하기 위해 하나의 Representation 요소가 다중 SegmentList를 가질 수 있도록 설계하였다. 이에 따라 각 SegmentList는 segmentSize 속성을 가지며, SegmentUrl은 세그먼트 길이에 따라 패키징된 파일의 URL과 메타데이터를 속성으로 포함한다. 그림 2는 이러한 설계를 반영한 MPD 구조의 예시이다.

```
<?xml version='1.0' encoding='UTF-8'?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" type="static"
    profiles="urn:mpeg:dash:profile:isoff-live:2011"
    mediaDuration="900" mediaFPS="30">
    <BaseURL>https://www.example.com/</BaseURL>
    <Period>
        <AdaptationSet id="0" codecs="draco" mimeType="3d/ptcl">
            <Representation id="0" bandwidth="50569456"
                            availableDurations="30, 120, 300">
                <FrameTemplate
                    media="/phil9/frames/compressed_ext/0/$frame%04d$.drc"
                    startNumber="0" />
                <SegmentList startNumber="0" segmentNum="30" segmentSize="30">
                    <SegmentUrl
                        media="/phil9/segments/segments-30f/0/segment_00000.bin"
                        startNumber="0" duration="30" />
                    <SegmentUrl
                        media="/phil9/segments/segments-30f/0/segment_00001.bin"
                        startNumber="30" duration="30" />
                    ...
                </SegmentList>
                <SegmentList startNumber="0" segmentNum="8" segmentSize="120">
                    ...
                </SegmentList>
                ...
            </Representation>
            <Representation id="1" bandwidth="180941216"
                            availableDurations="30, 120, 300">
                ...
            </Representation>
            ...
        </AdaptationSet>
    </Period>
</MPD>
```

그림 2. DASH MPD 구조 예시
Fig. 2. An example DASH MPD Structure

세그먼트 길이에 따른 특성(III-3장에서 상세 설명)을 고려하여 1초, 4초, 10초 세 가지 길이로 세그먼트를 구성하였다. VV-DASH [15]에서 제안하는 DVV 세그먼트를 참조하여, 각 세그먼트는 클라이언트에서 추출 및 디코딩이 가능하도록 설계하였다. 세그먼트 헤더에는 시작 프레임 인덱스, 포함된 프레임 수, 전체 데이터 길이, 그리고 프레임별 오프셋 정보가 포함된다. 페이로드에는 압축 효율과 디코딩 성능 간의 균형이 뛰어나 연구에서 일반적으로 사용되는 Google Draco로 압축한 프레임별 데이터가 담겨 있다.

3) 세그먼트 적응 엔진

세그먼트 적응 엔진은 상태 머신(state machine) 방식으로 작동하며, 현재 상태에 기반하여 최적의 결정을 내리기 위해 STARTUP, VOD, INTERACTION의 세 가지 모드 중 하나를 선택한다. 세그먼트 길이에 따른 특성을 고려하여 1초(30프레임), 4초(120프레임), 10초(300프레임) 단위의 세그먼트를 구성하였다. 1초 세그먼트는 사용자의 상호작용이 빠르게 반영되어 반응성 측면에서 유리하지만, 잦은 HTTP 요청과 세그먼트 추출 과정으로 인해 클라이언트 오버헤드가 발생할 수 있다. 반면 10초 세그먼트는 HTTP 요청 횟수가 적어 오버헤드가 낮고 품질이 안정적이나, 상호작용에 대한 반응이 느리고 큰 세그먼트 크기로 인해 추출 과정에서 오버헤드가 발생할 수 있다. 4초 세그먼트는 1초와 10초 세그먼트의 절충점 역할을 한다.

Input: $is\_interacting$, $buffer\_level \in [0, 1]$
Output: $segment\_duration$, $need\_flush$

$now \leftarrow time()$
$need\_flush \leftarrow False$
if $is\_interacting$ **then**
  if $current\_state \neq$ *'INTERACTION'* **then**
    $current\_state =$ 'INTERACTION'
    $max\_buf\_size \leftarrow 2\ seconds$
    $need\_flush \leftarrow True$
if $current\_state =$ *'INTERACTION' and* $now - first\_interaction \geq cooldown$ **then**
  $current\_state =$ 'VOD'
  $max\_buf\_size \leftarrow original\_max\_buf\_size$
if $current\_state =$ *'STARTUP'* **then**
  return $(1s, False)$
if $current\_state =$ *'INTERACTION'* **then**
  return $(1s, need\_flush)$
if $buffer\_level < buffer\_lower\_threshold$ **then**
  return $(1s, False)$
if $buffer\_level > buffer\_upper\_threshold$ **then**
  return $(10s, False)$
return $(4s, False)$

그림 3. 세그먼트 길이 적응 알고리즘
Fig. 3. Segment Adaptation Algorithm

시스템이 시작되면 STARTUP 모드를 유지하며 짧은 1초 세그먼트를 이용해 빠르게 버퍼를 채우고, 첫 프레임이 렌더링된 직후 VOD 모드로 전환된다.

이러한 초기 단계 이후, 세그먼트 적응 엔진은 상호작용성과 다운로드 버퍼 수준이라는 두 가지 주요 신호를 기준으로 세그먼트 길이를 전환한다. 먼저 상호작용성의 경우, 카메라 움직임이 임계값을 초과하면 엔진은 1초 세그먼트를 사용하는 INTERACTION 모드로 진입한다. 이 모드에서는 최신 정보를 즉각 반영하기 위해 최소한의 안전 프레임을 제외한 나머지 다운로드 버퍼를 비우며(*need_flush*), 반응성과 안정성 간의 균형을 맞추기 위해 최대 버퍼 크기를 2초로 제한하여 지연 시간을 최소화한다. 일정 시간 동안 상호작용이 감지되지 않으면 VOD 모드로 복귀하고 최대 버퍼 크기도 재설정한다. 다음으로 다운로드 버퍼 수준의 경우, 버퍼가 낮으면 1초 세그먼트를 사용하여 네트워크 상태를 빠르게 갱신하고 버퍼를 채운다. 반면 버퍼 수준이 높으면 시스템이 안정적인 것으로 판단하여 10초 세그먼트를 사용하여 효율성을 높인다.

세그먼트 선택 외에도 엔진은 버퍼 수준을 지속적으로 모니터링하며 위험 상황을 감지한다. 고품질의 큰 세그먼트를 다운로드하는 도중 버퍼 수준이 최대 버퍼 크기 대비 특정 임계값(20%) 밑으로 떨어지면, 진행 중인 다운로드를 취소하고 더 짧은 세그먼트로 강제 전환하여 네트워크 대역폭을 빠르게 다시 계산한다. 그림 3은 본 연구의 세그먼트 길이 적응 방식이다. 임계값은 실험을 통해 경험적으로 결정하였다. 실험에서는 버퍼 수준(*buffer_level*)을 최대 버퍼 크기를 기준으로 0~1로 정규화하고 버퍼 하한 임계값(*buffer_lower_threshold*)과 상한 임계값(*buffer_upper_threshold*)은 각각 0.3과 0.8로 설정하였으며, 쿨다운(*cooldown*)과 최대 버퍼 크기(*max_buf_size*)는 각각 5초와 15초로 설정하였다.

Ⅳ. 시스템 구현

이 장에서는 DASH 기반 볼류메트릭 영상 스트리밍 시스템의 서버, 클라이언트에 대한 구체적인 구현 방식을 설명한다.

1) 서버

볼류메트릭 영상의 적응형 스트리밍을 위한 DASH 서버는 Go 언어 [22]로 구현하였다. 볼류메트릭 영상으로는 MVUB(Microsoft Voxelized Upper Bodies) 데이터셋 [23]의 Phil-9 영상을 사용했으며, 프레임 당 평균 33.4만 개의 포인트로 구성되어 있다 (평균 7.09 MB, 420 Mbps). VV-DASH [15]를 참고해 245 프레임 길이의 영상을 반복해 총 900 프레임 길이의 영상으로 수정하였다. Open3D[24]를 사용하여 포인트 수를 10%, 40%, 70%, 100%로 샘플링해 네 가지 품질의 데이터를 생성하였고, DracoPy [25]를 이용해 인코딩하였다.

2) 클라이언트

볼류메트릭 영상의 적응형 스트리밍을 위한 DASH 클라이언트는 Python으로 구현하였다. 세그먼트 다운로드, 디코딩, 렌더링으로 구성된 클라이언트 파이프라인을 효율적으로 실행하기 위해 asyncio(Python 내장)와 aiohttp [26]를 활용해 비동기 방식으로 동작하도록 설계하였다. 실험에는 대역폭 기반의 ABR 알고리즘을 적용하여 세그먼트 길이 변화와 독립적으로 비트레이트를 제어하도록 구성하였으며, 프레임 디코딩에는 DracoPy를 사용하였다. 또한, 연산 집약적인 작업을 효율적으로 처리하기 위해 최대 6개의 프로세스를 사용하는 멀티프로세싱을 사용하였다. 평가를 위해 디코딩된 포인트 클라우드는 Open3D의 헤드리스(headless) 모드로 렌더링하였으며, 카메라 움직임을 통해 상호작용을 구현하였다.

Ⅴ. 실험 환경

이 장에서는 제안하는 적응형 세그먼트 기반의 볼류메트릭 영상 스트리밍 시스템을 분석하기 위한 실험 환경을 설명한다.

1) 테스트베드 구성

서버, 클라이언트, 그리고 네트워크 에뮬레이터는 단일 워크스테이션에서 Docker [27] 컨테이너를 이용하여 구동하였다. 워크스테이션은 Intel Xeon(R) Silver 4215R CPU, Quadro RTX 5000 GPU, 64GB RAM으로 구성되어 있다. 네트워크 환경 제어를 위해 Docker 네트워크 내부에서 Linux tc 명령어를 사용하였으며, 콘텐츠 품질과 시나리오를 바탕으로 설정된 대역폭 트레이스를 활용하여 클라이언트의 트래픽을 제어하였다.

2) 네트워크 시나리오

제안하는 시스템을 평가하기 위해 네 가지 시나리오를 정의하였다. 각 시나리오를 수행하는 동안

표 1. 시나리오 1(정적 네트워크 환경)에서의 측정 결과
Table 1. Evaluation results of Scenario 1 (Stable Network)

Segment Duration	Bitrate (Mbps)	Startup Time (s)	Stall Time (s)	Num. of Requests
Fixed-1s	414.54	0.18	0.0	61
Fixed-4s	396.05	0.32	0.0	17
Fixed-10s	339.45	0.78	0.0	7
Adaptive (Proposed)	414.54	0.16	0.0	16

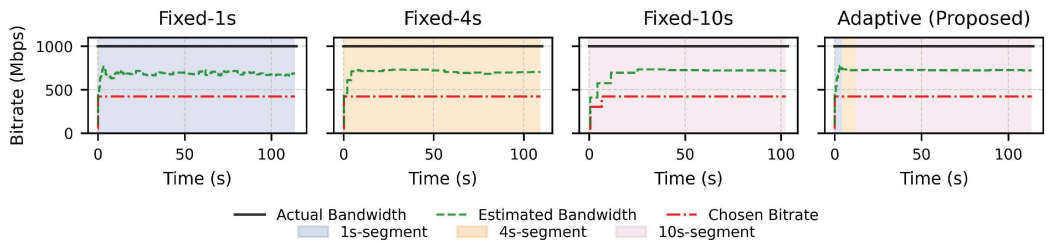


그림 4. 시나리오 1에서 시간의 흐름에 따른 비트레이트 및 세그먼트 길이 선택
Fig. 4. Bitrate and segment duration selection over time on Scenario 1

900 프레임의 포인트 클라우드 영상을 2회 반복 재생하여(30fps 기준 1분) 평가 지표를 측정하였다.

- 시나리오 1 (정적 네트워크): 대역폭을 1Gbps로 고정하여 모든 품질의 볼류메트릭 영상을 제공할 수 있게 하였다.
- 시나리오 2 (정적 네트워크): 시나리오 1의 네트워크 상황에서 정해진 시간대(그림 5의 빗금 영역)에 상호작용을 추가하였다.
- 시나리오 3 (동적 네트워크): 볼류메트릭 영상의 화질 별 대역폭을 고려해, (600Mbps, 400Mbps, 200Mbps, 100Mbps)의 대역폭을 (5초, 5초, 5초, 10초) 주기로 반복하는 시나리오를 구성하였다.
- 시나리오 4 (동적 네트워크): 시나리오 3의 네트워크 상황에서 정해진 시간대(그림 7의 빗금 영역)에 상호작용을 추가하였다.

3) 실험 과정 및 평가 지표

제안하는 세그먼트 적응 기법(Adaptive)을 세 가지 고정 길이 세그먼트 방식(Fixed-1s, Fixed-4s, Fixed-10s)과 비교하였다. Fixed-1s는 짧은 세그먼트나 청크(chunk)를 사용하는 LL-DASH [11]와 같은 저지연 프로토콜을 상정하였으며, Fixed-10s는 안정적인 VoD 스트리밍을 위한 기존 DASH 표준을, Fixed-4s는 중간 설정을 나타낸다. 2D 영상 스트리밍의 가변 세그먼트 방식[19, 20, 21]은 상호작용 상황이 존재하지 않아 반응성을 고려하지 않고, 대역폭 및 시스템 부담이 볼류메트릭 영상과 다른 특성을 가지고 있어 포함하지 않았다.

평가 지표로는 평균 비트레이트(수신 품질), 시작 시간(첫 프레임 지연 시간), 스톨 시간(버퍼가 비어 있는 총 시간), 그리고 HTTP 요청 횟수(자원 효율성)를 측정하였다. 상호작용을 포함하는 시나리오 2과 4에 대해서는 상호작용이 감지된 시점부터 해당 프레임이 렌더링될 때까지 걸리는 시간인 상호작용 응답 지연 시간을 추가로 측정하였다. 프레임 속도(FPS)는 클라이언트 기기 성능의 영향을 크게 받는 연산 집약적인 디코딩 및 렌더링 이후에 측정되었으므로 평가 및 분석에는 활용하지 않았다 [28].

Ⅵ. 실험 결과 및 분석

1) 시나리오 1 실험 결과

표 1과 그림 4는 시나리오 1의 결과를 보여준다. 시나리오 1은 네트워크 상태가 안정적인 환경이므로, 대부분의 경우 높은 품질의 세그먼트가 선택되었다. Fixed-10s의 경우 세그먼트 길이가 길어 초기 선택된 저품질 세그먼트가 평균 미디어 비트레이트를 낮추는 결과를 보였다. 반면, 제안하는 적응형 방식(Adaptive)은 초기에 1초 세그먼트를 사용하다가 시스템이 안정화됨에 따라 순차적으로 4초 및 10초 세그먼트를 선택하였다. Fixed-1s와 적응형 방식은 초기 단계에서 짧은 세그먼트를 사용하므로 시작 시간이 짧게 측정되었다.

표 2. 시나리오 2(정적 네트워크 환경 및 상호작용)에서의 측정 결과
Table 2. Evaluation results of Scenario 2 (Stable Network + Interaction)

Segment Duration	Bitrate (Mbps)	Startup Time (s)	Stall Time (s)	Num. of Requests	Interaction Latency (s)
Fixed-1s	414.54	0.18	0.0	61	27.71
Fixed-4s	396.05	0.33	0.0	17	32.69
Fixed-10s	359.08	0.73	0.0	7	38.79
Adaptive (Proposed)	411.66	0.12	0.0	42	5.39

표 3. 시나리오 3(동적 네트워크 환경)에서의 측정 결과
Table 3. Evaluation results of Scenario 3 (Dynamic Network)

Segment Duration	Bitrate (Mbps)	Startup Time (s)	Stall Time (s)	Num. of Requests
Fixed-1s	197.01	0.20	0.0	61
Fixed-4s	277.27	0.44	0.0	17
Fixed-10s	199.94	1.16	1.30	7
Adaptive (Proposed)	225.00	0.16	0.0	31

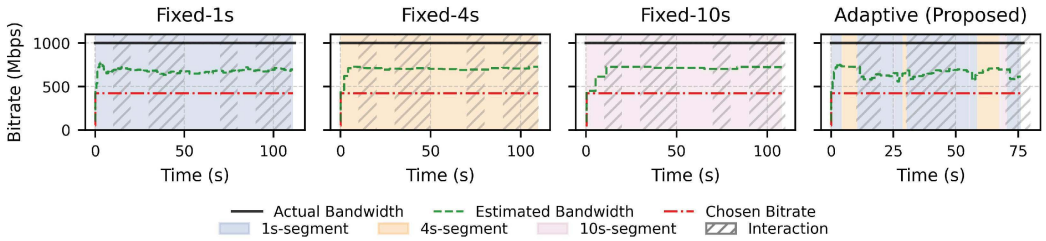


그림 5. 시나리오 2에서 시간의 흐름에 따른 비트레이트 및 세그먼트 길이 선택
Fig. 5. Bitrate and segment duration selection over time on Scenario 2

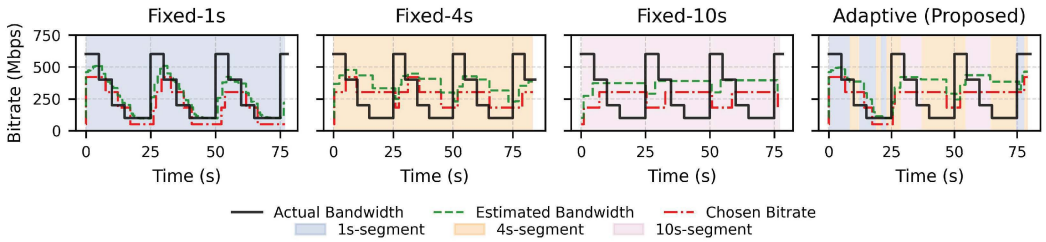


그림 6. 시나리오 3에서 시간의 흐름에 따른 비트레이트 및 세그먼트 길이 선택
Fig. 6. Bitrate and segment duration selection over time on Scenario 3

2) 시나리오 2 실험 결과

표 2과 그림 5는 시나리오 2의 결과를 보여준다. 시나리오 2는 시나리오 1과 유사하게 안정적인 네트워크 환경을 가정하므로 비트레이트 결과 역시 발생하는 구간에서 적응형 방식(Adaptive)이 1초 세그먼트를 선택한다는 점이다. 적응형 방식은 적절한 버퍼 관리를 통해 낮은 응답 지연 시간을 보인 반면, 고정 길이 방식들은 이미 버퍼에 쌓여 있는 프레임들로 인해 높은 응답 지연 시간이 측정되었다.

그림 8은 시나리오 2에서 적응형 방식의 시간에 따른 버퍼 수준 변화를 보여준다. 상호작용이 감지되면(빗금 영역), 진행 중인 다운로드를 취소하고 1초 세그먼트를 선택하며 버퍼를 비운다(일부 안전 프레임은 유지). 또한, 응답 지연 시간을 낮게 유지하기 위해 최대 버퍼 크기를 기존 수준의 20%로 조정한다. 상호작용이 종료된 후에는 버퍼를 다시 안정적인 수준까지 채우는 과정을 거친다. 이처럼 초기 상호작용 지연시간은 버퍼에 남아있는 상호작

표 4. 시나리오 4(동적 네트워크 환경 및 상호작용)에서의 측정 결과
Table 4. Evaluation results of Scenario 4 (Dynamic Network + Interaction)

Segment Duration	Bitrate (Mbps)	Startup Time (s)	Stall Time (s)	Num. of Requests	Interaction Latency (s)
Fixed-1s	191.00	0.14	0.0	61	14.06
Fixed-4s	277.48	0.46	0.0	17	15.58
Fixed-10s	240.51	0.97	0.0	7	22.93
Adaptive (Proposed)	207.80	0.16	0.0	58	3.80

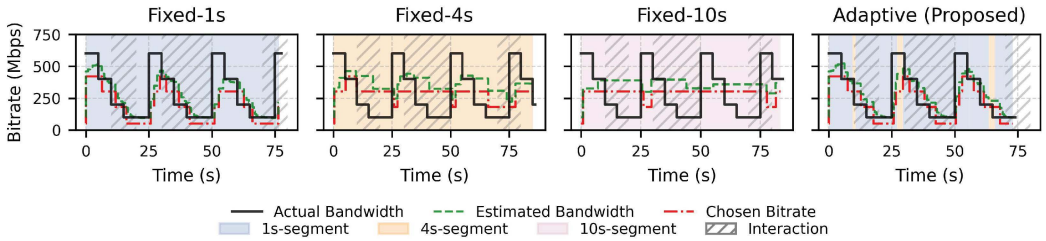


그림 7. 시나리오 4에서 시간의 흐름에 따른 비트레이트 및 세그먼트 길이 선택
Fig. 7. Bitrate and segment duration selection over time on Scenario 4

용 미반영 프레임의 양에 주로 영향을 받으며, 따라서 상호작용이 존재하는 경우 제안하는 시스템에서는 적응형 세그먼트 길이보다 버퍼 비우기 등의 버퍼 관리의 역할이 크게 작용하게 된다. 적응형 세그먼트 길이는 상호작용 구간 동안 지속되는 변화에 빠르게 반응하고, 상호작용 이후의 VOD 구간에서 다시 안정적인 재생 상태를 제공할 수 있도록 한다.

적응형 방식의 시작 시간은 Fixed-1s와 유사하게 측정되었으며, 상호작용으로 인해 이전 시나리오보다 HTTP 요청 횟수가 높게 나타났다.

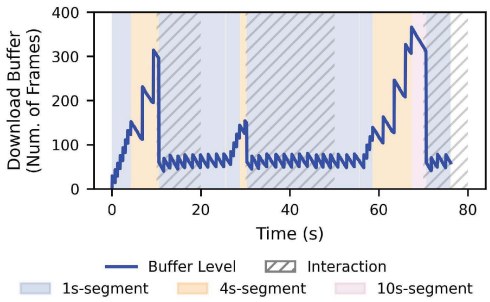


그림 8. 시나리오 2에서 시간의 흐름에 따른 버퍼 수준
Fig. 8. Buffer level over time on Scenario 2

3) 시나리오 3 실험 결과

표 3과 그림 6은 시나리오 3의 결과를 보여준다. 시나리오 3은 네트워크 상태가 변동적인 환경이므로, 네트워크에 적응하거나 미리 다운로드된 세그먼트를 활용하여 재생 안정성을 확보해야 한다. Fixed-1s는 세그먼트 길이가 짧아 대역폭 갱신이 빈번하게 일어나며 현재 네트워크 환경에 빠르게 적응한다. 반면, Fixed-4s와 Fixed-10s는 세그먼트 길이가 길어 대역폭 갱신이 즉각적이지 않으며, 이전에 다운로드된 세그먼트를 사용한다. 그림 6에서 Fixed-4s와 Fixed-10s는 최저 품질로 하향 조정되지 않고 영상을 재생하였다. 적응형 방식(Adaptive)은 초반(0~25초)에는 대역폭에 적응하고, 중반 이후(25초~)에는 10초 세그먼트를 사용하여 품질을 유지하는 두 가지 양상을 모두 보이며 초기 적응성과 재생 안정성을 보여주었다. 시작 시간은 시나리오 1과 유사한 결과를 보였으며, 대역폭 변동 구간에서 세그먼트를 활용함에 따라 요청 횟수는 증가하였다. Fixed-10s에서 스톨(stall)이 발생한 것과 달리, 적응형 방식은 10초 세그먼트를 사용하는 구간에서도 스톨 발생을 효과적으로 방지하였다.

4) 시나리오 4 실험 결과

표 4와 그림 7은 시나리오 4의 결과를 보여준다. 시나리오 4는 상호작용과 네트워크 상태를 모두 고려해야 하는 가장 복잡한 환경이다. 중반 이후 긴 세그먼트를 활용했던 시나리오 3과 달리, 시나리오 4의 적응형 방식(Adaptive)은 상호작용으로 인해 지속적으로 짧은 세그먼트를 선택하며 대역폭에 적응하였다. Fixed-4s 및 Fixed-10s은 낮은 대역폭과 상

호작용이 겹치는 구간에서 상호작용이 반영되지 않은 기존 프레임을 사용하여 품질을 유지한 것과 다르게, 적응형 방식(Adaptive)은 상호작용이 반영된 프레임들을 대역폭에 맞게 새로 다운받아 평균 비트레이트는 낮게 측정되었다. 시나리오 2와 유사하게 적응형 방식은 낮은 상호작용 응답 지연 시간을 보였다. 여기서 측정된 지연 시간의 절대값이 시나리오 2보다 낮은 이유는, 품질이 낮아 프레임 속도가 높게 유지됨에 따라 버퍼가 더 빠르게 소진되었기 때문이다. 적응형 방식의 시작 시간은 Fixed-1s와 유사하였으며, 시나리오 전반에 걸쳐 주로 1초 세그먼트를 사용함에 따라 HTTP 요청 횟수 또한 Fixed-1s와 비슷한 수준으로 나타났다.

5) 종합 결과

네 가지 시나리오에서의 평가를 통해, 제안하는 적응형 세그먼트 시스템이 재생 안정성과 상호작용 반응성 사이의 균형을 유지할 수 있음을 보여주었다. 상호작용이 없는 안정적(시나리오 1) 및 가변적(시나리오 3) 네트워크 환경에서, 시스템은 안정성과 효율성을 우선시한다. 평가 결과에 따르면, 시스템은 빠른 적응을 위해 짧은 세그먼트를 활용하는 동시에, 안정적인 재생과 요청 오버헤드 감소를 위해 긴 세그먼트를 적절히 혼용한다. 상호작용이 포함된 시나리오 2와 4에서는 짧은 세그먼트와 버퍼 관리를 통해 짧은 상호작용 응답 시간을 보였으며, 동시에 네트워크 상태에 대한 적절한 적응 성능을 보여주었다. 종합적으로, 본 시스템은 각 조건 및 시나리오에서 요구되는 안정성과 반응성 간의 목표 균형을 효과적으로 달성하며 적응성을 보여주었다.

Ⅶ. 논 의

본 연구는 적응형 세그먼트 시스템의 잠재력을 보여주었지만, 몇 가지 한계점이 존재한다. 우선, 본 연구에서는 제안 기법의 효과에 집중하기 위해 단순 대역폭 기반 ABR 알고리즘을 사용하였으나, 실제 환경에서 널리 쓰이는 버퍼 기반(BOLA [29])이나 하이브리드(MPC [30]), 혹은 딥러닝 기반(Pensieve [31]) 알고리즘과의 상호작용 및 최적화가 추가로 검증되어야 한다. 또한 클라이언트 측의 디코딩 및 렌더링 부하가 전체 파이프라인에 미치는 영향을 최소화하기 위해 스레드 및 멀티프로세싱 등을 사용하였지만, 여전히 일부 처리에 따른 병목 현상과 지연 문제가 남아있다. 기기의 실제 성능 상태를 ABR과 세그먼트 선택 과정에 실시간으로 피드백하는 구조가 동반되어야 한다 [28]. 마지막으로, 성능 지표 중심의 평가를 넘어 실제 사용자 체감을 반영하는 QoE 평가가 수반되어야 한다.

향후 연구에서는 이러한 한계를 극복하기 위해 다양한 접근이 가능하다. 현재는 세그먼트 길이 제어 로직이 ABR 알고리즘과 분리되어 동작하지만, 다양한 ABR 알고리즘의 통합 및 분석을 통해 서로 간의 상호작용 및 영향을 정밀하게 분석하고, 이와 함께 프레임 속도(FPS) 및 디코딩 시간 등 기기 성능 상태까지 활용하는 통합 알고리즘으로 발전시킬 수 있다. 특히 현재의 규칙 기반(rule-based) 세그먼트 선택 방식을 개선하여, 보다 정교한 QoE 공식을 실시간으로 반영하는 최적화 모델이나 인공지능 기반의 지능형 적응 모델을 도입하는 것도 한 가지 연구 방향이다. 아울러 실제 네트워크 대역폭 트레이스 및 콘텐츠를 적용하거나 인코딩 과정을 포함한 라이브 스트리밍 시나리오로 범위를 확장함으로써, 더욱 실질적인 몰입 경험을 제공하는 시스템으로 고도화할 수 있을 것이다.

Ⅷ. 결 론

본 연구는 DASH 시스템에서 볼류메트릭 영상의 새로운 요구 사항을 해결하기 위해 세그먼트 길이를 적응적으로 조절하는 DASH 기반 볼류메트릭 영상 스트리밍 시스템을 제안하였다. 주로 2D 영상을 위해 설계된 기존의 고정 길이 세그먼트 DASH 시스템은 볼류메트릭 스트리밍에 필요한 안정성과 반응성 간의 균형을 효과적으로 유지하는 데 어려움이 있었다. 이러한 적응을 구현하기 위해, 우선 클라이언트 측 세그먼트 적응을 지원하는 MPD 구조를 정의하였다. 이어 제안하는 세그먼트 적응 엔진은 MPD 파일에 정의된 세그먼트 길이 옵션을 활용하여 사용자의 상호작용, 네트워크, 버퍼 상태를 기반으로 최적의 세그먼트를 선택하며, 이와 더불어 적절한 버퍼 관리를 수행한다. 서로 다른 네트워크 환경과 상호작용 조건을 갖춘 네 가지 시나리오를 통해 시스템을 평가하였으며, 각 시나리오에서 시스템이 상황에 맞게 반응성과 안정성을 효과적으로 조정하며 적응적으로 동작하는 것을 확인하였다. 향후 연구로는 고도화된 적응형 비트레이트 알고리즘, 기기 성능 피드백 및 주관적 품질 평가와 더불어 더욱 실용적인 시스템을 위한 종단 간 라이브 스트리밍 환경으로의 확장을 제시하였다.

References

[1] T. Stockhammer, "Dynamic adaptive streaming over HTTP-- standards and design principles." Proceedings of the second annual ACM conference on Multimedia systems. 2011.

[2] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)." 2011 IEEE international conference on robotics and automation. IEEE, 2011.

[3] Draco 3D Data Compression, Retrieved June., 25, 2026, https://google.github.io/draco.

[4] MPEG V-PCC, Retrieved June., 25, 2026, https://www.iso.org/standard/83535.html.

[5] B. Han et al., "ViVo: Visibility-aware mobile volumetric video streaming." In Proceedings of the 26th annual international conference on mobile computing and networking (pp. 1–13). 2020.

[6] K. Lee et al., "GROOT: a real-time streaming system of high-fidelity volumetric videos." In Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (pp. 1–14). 2020.

[7] A. Zhang et al., "YuZu:Neural-Enhanced volumetric video streaming." 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). 2022.

[8] C. Wang et al., "Volut: Efficient volumetric streaming enhanced by lut-based super-resolution." Proceedings of Machine Learning and Systems 7 (2025).

[9] Y. Liu et al., "Muv2: Scaling up multi-user mobile volumetric video streaming via content hybridization and sharing." Proceedings of the 30th Annual International Conference on Mobile Computing and Networking. 2024.

[10] R. Pantos and W. May, HTTP live streaming. No. rfc8216. 2017.

[11] LL-DASH, Retrieved June., 25, 2026, https://dashif.org/news/low-latency-dash.

[12] WebRTC, Retrieved June., 25, 2026, https://webrtc.org.

[13] J. Van Der Hooft et al., "Towards 6dof http adaptive streaming through point cloud compression." Proceedings of the 27th ACM International Conference on Multimedia. 2019.

[14] M. Hosseini and C. Timmerer, "Dynamic adaptive point cloud streaming." Proceedings of the 23rd Packet Video Workshop. 2018.

[15] H. Heidarirad and M. Wang, "VV-DASH: A Framework for Volumetric Video DASH Streaming." Proceedings of the 16th ACM Multimedia Systems Conference. 2025.

[16] M. De Fré et al., "Scalable mdc-based volumetric video delivery for real-time one-to-many webrtc conferencing." Proceedings of the 15th ACM multimedia systems conference. 2024.

[17] M. De Fré et al., "Demonstrating adaptive many-to-many immersive teleconferencing for volumetric video." Proceedings of the 15th ACM multimedia systems conference. 2024.

[18] S. Schwarzmann et al., "Comparing fixed and variable segment durations for adaptive video streaming: A holistic analysis." Proceedings of the 11th ACM Multimedia Systems Conference. 2020.

[19] J. van der Hooft et al., "Low-latency delivery of news-based video content." Proceedings of the 9th ACM Multimedia Systems Conference. 2018.

[20] I. Irondi et al., "Optimized adaptation algorithm for HEVC/H. 265 dynamic adaptive streaming over HTTP using variable segment duration." Real-Time Image and Video Processing 2016. Vol. 9897. SPIE, 2016.

[21] B. J. Villa and P. E. Heegaard, "Group based traffic shaping for adaptive HTTP video streaming by segment duration control." 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2013.

[22] The Go Programming Language, Retrieved June., 25, 2026, https://go.dev/

[23] C. Loop et al., Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset, Retrieved June., 25, 2026, https://plenodb.jpeg.org/pc/microsoft. 2016.

[24] Q.-Y. Zhou et al., "Open3D: A modern library for 3D data processing." arXiv preprint arXiv:1801.09847 (2018).

[25] DracoPy, Retrieved June., 25, 2026,

https://github.com/seung-lab/DracoPy.
[26] Async http client/server framework, Retrieved June., 25, 2026, https://github.com/aio-libs/aiohttp.
[27] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment." Linux j 239.2 (2014): 2.
[28] S. Ryu, and J.W.K. Hong, "Analysis of the Impact of Client Computing Capability on ABR Algorithm Behaviors in Volumetric Video Streaming", KNOM Review, Vol. 28, No. 2, Dec. 2025, pp.23-32.
[29] K. Spiteri et al., "BOLA: Near-optimal bitrate adaptation for online videos." IEEE/ACM transactions on networking 28.4 (2020): 1698-1711.
[30] X. Yin et al., "A control-theoretic approach for dynamic adaptive video streaming over HTTP." Proceedings of the 2015 ACM conference on special interest group on data communication. 2015.
[31] H. Mao et al., "Neural adaptive video streaming with pensieve." Proceedings of the conference of the ACM special interest group on data communication. 2017.

유 상 우 (Sangwoo Ryu)



2020 포항공과대학교, 컴퓨터공학과 학사
2020 ~ 포항공과대학교, 인공지능대학원 석/박사 통합과정
<관심분야> 2D/3D 비디오 스트리밍, 영상회의, 인공지능, 멀티미디어

홍 원 기 (James Won-Ki Hong)



2020~현재 포스텍 크립토블록체인연구센터 공동센터장
2019-2023 포스텍 교육혁신센터장
2012~2014 KT 종합기술원 원장
2008~2012 포스텍 정보전자융합공학부장
2008~2010 포스텍 컴퓨터공학과 주임교수
2007~2010 포스텍 정보통신연구소 연구소장
2007~2011 포스텍 정보통신대학원장
1995~현재 포스텍 컴퓨터공학과 교수
1992~1995 Univ. of Western Ontario, 연구교수
<관심분야> AI 기반 네트워크관리, SDN/NFV, 무크기반 교육, 블록체인 및 암호화폐, Vmeeting 화상회의시스템

KNOM Review '26-01 Vol.29 No.01
https://doi.org/10.22670/knom.2026.29.1.24

자율 Kubernetes 관리를 위한 에이전트형 AI: LLM 기반 의사결정 사이클을 활용한 멀티 에이전트 아키텍처

Agentic AI for Autonomous Kubernetes Management: A Multi-Agent Architecture with LLM-Driven Decision Cycles

Kiran Batool[•], Wang-Cheol Song[°]

요 약

컨테이너 기반 클라우드 컴퓨팅의 도입으로 기업 인프라의 운영 복잡성이 증가하고 있다. 쿠버네티스(Kubernetes)가 널리 사용되고 있지만, 클러스터 관리는 여전히 인간의 전문 지식과 정적·임계값 기반 자동화에 의존하고 있다. 본 논문에서는 자율적인 쿠버네티스 클러스터 관리를 위해 대규모 언어 모델(LLM) 기반 의사결정 엔진과 4개의 에이전트로 구성된 에이전트형 인공지능(Agentic AI) 시스템을 제안한다. 제안 시스템은 모니터링, 결함 탐지, 스케일링 및 자원 최적화 기능을 통합하여 지속적인 상황 인식과 상호 조율된 자율적 조치를 수행한다. 세 가지 통제된 시나리오를 통해 시스템을 평가했다. 동일한 조건에서 AKMF를 비활성화한 경우와 활성화한 경우를 비교한 결과, 파드 복구 시간에는 통계적으로 유의미한 차이가 없었다(AKMF 비활성화 평균 3.49~4.02초, AKMF 활성화 평균 3.48~3.75초; n=10, 2회 독립 실행). 이는 본 프레임워크가 단순히 복구 시간을 단축하기보다는 의사결정 과정을 감사하고 설명하는 데 더 큰 가치를 제공함을 보여준다. 또한 워크로드가 48% 증가한 상황에서도 주기 중단 없이 3,200회 이상의 자율적 의사결정을 수행하면서 CPU 145 millicore와 메모리 256 MiB의 일정한 자원 오버헤드를 유지했다. 이러한 결과는 LLM 기반 자율 오케스트레이션이 기존의 반응형 클러스터 관리 방식을 대체할 수 있는 실현 가능한 대안임을 보여준다.

Key Words : AgenticAI, Autonomous Orchestration, Kubernetes, LLM, Container Management

ABSTRACT

Enterprise infrastructure is increasingly complex with container-based cloud computing. Although Kubernetes is widely used, its management still relies on human expertise and static, threshold-based automation. We propose an Agentic AI system using an LLM-based decision engine with four agents for monitoring, fault detection, scaling, and resource optimization. Three controlled scenarios were evaluated. Results showed no statistically significant difference in pod recovery time between AKMF-disabled (3.49 – 4.02s) and AKMF-enabled (3.48 – 3.75s) conditions (n=10, two independent runs), highlighting its value for auditing and explaining decisions rather than reducing recovery time. With a 48% workload increase, the system maintained 145 millicores CPU and 256 MiB memory overhead while handling over 3,200 autonomous decisions without cycle disruptions. These results demonstrate the feasibility of LLM-driven autonomous orchestration as an alternative to reactive Kubernetes management.

※ This work was supported by the 2026 education, research and student guidance grant funded by Jeju National University
• First Author : Jeju National University Department of Electronic Engineering, kiranbatool224@gmail.com
° Corresponding Author : Jeju National University Department of Computer Engineering, philo@jejunu.ac.kr

논문번호 : KNOM2026-01-006, Received July 24, 2026; Revised August 10, 2026; Accepted August 18, 2026

Ⅰ. Introduction

Kubernetes is now at the core of enterprise infrastructure due to the growing use of microservices architecture. Over 73% of businesses currently use Kubernetes in production settings, according to the Cloud Native Computing Foundation [1]. However, its acceptance comes with a significant operational complication. Performance tuning, incident response, and manual troubleshooting typically account for more than 40% of engineering team's working hours [2]. The cognitive burden placed on human operators increases with cluster sizes and service interdependencies to the point where errors are unavoidable and recovery times become needlessly long.

Current automation primitives most notably the Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), and Cluster Autoscaler use threshold-based rules to manage certain operational duties. When predetermined limits are exceeded, HPA adds or removes pod replicas while monitoring CPU and memory usage. However, by default, it examines metrics every 15 to 30 seconds, which means that before any remedial action is taken, a cluster may already have seriously deteriorated [3]. Although VPA modifies resource allocations, it needs pod restarts, which causes service interruptions. Neither tool can reason about application semantics, coordinate across failure domains or react to new situations outside of its encoded rule sets.

An entirely new operational paradigm has been made possible by the rise of autonomous agents driven by large language models (LLMs). Agentic AI systems can perceive the status of the environment, reason about underlying causes, choose right actions, and conduct those actions without human guidance, in contrast to traditional machine learning approaches that can forecast anomalies but depend on humans to conduct responses [4]. This paradigm is no longer theoretical as shown by recent industry deployments in telecom infrastructure, such as production agentic systems for radio access network optimization [5].Today autonomous Kubernetes administration is possible due to a number of supporting variables coming together. First, robust reasoning over structured system state is proved by frontier LLMs like Claude Sonnet. Second, well-defined interfaces for tool invocation and multi-agent coordination are provided via standardized inter-agent communication protocols, such as the Agent2Agent Protocol set up by Google in 2025 and the Model Context Protocol published by Anthropic in 2024. Third, programmatic examination and manipulation of all cluster resources is made possible by Kubernetes's rich and stable API surface.

The design, implementation, and empirical assessment of an Agentic Kubernetes Management Framework (AKMF) that takes advantage of these features are presented in this study.

The following are the contributions to this work:

1. An autonomous architecture with four agents that combines resource optimization, defect detection, monitoring, and auto-scaling into a single decision loop.
2. An LLM-powered decision engine that runs continuously for over 3,200 cycles without human supervision, reaching an autonomous cycle time of 2.34 seconds.
3. A careful, repeatable comparison of pod recovery under the two conditions, with AKMF disabled and enabled, to determine the actual measurable contribution of the framework to the cluster's self-healing capability.
4. A systematic comparison based on peer-reviewed literature between measured system performance and reported HPA behavior baselines.

The rest of this paper is structured as follows. In Section II, relevant research in autonomous networking and Kubernetes automation is reviewed. The suggested architecture is explained in Section III. The experimental setup is described in Section IV. Results and an analytical discussion are presented in Section V. The paper is concluded in Section VI, which also suggests future research areas.

Ⅱ. Related Work

2.1 Threshold-Based Kubernetes Automation

The Horizontal Pod Autoscaler which runs on a proportionate control loop over user-defined metric aims is the standard Kubernetes scaling mechanism [3]. The fundamental design principles of Kubernetes scheduling and resource management are explained by Burns et al., who trace architectural choices back to Google's internal cluster management systems, Borg and Omega [6]. These solutions are reactive by nature and lack application-level context, dependency graphs, and failure propagation models, even while they offer dependable baseline automation for steady-state workloads.

When compared to static HPA designs, Rossi et al.'s combination use of horizontal and vertical scaling employing reinforcement learning showed better resource efficiency [7]. Nevertheless, their method does not adapt to new failure patterns and causes a great deal of environment-specific training. In a similar vein, Toka and Dobrowiecki suggested adaptive pod scaling techniques that were assessed in an IEEE INFOCOM workshop setting and reported significant scaling latency reductions in comparison to traditional HPA [8]. These studies show the well-established ceiling effects of merely reactive tactics and inspire more clever solutions.

2.2 Machine Learning in Network and Infrastructure Management

An extensive overview of machine learning applications in computer networking, including anomaly detection, traffic classification, routing optimization, and quality-of-service management is given by Boutaba et al. [9]. Their survey highlights a recurring conclusion although supervised and reinforcement learning techniques have good predictive power, they usually need human interaction to convert predictions into practical actions, which restricts their use in zero-touch management situations. Autonomous resource management has drawn more attention in the context of Network Function Virtualization and Software-Defined Networking.

An intent-based autonomous network management framework that uses neural reasoning modules to convert high-level operator intentions into low-level configuration actions was presented by Kim et al. in the Journal of Communications and Networks [10]. The application of intent-driven autonomy to container orchestration, the topic of this study is directly motivated by their work. Furthermore, as shown by recent edge-native Kubernetes deployments, the convergence of edge computing and containerization needs that management frameworks function with low latency and without continuous cloud connectivity [11].

2.3 Agentic AI and Autonomous Systems

A comprehensive overview of LLM-based autonomous agents is provided by Wang et al. [4], who describe agent architecture in terms of perception, memory, planning and action modules. Their taxonomy identifies infrastructure management as a high-value domain for agentic systems, given its structured state space, well-defined action boundaries, and rich feedback signals. The ReAct architecture by Yao et al. [12], which interleaves chain-of-thought reasoning with action execution, further reinforces this relevance for closed-loop infrastructure control. Building on these foundations, [13] recently demonstrated autonomous self-healing and scaling for PostgreSQL databases in multi-zone Kubernetes environments. However, their approach remains constrained to a single service domain. The present study advances this direction by introducing a multi-agent LLM-driven framework that reasons across diverse workload types, delivering empirical validation of cluster-wide autonomous orchestration that has been largely absent from prior agentic infrastructure research.

Ⅲ. Proposed Architecture

3.1 Framework Overview

Figure 1 shows how the Agentic Kubernetes Management Framework (AKMF) is structured around five functional layers that provide a closed operational loop. Evaluation scenarios are defined at the outermost layer. Four specialized agents are housed within the Agentic System, continuously collecting and analyzing cluster state data. The LLM Decision Engine combines their output and produces structured action recommendations.The Autonomous Action Executor receives these recommendations and, without human intervention, translates them into one or more Kubernetes API calls. Lastly, a Performance Metrics layer continuously measures system behavior and sends data back to the agents.

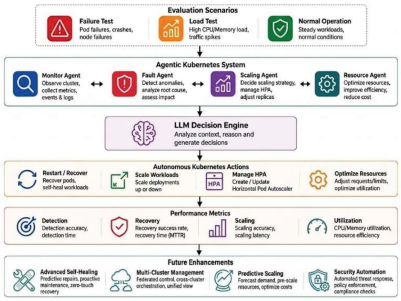


Fig. 1. Proposed Agentic Kubernetes Framework for Autonomous Monitoring, Decision-Making, and Resource Management.

3.2 Four Specialized Agents

To gather pod status, node conditions, cluster events, and container logs, the Monitoring Agent constantly asks the Kubernetes API Server. Additionally, it obtains CPU and memory information in real time from the Kubernetes information Server. After receiving this data stream, the Fault Detection Agent uses anomaly identification logic to determine the blast radius, evaluate the root cause, and pinpoint workload failures. The Scaling Agent assesses if the establishment of a Horizontal Pod Autoscaler, replica adjustment, or deployment modification is necessary by comparing workload demand signals to defined thresholds. To minimize resource waste without compromising headroom, the Resource Optimization Agent evaluates usage trends across pods and nodes and generates recommendations for appropriately sized resource requests and limits.

3.3 LLM Decision Engine

Through the Anthropic API, the four agents send their observations, event summaries, and potential courses of action to the Claude Sonnet 3.5-powered LLM Decision Engine. After receiving structured cluster state context, the engine thoroughly examines the operational state and formulates action recommendations through a structured natural language reasoning chain.

The decision record and this chain of reasoning are stored in a PostgreSQL database, allowing for complete auditability. This auditability was directly exercised during controlled validation testing, where the decision log was queried to confirm whether AKMF's decision engine engaged with specific cluster events demonstrating the log's function as a precise attribution mechanism, not merely a design feature .The Autonomous Action Executor then receives the choice and, without the need for human intervention, transforms it into one or more Kubernetes API calls.

3.4 Safety and Governance

Every autonomous action works in a limited action space that forbids irreversible damaging acts without explicit operator pre-authorization like node deletion or namespace termination. Post-hoc analysis of each decision, its context, the chain of reasoning, and its execution result is made possible by the PostgreSQL audit log. By fulfilling the interpretability requirements that have been highlighted as crucial for the production deployment of agentic infrastructure systems, this governance structure guarantees that autonomous activity is still transparent and accountable.

Ⅳ. Experimental Setup

4.1 Infrastructure Configuration

To enforce resource isolation, the AKMF was installed on a Kubernetes cluster running version 1.28.4, hosted within a specific kube-agents namespace. A CPU request of 200 millicores and a CPU limit of 500 millicores were set up for the agentic application container. Supporting services included a Prometheus and Grafana monitoring stack installed via the kube-prometheus-stack Helm chart for cluster metric collecting and visualization, and a PostgreSQL instance deployed via the Bitnami Helm chart for persistent decision recording. To preserve security isolation, a credential kept as a Kubernetes Secret was used to access the Anthropic API.

4.2 Evaluation Scenarios

To assess different system capabilities, three scenarios were created. To assess baseline monitoring continuity and decision cycle throughput, Normal Operation presents the cluster under stable workload conditions. The Failure Test evaluates fault detection latency and autonomous recovery time by purposefully dropping a pod. The Load Test verifies HPA configuration accuracy and quick scale-out performance by scaling a typical deployment to 10 copies. In a single continuous deployment session, all three scenarios were conducted one after the other.

4.3 Metrics Collection

Four dimensions were used to evaluate performance. Detection metrics measure the system's capacity to recognize workloads and failure occurrences, specifically the time elapsed between a fault occurrence and the initiation of a recovery action. Recovery metrics track the amount of time that passes between the introduction of a problem and the restoration of running state. The time between the issuance of a scale command and the full availability of replicas is captured by scaling metrics. Use metrics, which are gathered at each decision cycle milestone and expressed as CPU usage in millicores and memory consumption in mebibytes, quantify the resource footprint of the agentic framework itself.

Ⅴ. Results and Discussion

5.1 Scenario Validation

Table 1 summarizes the successful outcomes of all three evaluation situations. Throughout every stage, the framework supported autonomous decision execution and ongoing cluster monitoring without the need for human interaction.

Table 1. Evaluation Scenario Results

Scenario	Purpose	Outcome
Normal Operation	Continuous workload monitoring	Successful
Failure Test	Fault detection and recovery	Successful
Load Test	Auto-scaling under demand	Successful

5.2 Agent Initialization and Connectivity

After successfully initializing, all four agents have proven confirmed communication with each of the necessary subsystems. The agent connectivity verification findings derived from deployment logs are shown in Table 2.

Table 2. Agent Initialization and Connectivity Verification.

Verification Test	Measured Output	Result
Kubernetes API Connectivity	23 pods found	Pass
Pod Metrics Retrivel	145m CPU, 256Mi	Pass
LLM API(Claude Sonnect)	Ready, all cycles	Pass
Decision Cycle Time	2.34s Consistent	Pass
Health Endpoint/Health	HTTP 200 OK	Pass
PostgreSQL Database	Connected	Pass
Kubernetes Metrics Server	Accessible	Pass

5.3 Decision Cycle Progression

From Cycle 2 until Cycle 3,207, when the dashboard was finally verified, the LLM Decision Engine ran continuously, making 3,205 independent judgements. At each significant evaluation milestone, Table 3 documents the precise decision cycle numbers and associated cluster status, which are taken straight from agent logs.

Table 3. Decision Cycle Progression Across All Test Milestones

Test Phase	Cycle #	Pods	CPU avg	Mem avg
Initial startup	2	23	145m	256Mi
Log export	6	23	145m	256Mi
Operation	2573 - 2576	31	145m	256Mi
Post-failure	2937 - 2938	32	145m	256Mi
Load test	2974 - 2977	34	145m	256Mi
Final check	3151→3207	31	145m	256Mi

Despite a 48 percent growth in monitoredworkloads, resource consumption remained flat ,suggesting stable resource behavior across the tested pod range though larger deployments are needed to confirm this conclusively. This dual-axis relationship is seen in Fig. 2.

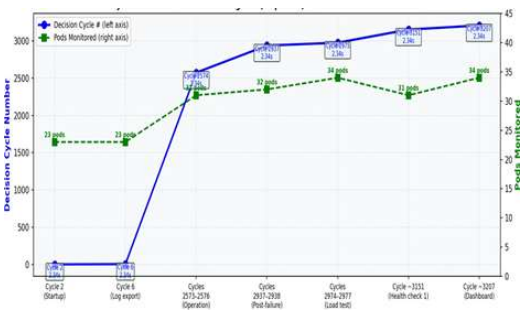


Fig. 2. Decision Cycle Progression and Pod Monitoring Growth Across Evaluation Period.

5.4 Autonomous Actions: Fault Recovery

To understand what role AKMF actually plays in pod recovery, we ran a direct comparison with the same deployment and the same failure method. In the first round of testing (10 trials), the average recovery time was 3.49 seconds (without AKMF) and 3.48 seconds (with AKMF); in the second round of testing (10 trials), it was 4.02 seconds (without AKMF) and 3.75 seconds (with AKMF). Both rounds showed no statistically meaningful difference between the two conditions. A review of AKMF's own decision log during these time windows revealed no recorded decisions whatsoever, despite the system being fully connected and operating normally throughout. The most likely explanation is that native Kubernetes recovery completes faster than AKMF's monitoring cycle can detect a problem and take action.

This finding is notable inlight of the ~12-second recovery figure reported in the original submission,which was based on a single trial (n=1). The repeated-trial results reportedhere (n=10 per condition, two independent runs) are considered more reliableand representative of AKMF's actual pod-recovery behavior, as shown in Fig. 3.

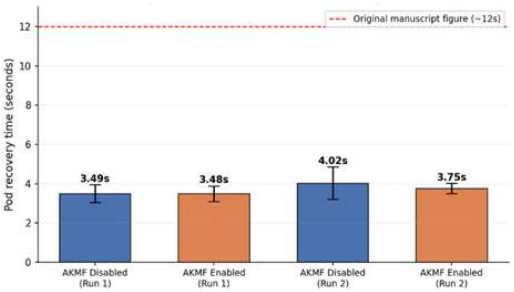


Fig. 3. Pod-crash recovery time: AKMF-disabled vs. AKMF-enabled (n=10/condition, tworuns).

5.5 Auto-Scaling Behavior: HPA-AKMF Attribution

In order to test if it could scale on its own(Fig 4), without having to type a scaling command at any time, we created real traffic against the target deployment and observed the results CPU

consumption rose from a 0% baseline to approximately 190% of the configured request well above the 50 percent target and the deployment increased on its own, from 2 replicas to 4 replicas and then 4 to 8 replicas. After the load ceased it reduced itself, without our help, back down to 2 replicas again. This entire process was done by Kubernetes' own Horizontal Pod Autoscaler and not by AKMF, as can be seen in the audit log that had no corresponding event from AKMF during this time, and as we saw in the recovery test.

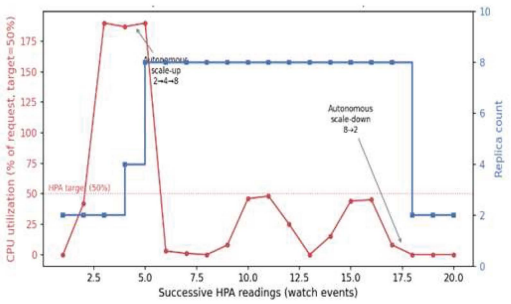


Fig. 4. Autonomous HPA scaling under real HTTP load: CPU utilization and replica count (2→4→8→2) with no manual intervention.

5.6 Component Health

Health status assessment (Fig 6) of the Prometheus monitoring stack revealed that 13 of 17 watched components were operational (76.47 percent health rate), while 4 components remained unavailable. The Monitoring Agent's core metrics pipeline was unaffected by the missing components, which were limited to supplementary Prometheus exporters.

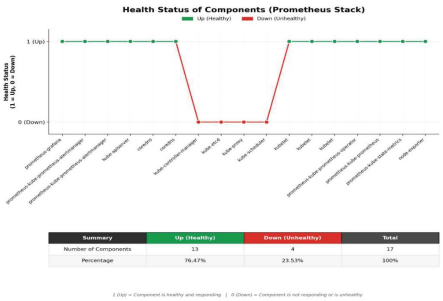


Fig. 5. Component health monitoring results.

5.7 Resource Utilization

At every cycle milestone, the Monitor Agent recorded the AKMF container's CPU and memory usage. Despite the 48 percent increase in checked workloads , both indicators remained entirely flat throughout all five measurement points, as seen in Fig. 5. Memory remained at 256 MiB throughout, and CPU use remained at 145 milli cores, or 29% of the specified 500 milli core maximum. This constant resource usage is a promising sign, though testing on larger clusters would be needed to confirm the pattern holds at scale.

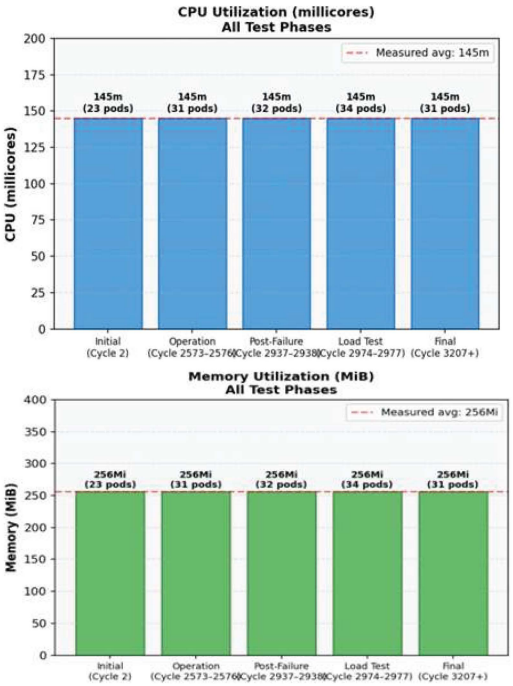


Fig. 6. CPU and Memory Utilization of the AKMF Framework Across All Test Phases.

5.8 Monitoring Dashboard

The final verification attests to the fact that every framework component continued to function during the assessment period. Stable communication with all supporting services allowed for the execution of over 3,200 autonomous choices. These findings confirm that the suggested Agentic Kubernetes framework for autonomous cluster management is dependable, scalable, and useful.



Fig. 7. Agentic Kubernetes Monitoring Dashboard: Live System Status and Connectivity Overview.

5.9 Comparison with Traditional HPA

Table 4 is a comparison of AKMF to the traditional Kubernetes HPA behavior. The reference values for HPA (polling interval, scale stabilization) were taken from official Kubernetes documentation [3] and the decision-cycle timing and monitoring characteristics of AKMF are based on its measurements. Pod recovery is based on a controlled experiment using AKMF with and without AKMF (all other conditions the same: n=10 trials/condition, 2 runs), so that it can be directly attributed without relying on external literature. The observed recovery behavior was primarily driven by native Kubernetes reconciliation (not AKMF's decision engine) and no statistically significant difference in recovery time was found between conditions (p>0.3).

Note: The values for polling interval and scale-stabilization are taken from the Kubernetes official documentation [3] for the Traditional HPA. In this study the direct measurement of pod-recovery values for AKMF was performed under the controlled, same cluster, same conditions testing (AKMF-disabled vs. AKMF-enabled). Polling interval and AKMF decision-cycle time are of different type of metrics and cannot be compared directly as speed.

Table 4. Comparison of Traditional HPA and AKMF

Metrics	Traditional HPA	AKMF	Ref.
Polling interval	15s	2.34s per cycle	[3]
Scale stabilization	5 min window	Real-time event	[3]
Monitoring scope	CPU/Memory only	API + DB + LLM + Metrics	This study
LLM reasoning	Not supported	Claude Sonnet	This study
Audit logging	Not supported	PostgreSQL full log	This study
Autonomous decisions	Rule-Based only	3,207+ monitoring cycles executed	This study
Pod recovery – AKMF-disabled	–	3.49s - 4.02s (n=10/run, 2 runs)	This study
Pod recovery – AKMF-enabled	–	3.48s - 3.75s; no significant difference disabled (p>0.3)	This study

5.10 Limitations and Scope

This evaluation is a proof-of-concept deployment. The pod scale ranged from 23 to 34 pods during testing which will have to be verified at the production cluster scale of several thousand pods. More trials need to be performed in future research to obtain a statistical picture of variation, as there were repeated trials (n=10/condition, two runs) for the failure scenario, but a single trial for the load/scaling scenario, with means and ranges presented here, and standard deviations and failure rates from a larger number of trials for a fuller statistical picture. This evaluation also does not measure the LLM's decision-cycle latency, cost of tokens, or API cost, .nor does it describe the structure of prompts and the output-validation

protections the LLM Decision Engine uses.
Documenting all of this would be a necessary future task, especially to determine the cost effectiveness at scale. During testing reported metrics were not affected by the four Prometheus exporters being temporarily unavailable, but subsequent installations should ensure that the whole monitoring-stack is healthy. Finally, this study only examined one type of failure (pod deletion) and one type of load (HTTP traffic), it would be useful to see how many different types of failure and load could be tested to determine the generality of these findings.

Ⅵ. Conclusion

The aim of this research is to create and test an Agentic Kubernetes Management Framework with an LLM-powered decision engine comprised of four separate agents. After an initial evaluation, a controlled comparison was conducted which showed no statistically significant difference in pod-recovery time between AKMF-enabled and AKMF-disabled conditions (3.48s – 3.75s vs. 3.49s – 4.02s, n=10 per condition, two independent runs), and demonstrated the true scaling. The framework consistent operations across three evaluation scenarios for over 3,207 decision cycles at a steady cycle time of 2.34 seconds, with a constant overhead of 145 millicores CPU and 256 MiB memory despite a 48% workload increase though this will need to be validated at production scale before drawing strong conclusions. And also, future work will quantify the LLM's contribution to decision latency and operating cost, add repeated-trial statistical validation at larger cluster scale, and evaluate a broader range of failure and load conditions, as detailed in Section 5.10.
In addition, evaluation will be expanded to production-scale clusters of 500 – 5,000 pods, network latency and traffic-engineering agents will be integrated into the multi-agent architecture, formal governance protocols will be developed for safe production rollout, and an open-source implementation will be made available to enable community validation and expansion of this work.

References

[1] Cloud Native Computing Foundation (CNCF), "CNCF Annual Survey 2024," CNCF Technical Report, 2024. [Online]. Available: CNCF Annual Survey 2024

[2] G. Kim, P. Debois, J. Willis, and J. Humble, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland, OR, USA: IT Revolution Press, 2016.

[3] Kubernetes Authors, "Horizontal Pod Autoscaling," Kubernetes Documentation, 2024. [Online]. Available: Horizontal Pod Autoscaling

[4] A. Wang, B. Chen, C. Liang, F. Wu, X. Wang, and Z. Chen, "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, p. 186345, 2024, doi: 10.1007/s11704-024-40231-1.

[5] Rakuten Mobile and Rakuten Symphony, "Rakuten Mobile and Rakuten Symphony Deploy Intelligent AI-powered RIC Platform in Rakuten's 4G and 5G Open RAN Network in Japan," Press Release, May 2025. [Online]. Available: Rakuten Mobile and Rakuten Symphony press release

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade," ACM Queue, vol. 14, pp. 70–93, 2016, doi: 10.1145/2898442.2898444.

[7] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in Proc. IEEE International Conference on Cloud Computing (CLOUD), Milan, Italy, 2019, pp. 35–45, doi: 10.1109/CLOUD.2019.00020.

[8] L. Toka and O. Dobrowiecki, "Adaptive scaling of Kubernetes pods," in Proc. IEEE INFOCOM 2021 Workshops Vancouver, BC, Canada, 2021, pp. 1–6, doi: 10.1109/INFOCOMWKSHPS51825.2021.9484516.

[9] R. Boutaba, M. A. Salahuddin, N. Limam, S.

Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," Journal of Internet Services and Applications, vol. 9, no. 1, p. 16, Jun. 2018, doi: 10.1186/s13174-018-0087-2.

[10] J. Kim, S. Lee, and H. Park, "Intent-based autonomous network management using neural reasoning for software-defined infrastructure," Journal of Communications and Networks, vol. 25, no. 3, pp. 312–325, Jun. 2023, doi: 10.23919/JCN.2023.000021

[11] M. Rosendo, P. Costan, G. Antoniu, and K. Bhattacharya, "Distributed intelligence on the edge-to-cloud continuum: A systematic literature review," Journal of Parallel and Distributed Computing, vol. 166, pp. 71–94, Aug. 2022, doi: 10.1016/j.jpdc.2022.04.004.

[12] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in Proc. International Conference on Learning Representations (ICLR), Kigali, Rwanda, 2023. [Online]. Available: arXiv:2210.03629

[13] T. Amin, M. Asif, U. Mahmood, and W. Song, "Autonomous PostgreSQL operator for self-healing and scaling in multi-zone Kubernetes environments," KNOM Review, vol. 28, no. 2, pp. 67–77, Dec. 2025.

Kiran Batool



2021 September: B.S. in Information Technology, Government College University Faisalabad, Pakistan.

2024 September: M.S. in Information Technology, National University of Sciences and Technology (NUST), Pakistan.

2026 March – Present: Ph.D. in Electronic Engineering, Jeju National University, Republic of Korea.

<Field of Interest>: Agentic AI and LLMs, Autonomous and Intent-Based Systems, Cloud-Native Computing and Kubernetes, Software-Defined Networking (SDN), and Internet of Things (IoT).

Wang-Cheol Song



1986 February: B.S in Food Engineering, Yonsei University, Republic of Korea

1989 February: B.S in E lectronic Engineering, Yonsei University, Republic of Korea.

1991 February: M.S in Electronic Engineering, Yonsei University, Republic of Korea.

1995 August: Ph.D. in Electronic Engineering, Yonsei University, Republic of Korea.

<Field of Interest>: SDN/NFV, Intent-based networking, and Network management.

KNOM Review '26-01 Vol.29 No.01
https://doi.org/10.22670/knom.2026.29.1.34

PRISM-2026:엔드포인트 기반 자동 라벨링 다계층 네트워크 트래픽 데이터셋

김 지 민•, 유 경 민*, 장 윤 성**, 남 승 우**, 김 주 성**, 백 의 준°°, 김 명 섭°

PRISM-2026: A Multi-Level Network Traffic Dataset with Endpoint-Based Automatic Labeling

Ji-min Kim•, Gyeong-min Yu*, Yun-Seong Jang**, Seung-Woo Nam**, Ju-Sung Kim**, Ui-Jun Baek°°, Myung-Sup Kim°

요 약

암호화 트래픽의 확산으로 정확하게 라벨링된 고품질 데이터셋 확보가 트래픽 분류 연구의 핵심 과제로 대두되고 있다. 기존 데이터셋은 단일 응용 격리, 시간 구간 분할 등의 방식에 의존하여 실사용 환경의 동시 트래픽을 반영하지 못하거나 라벨 무결성에 한계가 있다. 본 연구는 트래픽을 발생시킨 엔드포인트로부터 라벨을 직접 획득하는 자동 라벨링 프레임워크 PRISM을 활용하여 다계층 네트워크 트래픽 데이터셋 PRISM-2026을 구축·공개한다. PRISM-2026은 7대의 실사용 호스트에서 3일간 수집한 1,320,366개의 세션으로 구성되며, 각 세션에 5-튜플 종단점 정보와 프로세스·서비스·사이트의 3단계 라벨을 부여한다. 특히 svchost.exe에 대한 서비스 단위 라벨링과 SNI 기반 브라우저 사이트 단위 라벨링을 도입하여, 프로세스 단위 라벨이 구분하지 못하던 공유 호스트 프로세스와 브라우저 트래픽을 세분화하고, 프로세스 단위(108종)와 응용 단위(89종)의 두 라벨 집합을 제공한다. 대표적 분류 모델(ET-BERT, TSHIRT, 2D CNN, XGBoost) 기준 실험에서 XGBoost가 최고 정확도(0.917)를 기록하였다.

Key Words : Network Traffic Classification, Traffic Dataset, Automatic Labeling, Endpoint-based Labeling, Encrypted Traffic

ABSTRACT

As encryption becomes pervasive, the availability of accurately labeled, high-quality datasets is a central challenge in network traffic classification. Existing datasets rely on single-application isolation or time-interval segmentation and thus fail to capture the concurrent traffic of real hosts or suffer from limited label integrity. This study builds and releases PRISM-2026, a multi-level network traffic dataset created with PRISM, an automatic labeling framework that obtains labels directly from the endpoints generating the traffic. PRISM-2026 comprises 1,320,366 sessions collected from seven in-use hosts over three days, with each session annotated by five-tuple endpoint information and three-level labels spanning process, service, and site. In particular, service-level labeling for svchost.exe and SNI-based site-level labeling for browsers refine the shared-host-process and browser traffic that process-level labels alone cannot distinguish, yielding two label sets at the process level (108 classes) and application level (89 classes). In baseline experiments with representative classifiers (ET-BERT, TSHIRT, 2D CNN, XGBoost), XGBoost achieved the highest accuracy (0.917).

※ 이 논문은 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구이며(RS-2025-16067207, 양자 네트워크 라우팅 및 제어/관리 핵심 기술과 표준 개발), 2026년도 정부(산업통상부)의 재원으로 한국산업기술진흥원의 지원을 받아 수행된 연구임 (P0031136, 2026년 중부권 지역혁신클러스터육성).
◆ First Author : Korea University Department of Computer Science and Software Engineering, illiard1209@korea.ac.kr
° Corresponding Author : Korea University Department of Computer Science and Software Engineering, tmskim@korea.ac.kr
°° Corresponding Author : Korea Institute of Science and Technology Information (KISTI), pb1069@kisti.re.kr
* Korea University Department of Computer Science and Software Engineering rudals2710@korea.ac.kr
** Korea University Department of Computer Information and Science, {brave1094, nam131119, jsung0514}@korea.ac.kr
논문번호 : KNOM2026-01-007, Received July 25, 2026; Revised August 10, 2026; Accepted August 18, 2026

Ⅰ. 서 론

네트워크 기술이 고도화되면서 이를 안정적으로 관리하고 모니터링하기 위한 기술의 중요성도 함께 커지고 있다. 특히 트래픽 분류는 이상 트래픽 탐지, 사용자 행위 분석 등 다양한 응용으로 확장될 수 있어 네트워크 관리 분야에서 핵심적인 역할을 담당한다.

네트워크 트래픽 분류 기술은 포트 분석[1], DPI 기반 분석[2]을 거쳐 ML/DL 기반 방식[3]으로 발전해왔으며, 포트의 동적 할당과 암호화 트래픽의 증가 등으로 인해 최근에는 ML/DL 기법이 주로 연구되고 있다[4]. 다만 이러한 학습 기반의 모델 성능은 데이터셋의 품질에 의한 성능의 편차가 커서 정확하게 라벨링된 고품질 데이터셋의 확보가 무엇보다 중요하다.

본 연구에서는 네트워크 트래픽 분류 분야에서 사용되는 데이터셋들의 수집 방법 및 품질을 분석하고, 기존 데이터셋들이 지닌 구조적 한계를 극복하기 위한 자동 라벨링 프레임워크 PRISM을 활용한 데이터셋 PRISM-2026을 제안한다. PRISM-2026은 다음과 같은 특성을 갖는다. 첫째, 라벨을 패킷에서 추론하는 대신 트래픽을 발생시킨 엔드포인트로부터 직접 획득하여 무결성을 보장한다. 둘째, 라벨링 단위를 프로세스·서비스·방문 사이트 수준으로 유연하게 확장할 수 있는 확장성을 제공한다. 셋째, 특정 애플리케이션만을 격리해 수집하는 방식이 아니라 실제 호스트에서 동시에 발생하는 트래픽을 그대로 담아 실사용 환경을 반영한다. 넷째, 수집부터 라벨링까지 전 과정을 사람의 개입 없이 완전히 자동화하여 다중 호스트에서 지속적으로 데이터셋을 축적할 수 있다.

Ⅱ. 관련 연구

네트워크 트래픽 분류 연구에서 데이터셋은 모델의 학습과 평가를 좌우하는 핵심 요소이며, 특히 딥러닝 기반 분류의 성능은 라벨의 정확성과 트래픽의 다양성에 크게 의존한다. 본 절에서는 기존 주요 데이터셋을 라벨링 방식에 따라 세 가지로 분류하고 각 방식이 지닌 구조적 한계를 논의한 뒤, 본 연구가 기반으로 하는 선행 프레임워크와의 관계를 설명한다.

2.1. 단일 응용 격리 수집 방식

ISCX-VPN-NonVPN 데이터셋[5]은 목표 응용 하나만 실행하고 나머지 모든 서비스와 응용을 종료한 상태에서 트래픽을 캡처하여, 해당 시간대의 트래픽 전체를 그 응용으로 간주하는 방식을 사용한다. 이 방식은 단순하고 직관적이지만, 복수의 응용이 동시에 실행되는 실제 호스트 환경을 반영하지 못하고, 목표 응용을 격리하더라도 운영체제의 백그라운드 서비스 트래픽이 혼입될 가능성이 있으며, 매 응용마다 수동으로 캡처 세션을 구성해야 하므로 수집 규모와 현실성이 모두 제한된다.

2.2. 시간 구간 분할 기반 라벨링 방식

CIC-IDS2017 데이터셋[6]은 침입 탐지 시스템 평가를 위해 정상 트래픽과 공격 트래픽을 사전에 정의된 시간대에 따라 구분하여 라벨을 부여한다. USTC-TFC2016[7]은 오늘날까지 널리 사용되는 바이트 이미지 기반 합성곱 처리 파이프라인을 정립하였다. 두 데이터셋 모두 시간 또는 캡처 구간을 기준으로 라벨을 부여하므로, 공격 시간대에 혼재하는 정상 트래픽을 정밀하게 구분하기 어렵고, svchost.exe와 같이 하나의 프로세스가 여러 서비스를 실행하는 공유 호스트 트래픽을 세분화할 수 없다는 한계를 지닌다.

2.3. 호스트 기반 프로세스 태깅 방식

호스트 기반 패킷 캡처 도구인 Microsoft Network Monitor는 대상 호스트에서 직접 트래픽을 캡처하면서 각 패킷에 소유 프로세스 정보를 태깅할 수 있어 라벨링에 유리하다. 그러나 이 방식은 단일 호스트에서만 동작하며, 고트래픽 환경에서는 캡처 도구 자체의 CPU 및 디스크 I/O 부하로 인해 저장 자체가 안 되는 문제가 발생한다. 또한 호스트나 캡처 에이전트에 장애가 발생하면 수집이 함께 중단되어 트래픽 수집의 연속성이 보장되지 않는다.

2.4. 선행 프레임워크와 본 연구의 관계

본 연구는 선행 연구에서 제안한 프로세스 수준 자동 라벨링 프레임워크[8]를 기반으로 한다. [8]은 스위치 포트 미러링을 통한 호스트 독립적 트래픽 수집과 Windows 소켓 폴링 및

표 1. 기존 데이터셋과 PRISM-2026의 비교
Table 1. Comparison of Existing Datasets and PRISM-2026

데이터셋	라벨링 방식	동시 실행 반영	라벨 단위	다중 호스트·연속수집
ISCX-VPN-NonVPN	단일 응용 격리	×	응용	×
CIC-IDS2017	시간 구간 분할	×	정상/공격	×
MS Network Monitor	호스트 태깅	○	응용	×
PRISM-2026(제안)	엔드포인트 매칭	○	프로세스/서비스/사이트	○

ETW 보조 수집을 결합하고, 소켓 직접 매칭·규칙 저장소·캐시 계층으로 구성된 다단계 매칭 파이프라인과 SNI 기반 후처리를 통해, 7대 호스트 24시간 연속 수집 환경에서 99.8% 매칭률을 달성하였다. 이로써 앞서 언급한 세 방식의 한계-무결성, 확장성, 수집 연속성-를 상당 부분 해소하였으나, 라벨링 단위를 프로세스로 한정하였다는 제약이 있다. 실제로 [8]에서 전체 세션의 절반 이상을 차지한 svchost.exe와 그 뒤를 이은 브라우저 트래픽은 각각 하나의 프로세스로만 표현되어, 공유 호스트 프로세스 내부의 개별 서비스나 브라우저가 접속한 목적지를 구분하지 못한다. 본 연구는 이 프레임워크 위에서 svchost.exe에 대한 서비스 단위 라벨링과 브라우저에 대한 사이트 단위 라벨링을 도입하여 라벨 granularity를 확장하고, 이를 통해 구축한 데이터셋을 공개한다.

2.5. 데이터셋 비교

앞서 논의한 기존 데이터셋과 PRISM-2026의 특성을 표1에 정리하였다. PRISM은 엔드포인트에서 직접 획득한 라벨을 통해 무결성을 확보하고, 실제 호스트의 동시 실행 트래픽을 반영하며, 프로세스·서비스·사이트의 3단계 라벨 granularity를 지원한다는 점에서 기존 데이터셋과 차별화된다.

Ⅲ. 본 론

3.1. 시스템 개요

본 연구의 데이터셋은 선행 연구에서 제안한 프로세스 수준 자동 라벨링 프레임워크 PRISM을 기반으로 생성되었다. PRISM은 스위치 포트 미러링을 통해 대상 호스트에 부하를 주지 않고 트래픽을 수집하고, 각 Windows 호스트에서 50ms 주기의 소켓 폴링과 ETW(Event Tracing for Windows) 기반 보조 수집을 병행하여 소켓과 프로세스의 대응 관계를 확보한다. 수집된 소켓 정보는 소켓 직접 매칭, 규칙 저장소, 캐시 계층으로 구성된 다단계 매칭 파이프라인과 SNI 기반 후처리를 거쳐 각 세션에 이를 발생시킨 프로세스 라벨을 부여한다. 선행 연구[8]는 7대 호스트 24시간 연속 수집 환경에서 99.8%의 프로세스 매칭률을 보고한 바 있다.

표 2. PRISM-2026 일자별 프로세스 매칭률 및 공개본 구성
Table 2. Daily Process-Matching Rate and Composition of the Released PRISM-2026 Dataset

구분	2026. 07.10	2026. 07.11	2026. 07.12	합계
전체 세션	538,176	458,699	464,257	1,461,132
unknown	5,623	5,507	5,612	16,742
매칭률	98.96%	98.80%	98.79%	98.85%
System	44,022	38,023	40,891	122,936
소수 라벨(<10)	389	363	336	1,088
공개 세션	488,142	414,806	417,418	1,320,366

본 데이터셋 구축 과정에서 측정한 매칭률과 공개본 구성 내역은 표 2와 같다. 3일간 수집된 전체 1,461,132개 세션 가운데 프로세스를 확정하지 못한 세션은 16,742개(1.15%)로, 프로세스 매칭률은 98.85%였다. 매칭에 실패한 세션은 라벨을 임의로 추정하지 않고 unknown으로 분리하였으며, 커널에 귀속되어 특정 응용으로 환원되지 않는 System 트래픽 122,936개(8.41%)와 함께 공개본에서 제외하였다. 여기에 더해 특정 호스트·일자에서 세션 수가 10개 미만으로 관측되어 통계적 의미를 갖기 어려운 라벨의 세션 1,088개(0.07%)를 추가로 제외하였다. 그 결과 공개되는 1,320,366개 세션(전체의 90.37%)은 모두 프로세스 라벨이 확정된 세션으로 구성된다.

매칭 실패 세션은 일자별로 5,507 - 5,623개로 거의 일정하며, 매칭률의 일자 간 차이(98.79 - 98.96%)는 주로 일자별 총 세션 수의 차이에서 기인한다. 본 프레임워크의 규칙 저장소와 캐시 계층은 반복 관측되는 소켓 - 프로세스 대응 패턴을 지속적으로 누적하므로, 수집이 장기화될수록 소켓 직접 매칭에 실패한 세션을 흡수하여 unknown 비율을 낮출 여지가 있다. 다만 3일이라는 관측 구간에서는 그러한 감소 추세가 뚜렷하게 확인되지 않았는데, 이는 규칙과 캐시가 충분히 축적되기에 기간이 짧았거나, 잔여 unknown이 폴링 주기와 ETW 이벤트 모두에서 포착되지 않는 극히 짧은 수명의 세션과 같은 구조적 요인에 기인할 가능성을 시사한다. 수집 기간을 확대하여 규칙 축적에 따른 매칭률의 변화를 관측하는 것은 향후 연구 과제로 남긴다. 한편 여기서의 매칭률은 세션에 라벨이 부여된 비율을 의미하며, 부여된 라벨이 실제와 일치하는 비율, 즉 라벨 정확도와는 구분되는 지표이다. 라벨 정확도에 대한 정량적 검증은 본 연구의 한계로 5.1절에서 논의한다.

그러나 선행 프레임워크는 라벨링 단위를 프로세스로 한정하였다는 제약이 있다. 실제 수집 결과에서 전체 세션의 절반 이상을 차지한 svchost.exe는 다수의 Windows 서비스를 대리 실행하는 공유 호스트 프로세스임에도 하나의 라벨로만 표현되며, 그 뒤를 잇는 브라우저 트래픽 역시 chrome.exe·msedge.exe 등 프로세스 이름으로만 묶여 실제 접속한 목적지를 구분하지 못한다. 이는 동일한 프로세스 라벨 안에 성격이 전혀 다른 트래픽이 혼재함을 의미하며, 세밀한 트래픽 분석을 목표로 하는 데이터셋의 활용도를 제한한다.

본 연구는 이 프레임워크 위에서 라벨 granularity를 확장한다. 구체적으로 (i) svchost.exe에 대해서는 소켓을 실제로 점유한 서비스를 식별하는 서비스 단위 라벨링을, (ii) 브라우저에 대해서는 접속 목적지를 식별하는 사이트 단위 라벨링을 도입한다. 그 결과 최종 라벨은 프로세스·서비스·사이트의 3단계 granularity를 갖게 되며, 수집부터 라벨링·저장까지의 전 과정은 사람의 개입 없이 자동으로 수행되어 다중 호스트에서 지속적으로 데이터셋을 축적한다. 이렇게 구축된 데이터셋을 PRISM-2026로 명명하고 공개한다.

3.2. 프로세스 수준 자동 라벨링 파이프라인

데이터셋 생성의 토대가 되는 파이프라인은 트래픽 수집부, 소켓 수집부, 매칭부, 세션 생성부의 네 단계로 구성되며, 각 단계는 독립 프로세스로 상시 동작한다.

트래픽 수집부는 스위치의 포트 미러링 포트에 연결된 수집 서버에서 미러링된 원본 트래픽을 그대로 받아 1초 주기로 플로우를 생성한다. 대상 호스트에 에이전트 부하나 캡처 손실을 유발하지 않으므로 고트래픽 환경에서도 수집 연속성이 유지된다. 생성된 플로우와 패킷은 5-튜플(프로토콜, 출발지·목적지 IP 및 포트)을 기준으로 세션 단위로 정리된다.

소켓 수집부는 각 Windows 호스트에서 동작하며, 50ms 주기의 소켓 테이블 폴링으로 (5-튜플, PID, 프로세스 경로)의 대응 관계를 지속적으로 수집한다. 짧은 수명의 세션이 폴링 주기 사이에 누락되는 것을 보완하기 위해 ETW 기반 이벤트 수집을 병행하여, 연결 생성·종료 시점의 소켓-프로세스 정보를 함께 확보한다. 수집된 소켓 정보는 수집 서버로 전송되어 매칭부의 입력이 된다.

매칭부는 미러링에서 얻은 세션과 호스트에서 얻은 소켓 정보를 대응시켜 각 세션에 프로세스 라벨을 부여한다. 매칭은 비용이 낮은 순서대로 계층화되어, 먼저 5-튜플과 시간 구간이 정확히 일치하는 소켓을 찾는 소켓 직접 매칭을 시도하고, 실패한 세션에 대해서는 반복적으로 관측되는 패턴을 축적한 규칙 저장소와 최근 매칭 결과를 보관한 캐시 계층을 순차적으로 적용한다. 이 다단계 구조는 폴링 누락이나 시간 오차로 직접 매칭이 되지 않는 세션까지 흡수하여 매칭률을 높인다.

세션 생성부는 라벨이 확정된 세션을 파일 시스템에 물리적으로 정리한다. 이때 세션을 발생시킨 호스트, 부여된 라벨에 따라 디렉토리를 구성하고 각 세션을 개별 pcap 파일로 저장한다. 이 단계의 출력이 곧 PRISM-2026 데이터셋의 원본이며, 이후 3.3 - 3.4에서 설명하는 서비스·사이트 단위 라벨링은 이 파이프라인의 후처리 단계로 통합되어 동작한다.

3.3. svchost.exe 호스트 단위 라벨링

svchost.exe는 여러 Windows 서비스를 하나의 프로세스로 대리 실행하는 공유 호스트 프로세

스로, PID만으로는 어떤 서비스가 해당 소켓을 사용했는지 구분할 수 없다. 따라서 프로세스 단위 매칭 결과가 svchost.exe인 세션은 성격이 서로 다른 다수 서비스의 트래픽이 하나의 라벨에 뭉쳐 있는 상태가 된다.

이를 해소하기 위해 소켓 수집 단계에서 각 소켓을 점유한 서비스를 함께 식별한다. Windows에서 svchost.exe 인스턴스와 그 내부에서 구동되는 서비스의 대응 관계를 조회하여, 소켓의 소유 PID를 프로세스뿐 아니라 개별 서비스까지 해석한다. 그 결과 svchost.exe 라벨은 소켓을 실제로 사용한 서비스 이름으로 세분화되며, 예컨대 원격 데스크톱(TermService), 이름 확인(Dnscache), 백그라운드 전송(BITS), 진단 데이터(DiagTrack), 업데이트 전송 최적화(DoSvc), 인증서 서비스(CryptSvc) 등으로 구분된다.

이 서비스 단위 라벨링을 통해, 기존에 전체 세션의 절반 이상을 차지하며 분석을 저해하던 svchost.exe 트래픽이 서비스별로 분리되어, 각 서비스의 통신 패턴을 독립적으로 관찰할 수 있게 된다.

3.4. 브라우저 사이트 단위 라벨링

브라우저 트래픽은 프로세스 라벨이 chrome.exe·msedge.exe 등으로 동일하더라도 실제로는 서로 다른 다수의 웹 서비스에 대한 접속이 혼재한다. 프로세스 라벨만으로는 이러한 목적지 차이를 구분할 수 없으므로, 브라우저로 매칭된 세션에 대해 접속 사이트를 식별하는 후처리를 추가한다.

사이트 식별은 암호화된 트래픽에서도 평문으로 노출되는 TLS ClientHello의 SNI(Server Name Indication) 필드를 활용한다. 세션의 SNI에서 목적지 도메인을 추출하고, 이를 사전에 정의한 서비스·사이트 매핑에 대응시켜 각 세션을 접속한 사이트 단위로 라벨링한다. 그 결과 동일한 브라우저 프로세스에서 발생한 트래픽이 접속 목적지(예: Google, Microsoft Edge 관련 서비스, KakaoTalk, Notion 등)별로 분리된다. SNI가 존재하지 않거나 매핑되는 목적지를 확정할 수 없는 세션은 별도의 no_visit 그룹으로 분류하여 라벨의 신뢰성을 유지한다.

이 사이트 단위 라벨링은 브라우저라는 단일 프로세스 안에서 사용자가 실제로 소비한 서비스를 구분해내며, 서비스 단위 라벨링과 함께 데이터셋의 라벨 granularity를 프로세스·서비스·사이트의 3단계로 확장한다.

3.5. 디렉토리 구조

PRISM-2026은 동일한 세션 집합을 세 가지 관점으로 제공하며, 라벨 계층을 그대로 파일시스템 경로에 반영하여 별도의 색인 없이 디렉토리 탐색만으로 원하는 단위의 트래픽을 선택할 수 있도록 구성된다. 실제 pcap 파일은 sessions/에만 존재하고, 나머지 두 관점은 이를 가리키는 상대 심볼릭 링크로 제공되어 저장 공간을 추가로 소모하지 않는다.

```
PRISM-2026/
  ├──
sessions/<수집날짜>/<호스트ID>/<프로세스
라벨>.exe/<세션>.pcap
  │
/svchost.exe/<서비스 이름>/<세션>.pcap
  │
/<브라우저>.exe/<사이트 라벨 |
no_visit>/<세션>.pcap
  ├── byproduct/<수집날짜>/<응용 라벨>/…
        (응용 단위, 89종)
  ├──
byproduct_process/<수집날짜>/<프로세스
라벨>/…  (프로세스 단위, 108종)
  └── *.manifest.tsv
        (관점 ↔ sessions 경로 대응표)
```

sessions/ 는 수집 원본 레이아웃으로 호스트 단위 분석에 사용하고, byproduct/ 와 byproduct_process/ 는 호스트를 통합하여 각각 응용 단위·프로세스 단위 분류 실험에 바로 사용할 수 있다. 세 관점의 세션 수는 모두 1,320,366으로 동일하다. 호스트 식별자는 개인정보 보호를 위해 익명화된 값이다.

개별 세션 파일은 5-튜플을 그대로 파일명에 담아, 파일명만으로 방향·종단점을 식별할 수 있도록 한다. 명명 규칙은 다음과 같으며, 동일 5-튜플이 재발생한 경우 캡처 시각(연월일_시분초_밀리초)을 접미로 덧붙여 구분한다.

```
TCP_<출발지IP>_<출발지포트>__<목적지IP>_
<목적지포트>[__<캡처시각>].pcap
```

이와 같은 경로 기반 구성은 호스트별·프로세스별·서비스별·사이트별 트래픽을 즉시 분리·선택할 수 있게 하며, 날짜 단위 파티셔닝은 다중

호스트 연속 수집 결과를 지속적으로 누적·관리하는 데 적합하다. 또한 날짜별로 분리된 구성은 학습·시험 집합을 날짜 기준으로 나누는 평가에도 그대로 활용할 수 있다.

3.6. 라벨 체계 및 세션 필드

PRISM-2026의 라벨은 앞 절에서 설명한 다단계 매칭 및 후처리 파이프라인의 출력으로 확정되며, 프로세스·서비스·사이트의 3단계 granularity를 갖는다. 가장 기본이 되는 프로세스 단위 라벨은 세션을 발생시킨 실행 파일 이름(예: chrome.exe, spotify.exe)으로 부여되고, svchost.exe에 대해서는 소켓을 실제로 점유한 서비스 이름으로, 브라우저에 대해서는 SNI로 식별한 접속 사이트 이름으로 세분화된다. 본 데이터셋은 이러한 라벨 계층을 활용하여 두 가지 관점의 라벨 집합을 함께 제공한다. 하나는 실행 파일 단위를 그대로 반영한 프로세스 단위 라벨(108종)이며, 다른 하나는 동일 응용을 서비스·사이트 관점으로 통합·정규화한 응용 단위 라벨(89종)이다. 후자는 chrome.exe·msedge.exe 등으로 흩어진 브라우저 트래픽을 접속 목적지(예: Google Chrome, ChatGPT, Claude, Notion) 기준으로 재정렬한 결과로, 프로세스 이름만으로는 드러나지 않는 사용자 수준의 서비스 소비 양상을 반영한다.

각 세션은 하나의 pcap 파일로 저장되며, 세션을 식별하는 핵심 필드는 파일명에 그대로 인코딩된 5-튜플(전송 프로토콜, 출발지 IP·포트, 목적지 IP·포트)이다. 여기에 수집 날짜, 호스트 IP, 라벨(프로세스/서비스/사이트) 정보가 디렉토리 경로로 부여되며, 동일 5-튜플이 재발생한 세션은 캡처 시각(연월일_시분초_밀리초) 접미로 구분된다. SNI가 존재하지 않거나 매핑 가능한 목적지를 확정할 수 없는 브라우저 세션은 no_visit 그룹으로 분리하여 사이트 라벨의 신뢰성을 유지한다. 이로써 하나의 세션은 프로토콜·종단점 정보(5-튜플)와 다단계 라벨을 동시에 갖추게 되어, 분석 목적에 따라 원하는 granularity의 부분 집합을 손쉽게 구성할 수 있다. 공개본에서는 이러한 라벨 정의를 별도 문서 없이 확인할 수 있도록, 응용 단위·프로세스 단위 라벨의 전체 목록과 각 라벨의 세션 수를 일자·호스트별 통계 파일(_exp_summary.json)로 함께 제공하며, 관점 트리와 원본 세션 경로의 대응 관계는 manifest 파일에 기록하여 라벨 부여 결과를 세션 단위로 역추적할 수 있도록 하였다.

3.7. 트래픽 분포

PRISM-2026은 7대 호스트의 3일간(2026.07.10 - 07.12) 연속 수집을 통해 총 1,320,366개의 세션으로 구성된다. 전송 계층 기준으로는 UDP가 55.4%, TCP가 44.6%를 차지하여, QUIC 및 DNS로 대표되는 UDP 트래픽의 비중이 TCP를 상회한다. 이는 최근 웹·응용 트래픽이 HTTP/3(QUIC)로 이전하는 경향을 반영한다.

표 3. PRISM-2026 상위 10개 라벨 분포 (프로세스 단위)
Table 3. Top-10 Label Distribution of PRISM-2026 (Process-level)

순위	라벨	세션 수	비율
1	svchost.exe	518,750	39.29%
2	chrome.exe	244,876	18.55%
3	servicemapcollector.exe	58,320	4.42%
4	claude.exe	49,188	3.73%
5	notion.exe	43,601	3.30%
6	code.exe	38,072	2.88%
7	remoting_host.exe	34,483	2.61%
8	codex.exe	33,041	2.50%
9	nexonplug.exe	29,287	2.22%
10	msedgewebview2.exe	28,076	2.13%

라벨 분포는 실제 호스트 환경의 특성을 그대로 드러내며 강한 불균형을 보인다. 표 3에서 보듯 다수의 Windows 서비스를 대리 실행하는 svchost가 전체 세션의 39.3%로 가장 큰 비중을 차지하고, 그 뒤를 브라우저(Google Chrome, 18.6%)가 잇는다. 상위 2개 라벨이 전체의 57.8%, 상위 5개가 69.3%, 상위 10개가 81.6%를 차지하는 반면, 하위 다수의 라벨은 수십 개 이하의 세션만을 가진다. 실제로 가장 큰 라벨(518,750 세션)과 가장 작은 라벨(2 세션)의 차이는 약 26만 대 1에 이르며, 프로세스 단위 108종 가운데 6종은 세션 수가 10개 미만이다. 이러한 심한 클래스 불균형 분포는 특정 응용만을 격리 수집하는 기존 데이터셋과 달리, 실사용 호스트에서 동시에 발생하는 트래픽을 그대로 담아낸 결과이며, 소수 클래스 학습이라는 현실적 난제를 함께 제공한다는 점에서 의의가 있다.

3.8. 암호화 비율

PRISM-2026이 실제 응용 트래픽을 반영하는지 확인하기 위해, 세션의 5-튜플에 나타난 서비스

표 4. PRISM-2026 서비스 포트 기반 암호화 트래픽 분류
Table 4. Service-Port-based Classification of Encrypted Traffic in PRISM-2026

구분	대표 포트	세션 수	전체 대비	DNS 제외 대비
암호화 (TLS/QUIC/SSH)	443, 22, 8443, 993...	471,927	35.7%	55.8%
평문(DNS 포함)	53, 80, 137-139 ...	513,672	38.9%	-
DNS(제어)	53	473,976	35.9%	-
비DNS 평문	80 등	39,696	3.0%	4.7%
기타/독자 프로토콜	-	334,767	25.4%	39.6%
전체	-	1,320,366	100%	-

포트를 기준으로 암호화 트래픽의 비중을 분석하였다. 표 4과 같이 TLS/QUIC(443), SSH(22) 등 암호화 프로토콜로 식별되는 세션이 전체의 35.7%를 차지하며, 이 중 단일 포트로는 HTTPS/QUIC(포트 443)가 35.0%로 콘텐츠 트래픽의 대부분을 구성한다. 평문으로 분류되는 세션은 38.9%이나, 그 대부분(전체의 35.9%)은 이름 확인을 위한 DNS(포트 53) 제어 평면 트래픽이며 실제 평문 콘텐츠 트래픽(HTTP 등)은 5% 미만에 불과하다. 나머지 25.4%는 게임·실시간 통신·내부 수집기 등이 사용하는 비표준 고포트 트래픽으로, 표준 포트만으로는 암호화 여부를 단정하기 어려운 독자 프로토콜에 해당한다.

DNS 제어 트래픽을 제외한 응용·콘텐츠 세션만을 기준으로 보면, TLS/QUIC 암호화 트래픽이 55.8%로 과반을 차지하고 평문 트래픽은 4.7%에 그친다. 즉 PRISM-2026의 콘텐츠 트래픽은 사실상 암호화되어 있으며, 이는 페이로드 기반 분석이 어려운 현대 네트워크 환경을 충실히 반영한다. 이러한 특성은 본 데이터셋이 암호화 트래픽 분류·행위 분석 연구의 벤치마크로 적합함을 뒷받침한다.

Ⅳ. 실험 및 결과

본 장에서는 PRISM-2026이 트래픽 분류 모델 학습에 실효적으로 활용될 수 있는지를 검증하기 위해, 라벨 granularity가 서로 다른 두 라벨 집합(프로세스 단위 108종, 응용 단위 89종)에 대하여 대표적 트래픽 분류 모델인 ET-BERT, TSHIRT, 2D CNN, XGBoost 네 가지를 각각 학습·평가하였다. 이들은 각각 사전학습 트랜스포머(ET-BERT), 프로토콜 구조 인지 계층 표현 모델(TSHIRT), 바이트 이미지 기반 CNN(2D CNN), 트리 기반 앙상블(XGBoost)로서 서로 다른 입력 표현과 학습 방식을 대표하며, 동일 데이터셋에서 표현 방식에 따른 성능 차이를 비교하기 위해 선정하였다. 성능은 정확도(Accuracy)를 기준으로 평가하였다.

4.1. 실험 환경

모델별 입력 표현과 학습 설정은 다음과 같다. ET-BERT[9]는 세션 페이로드 선두 784바이트를 토큰화한 시퀀스(seq_length 128)를 입력으로 하며, 사전학습된 암호화 트래픽 표현 모델을 미세조정하는 방식으로 배치 크기 32, 3 epoch, 학습률 기본값으로 학습하고 데이터를 7:1:2(train:valid:test)로 분할하였다.2D CNN은 Ethernet·IP·전송 계층 헤더를 제거한 784바이트를 28×28 그레이스케일 이미지로 변환·정규화(0 - 1)하여 입력하며, 두 개의 합성곱 계층(32·64 채널, 5×5 커널)과 두 개의 완전연결 계층으로 구성된다. 배치 크기 3,000, 300 epoch, Adam(학습률 1e-3)으로 학습하고 데이터를 8:2로 분할하였다. TSHIRT[10]는 패킷을 프로토콜 디섹터(Wireshark)로 파싱하여 세션 - 패킷 - 계층 - 필드의 계층 구조로 토큰화하는 구조 인지 표현 학습 모델이다. 각 필드를 하나의 토큰으로 두고, 필드 내부 바이트를 순서 불변 통계(mean·max 풀링)로 요약한 콘텐츠 임베딩에 위치 임베딩과 필드 길이 임베딩을 결합한다. 세션당 최대 256개 토큰, 필드당 최대 1,400바이트를 사용하며, 6개 계층·8개 어텐션 헤드·은닉 차원 528의 트랜스포머 인코더를 백본으로 한다. 계층 셔플 사전학습(필드·계층·패킷 순서 복원)으로 구조적 의존성을 학습한 뒤, 계층적 어텐션 풀링과 레벨 간 게이팅으로 미세조정하며, AdamW(학습률 1e-5), 배치 크기 32로 학습하고 데이터를 8:2로 분할하였다. 전처리 단계에서 TCP 3-way handshake 패킷과 DNS·NTP·SSDP 등 시스템 수준 트래픽을 제거하고 응용 계층(L7) 정보만을 입력으로 사용한다. XGBoost는 원시 바이트가 아닌 플로우 단위 통계 피처를 입력으로 사용한다. 각 세션(플로우)에서 (i) 패킷 크기·패킷 간 도착시간(IAT)·페이로드 길이에 대해 방향(양방향·정방향·역

방향)과 관측 패킷 수(전체 및 상위 5·10·15·20·25·30개)별로 15종 통계량을 산출한 945개 피처, (ii) 패킷 크기 시퀀스와 IAT 시퀀스를 각각 앞 100개까지 취한 200개 시계열 피처, (iii) 첫 패킷의 IP/TCP/UDP 헤더 필드 30개, (iv) 초당 바이트·패킷 수, 방향별 비율, 지속시간 등 플로우 요약 21개, (v) TCP 플래그 유형별 카운트 9개, 그리고 전송 프로토콜 구분 1개를 결합하여 총 1,206개의 수치 피처를 구성하였다. 이때 세션의 출발지·목적지 IP와 포트, 파일명에서 파생된 식별 정보는 모델이 종단점 신원을 암기하여 성능이 비현실적으로 과대평가되는 것(정보 누출)을 방지하기 위해 피처에서 명시적으로 제외하였다. 모델은 트리 수 600, 최대 깊이 8, 학습률 0.1, subsample 0.8, colsample_bytree 0.8, L2 정규화 계수 1.0의 히스토그램 기반 GBDT[11]로 학습하였으며, 데이터를 8:2로 계층적 분할(stratified)하였다. 네 모델은 표본이 10개 미만인 클래스를 제외한 세션 집합을 사용하되, 데이터 분할은 ET-BERT의 경우 7:1:2(train:valid:test), TSHIRT, 2D CNN 및 XGBoost의 경우 8:2(train:test) 계층적 분할을 적용하였으며, 입력 표현은 모델 특성에 따라 상이하다.

4.2. 기준 성능 평가

표 5. PRISM-2026에 대한 기준 모델 분류 성능
Table 5. Baseline Model Classification Performance on PRISM-2026

모델	라벨 단위	클래스 수	Test Acuracy
ET-BERT	프로세스	108	0.7730
ET-BERT	응용	89	0.7715
TSHIRT	프로세스	108	0.7942
TSHIRT	응용	89	0.7915
2D CNN	프로세스	108	0.8291
2D CNN	응용	89	0.8288
XGBoost	프로세스	108	0.9173
XGBoost	응용	89	0.9164

표 5는 네 모델의 두 라벨 집합에 대한 분류 정확도를 정리한 것이다. 네 모델 모두 프로세스 단위(108종)와 응용 단위(89종)에서 거의 동일한 정확도를 보여, 라벨 정의 방식보다 모델의 입력 표현과 특징 설계가 성능을 좌우함을 확인할 수 있다. 정확도 기준으로는 플로우 통계 피처를 사용한 XGBoost가 두 라벨 집합에서 각각 0.9173, 0.9164로 가장 높았고, 2D CNN(약 0.829), TSHIRT(약 0.79), ET-BERT(약 0.77)가 그 뒤를 이었다. 이는 세밀하게 설계된 플로우 수준 통계 특징이 본 데이터셋의 프로세스·응용 구분에 효과적이었음을 시사한다.

다만 PRISM-2026은 소수의 라벨에 세션이 크게 편중된 심한 클래스 불균형 특성을 가지므로, 정확도만으로는 표본이 적은 클래스의 분류 성능을 온전히 반영하기 어렵다. 따라서 클래스 불균형을 고려한 Macro-F1 등 보조 지표를 병행하여, 표본이 적은 클래스에 대한 성능까지 함께 평가할 필요가 있다.

이러한 결과는 PRISM-2026이 트랜스포머·구조 인지 모델·CNN·트리 기반 앙상블 등 서로 다른 계열의 분류 모델 모두에서 유의미한 학습이 가능한 라벨 품질을 갖추었으며, 프로세스·응용 두 라벨 granularity가 서로 다른 분석 관점을 제공하는 다단계 벤치마크로 활용될 수 있음을 뒷받침한다.

V. 결 론

본 연구에서는 패킷에서 라벨을 추론하는 기존 방식의 한계를 극복하기 위해, 트래픽을 발생시킨 엔드포인트로부터 라벨을 직접 수집하는 자동 라벨링 프레임워크 PRISM을 활용하여 다계층 네트워크 트래픽 데이터셋 PRISM-2026을 구축하였다. PRISM-2026은 7대의 실사용 호스트에서 3일간 수집한 총 1,320,366개의 세션으로 구성되며, 각 세션에 프로세스·서비스·사이트의 다계층 라벨과 5-튜플 기반 종단점 정보를 함께 부여함으로써 분석 목적에 따라 원하는 granularity의 라벨 집합을 유연하게 구성할 수 있도록 하였다.

5.1. 논의 및 한계

데이터 분석 결과 PRISM-2026은 실사용 환경의 특성을 그대로 반영한다. 전체 트래픽의 상당수가 TLS/QUIC 등으로 암호화되어 있어 페이로드 기반 분석이 어려운 현대 네트워크 환경을 잘 담고 있으며, 소수 프로세스가 트래픽의 대부분을 차지하는 심한 클래스 불균형을 보여 소수 클래스 학습이라는 현실적 난제를 함께 제공한다. 다만 본 연구에는 몇 가지 한계가 있다. 첫째, 원본 패킷 페이로드가 외부 스토리지에 보관되어 있어 암호화 여부 분석을 5-튜플(서비스 포트) 기반 추정으로 수행하였으므로, TLS/QUIC 핸드셰이크에 대한 심층 패킷 검사(DPI)를 통한 정밀화가 필요하다. 둘째, 수집 기간과 호스트 수가 제한적이어서 라벨 분포가 특정 사용 환경에 편중될 수 있다. 셋째, 기준 모델 실험은 대표적인 분류기에 한정하였고 평

가 지표 또한 정확도(Accuracy) 중심으로 제시하였다. 본 데이터셋은 상위 2개 라벨이 전체의 57.8%를 차지하고 프로세스 단위 108종 중 6종은 세션 수가 10개 미만인 심한 불균형 구조를 가지므로, 정확도만으로는 소수 클래스의 분류 성능이 충분히 드러나지 않는다. 따라서 Macro-F1·Weighted-F1 및 클래스별 precision/recall을 병행한 평가가 요구된다. 또한 현재의 계층적 무작위 분할은 동일 호스트·인접 시간대의 유사 세션이 학습 집합과 시험 집합에 함께 포함될 수 있어 성능이 낙관적으로 추정될 여지가 있으므로, 날짜별 또는 호스트별로 분리한 분할 평가를 병행할 필요가 있다. 넷째, 본 연구는 세션에 라벨이 부여된 비율인 매칭률(98.85%)을 보고하였을 뿐, 부여된 라벨이 실제와 일치하는지를 나타내는 라벨 정확도는 정량적으로 검증하지 못하였다. 특히 본 연구에서 새로 도입한 서비스 단위 라벨과 SNI 기반 사이트 단위 라벨은 표본에 대한 수작업 대조를 통해 정확도와 오류 유형을 제시할 필요가 있다. 이는 데이터셋의 신뢰성과 직결되는 사안으로 후속 연구에서 우선적으로 다룰 예정이다.

5.2. 결론 및 향후 연구

기준 모델 실험에서 ET-BERT, TSHIRT, 2D CNN, XGBoost를 프로세스(108종)·응용(89종) 두 라벨 집합에 대해 평가한 결과, 트리 기반의 XGBoost가 각각 0.9173, 0.9164의 정확도로 가장 우수하였고, 2D CNN(약 0.829), TSHIRT(약 0.79), ET-BERT(약 0.77)가 그 뒤를 이었다. 이는 라벨 정의 방식보다 모델의 표현력과 특징 설계가 성능을 좌우함을 보여주며, PRISM-2026이 다양한 분류 모델의 벤치마크로 활용될 수 있음을 뒷받침한다. 향후에는 (1) TLS/QUIC 핸드셰이크 기반 DPI를 통한 암호화 트래픽 정밀 분석, (2) 수집 호스트·기간의 확대를 통한 라벨 다양성 및 대표성 강화, (3) 클래스 불균형을 고려한 소수 클래스 학습 기법의 적용을 통해 데이터셋과 분류 성능을 함께 발전시키고자 한다.

구축한 PRISM-2026 데이터셋은 개인정보 보호를 위한 익명화 처리를 거쳐 GitHub 저장소(https://github.com/illiard1209/PRISM-2026-Dataset)에서 공개한다. 공개본은 동일한 세션 집합을 세 가지 관점(sessions, byproduct, byproduct_process)으로 구성하여 제공하며, 일자·호스트별 라벨 통계(_exp_summary.json)와 관점별 디렉토리 구조와 원본 세션 경로의 대응 관계를 기록한 manifest 파일을 함께 포함한다. 또한 배포 파일의 무결성을 확인할 수 있도록 SHA-256 체크섬을 제공한다. 개인정보 보호를 위해 수집 기관 대역에 속하는 IPv4 주소는 1:1 전단사 방식으로 익명화하되 /24 서브넷 구조를 보존하였으며, 해당 처리는 디렉토리명, 파일명 및 패킷 IPv4 헤더에 일관되게 적용하였다. 이를 통해 후속 연구자가 PRISM-2026의 다계층 라벨 구조와 실제 네트워크 트래픽을 활용하여 다양한 트래픽 분류 및 분석 연구를 수행할 수 있도록 한다.

References

[1] Guang Cheng and Song Wang, "Traffic classification based on port connection pattern," 2011 International Conference on Computer Science and Service System (CSSS), Nanjing, 2011, pp. 914-917, doi: 10.1109/CSSS.2011.5974374.

[2] M. S. Raza, S. B. A. Kazmi, R. Ali, M. M. Naqvi, H. Fiaz and A. Akram, "High Performance DPI Engine Design for Network Traffic Classification, Metadata Extraction and Data Visualization," 2024 5th International Conference on Advancements in Computational Sciences (ICACS), Lahore, Pakistan, 2024, pp. 1-6, doi: 10.1109/ICACS60934.2024.10473274.

[3] Moore, Andrew & Zuev, Denis. (2005). Internet traffic classification using Bayesian analysis techniques. Sigmetrics Performance Evaluation Review - SIGMETRICS. 33. 50-60. 10.1145/1064212.1064220.

[4] Lotfollahi, Mohammad & Shirali hossein zade, Ramin & Jafari Siavoshani, Mahdi & Saberian, Mohammadsadegh. (2020). Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning. Soft Computing. 24. 10.1007/s00500-019-04030-2.

[5] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun and A. A. Ghorbani, "Characterization of

Encrypted and VPN Traffic Using Time-Related Features," Proc. 2nd International Conference on Information Systems Security and Privacy (ICISSP), Rome, Italy, 2016, pp. 407-414, doi: 10.5220/0005740704070414.

[6] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," Proc. 4th International Conference on Information Systems Security and Privacy (ICISSP), Funchal, Portugal, 2018, pp. 108-116, doi: 10.5220/0006639801080116.

[7] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye and Yiqiang Sheng, "Malware traffic classification using convolutional neural network for representation learning," 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, 2017, pp. 712-717, doi: 10.1109/ICOIN.2017.7899588.

[8] J.-M. Kim, Y.-S. Jang, G.-M. Yu, S.-W. Nam, J.-S. Kim, and M.-S. Kim, "Socket-based automatic labeling framework for process-level network traffic classification," in Proc. KNOM Conf., 2026.

[9] Lin, Xinjie & Xiong, Gang & Gou, Gaopeng & Li, Zhen & Shi, Junzheng & Yu, Jing. (2022). ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification. 633-642. 10.1145/3485447.3512217.

[10] G.-M. Yu, Y.-S. Jang, J.-S. Kim, S.-W. Nam, J.-M. Kim, Y.-S. Choi and M.-S. Kim, "Beyond Byte-Level Modeling: Structure-Aware and Adaptive Traffic Classification for Encrypted Networks," Electronics, vol. 15, no. 9, 1828, 2026, doi: 10.3390/electronics15091828.

[11] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 2016, pp. 785-794, doi: 10.1145/2939672.2939785.

김 지 민 (Ji-Min Kim)



2026 : 고려대학교 컴퓨터융합소프트웨어학과 학사
2026 ~ 현재 : 고려대학교 컴퓨터소프트웨어학과 석박사통합과정
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

유 경 민 (Gyeong-Min Yu)



2023 : 고려대학교 컴퓨터융합소프트웨어학과 학사
2026 : 고려대학교 컴퓨터융합소프트웨어학과 석사
2026 ~ 현재 : 고려대학교 컴퓨터소프트웨어학과 박사과정
<관심분야> 네트워크 트래픽 분류, 암호화 트래픽 분석, 딥러닝 기반 네트워크 트래픽 탐지

장 윤 성 (Yoon-Seong Jang)



2023 : 고려대학교 컴퓨터융합소프트웨어학과 학사
2023 ~ 현재 : 고려대학교 컴퓨터소프트웨어학과 석박사통합과정
<관심분야> 네트워크 트래픽 분류, 암호화 트래픽 분석, 공격 트래픽 탐지

남 승 우 (Seung-Woo Nam)



2023 : 고려대학교 컴퓨터융합소프트웨어학과 학사
2023 ~ 현재 : 고려대학교 컴퓨터소프트웨어학과 석박사통합과정
<관심분야> 트래픽 분석, 네트워크 보안, ZTA

김 주 성 (Ju-Sung Kim)



2023 : 고려대학교 컴퓨터융합소프트웨어학과 학사
2023 ~ 현재 : 고려대학교 컴퓨터소프트웨어학과 석박사통합과정
<관심분야> 네트워크 트래픽 분류, 암호화 트래픽 분석

백 의 준 (Ui-Jun Baek)



2018년 : 고려대학교 컴퓨터정보학과 학사
2025년 : 고려대학교 컴퓨터정보학과 박사
2025년 ~ 현재 : 한국과학기술정보연구원(KISTI) 선임연구원

<관심분야> 블록체인 거래 모니터링, 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

김 명 섭(Myung-Sup Kim)



1998년 : 포항공과대학교 전자계산학과 학사
2000년 : 포항공과대학교 전자계산학과 석사
2004년 : 포항공과대학교 전자계산학과 박사
2024년~2006년 : Dept. of ECS, Univ of Toronto Canada
2006년 ~ 현재 : 고려대학교 컴퓨터정보학과 교수
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석, 단일미디어 네트워크

https://doi.org/10.22670/knom.2026.29.1.45

사물인터넷 환경에서 효과적인 분할 연합 학습을 위한 통신 오버헤드 최적화 연구 동향

정 한 길◆, 김 종 욱*, 최 미 정°

A Survey on Communication Overhead Optimization for Efficient Split Federated Learning in Internet of Things Environments

Han-Gil Jeong◆, Jong-Wouk Kim*, Mi-Jung Choi°

요 약

분할 연합 학습(Split Federated Learning)은 원시 데이터를 공유하지 않고도 자원 제약적인 장치들이 학습에 참여할 수 있도록 함으로써 IoT(Internet of Things) 환경에서 많은 주목을 받고 있다. IoT 장치들의 데이터를 활용하면 지능형 헬스케어, 제조, 도시, 교통 등 다양한 분야에서 지능형 서비스를 제공할 수 있다. 그러나 분할 연합 학습은 학습 과정에서 학습 관련 정보를 반복적으로 송수신하는 구조로 인해 통신 오버헤드를 발생시킨다. 이러한 오버헤드는 제한된 대역폭과 에너지 자원을 갖는 IoT 환경에서 학습 시간과 에너지 소모를 증가시키며, 결과적으로 실제 적용을 제약하는 요인으로 작용한다. 이에 따라 분할 연합 학습의 통신 오버헤드를 최적화하기 위한 연구가 진행되고 있다. 본 논문은 IoT 환경에서 분할 연합 학습을 적용하기 위해 고려해야 하는 주요 통신 오버헤드 문제를 소개하고, 각 연구가 다루는 세부적인 통신 오버헤드 문제를 중심으로 해결 전략을 체계적으로 비교한다.

Key Words : Internet of Things, Split Federated Learning, Communication Overhead Optimization

ABSTRACT

Split federated learning has attracted significant attention in Internet of Things (IoT) environments, as it enables resource-constrained devices to participate in training without sharing raw data. By utilizing data from IoT devices, intelligent services can be provided in healthcare, manufacturing, cities, and transportation. However, split federated learning incurs communication overhead due to the repeated exchange of training-related information during the training process. This overhead increases training time and energy consumption in IoT environments with limited bandwidth and energy resources, becoming a major constraint on practical deployment. Thus, recent studies on optimizing the communication overhead of split federated learning have been conducted. This paper introduces major communication overhead problems in split federated learning for IoT environments and systematically compares solution strategies for the specific problems addressed in each study.

※ 이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원-지역지능화혁신인재양성사업의 지원(IITP-2026-RS-2023-00260267, 0.5)과 정부(대한민국 교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2024S1A5C3A01043273, 0.5).
◆ First Author : Kangwon National University Department of Data Science, jeong00400@kangwon.ac.kr
° Corresponding Author : Kangwon National University Department of Computer Science and Kangwon National University IGP. in Medical Bigdata Convergence, mjchoi@kangwon.ac.kr
* Kangwon National University Department of Computer Science, jw.kim@kangwon.ac.kr
논문번호 : KNOM2026-01-009, Received July 28, 2026; Revised August 11, 2026; Accepted August 18, 2026

Ⅰ. 서 론

최근 스마트폰, 웨어러블 기기, 드론, 그리고 스마트 센서와 같은 Internet of Things(IoT) 장치가 다양한 서비스 환경에 보급되면서, Deep Neural Network(DNN) 모델 학습에 활용할 수 있는 데이터가 지속적으로 생성되고 있다[1][2]. IoT 장치는 사용자의 행동, 생체 신호, 주변 환경, 설비 상태, 영상 및 음성 정보와 같은 다양한 데이터를 생성 및 수집하며, 이러한 데이터는 DNN 모델 기반 지능형 서비스를 구현하기 위한 주요 학습 자원으로 활용된다[3]. 예를 들어, 지능형 헬스케어 서비스 분야에서는 웨어러블 기기 및 의료 IoT 장치의 데이터를 기반으로 사용자 건강 상태를 모니터링하거나 이상 징후를 탐지할 수 있다[1]. 지능형 제조 서비스 분야에서는 설비 센서 데이터를 활용하여 설비 고장을 예측하고 생산 품질을 관리할 수 있다[3][4]. 나아가 지능형 도시 및 지능형 교통 서비스 분야에서는 카메라, 차량, 도로 센서 등에서 생성되는 데이터를 활용하여 교통 혼잡 예측, 사고 감지와 같은 지능형 서비스를 제공할 수 있다[2][4].

일반적으로 DNN 모델 학습은 학습 데이터를 중앙 서버에 수집한 뒤, 해당 서버에서 모델을 학습하는 중앙 집중형 방식으로 수행된다[5]. 이는 높은 성능의 모델을 구축하기 위해서는 대규모 학습 데이터로부터 고수준 특징을 학습해야 하기 때문이다[6][7]. 그러나 IoT 환경에서 학습 데이터는 중앙 서버에 미리 저장되어 있는 것이 아니라, 여러 IoT 장치에서 분산되어 생성된다. 따라서 IoT 환경에서 중앙 집중형 학습 방식을 적용하기 위해서는 각 IoT 장치가 생성한 원시 데이터를 중앙 서버로 전송한 뒤, 서버에서 모델을 학습해야 한다[8]. 대규모 IoT 환경에서 생성되는 원시 데이터를 중앙 서버로 수집하는 과정은 높은 네트워크 대역폭을 요구하며, IoT 장치의 통신 지연과 에너지 소모를 증가시킬 수 있다[4][9]. 또한 원시 데이터는 사용자의 생체 정보, 얼굴 이미지, 위치 정보와 같은 민감한 정보를 포함할 수 있으므로, 이러한 데이터를 중앙 서버로 전송하는 과정에서 데이터 프라이버시 문제를 초래할 수 있다[8]. 따라서 IoT 환경에서는 원시 데이터를 중앙 서버로 직접 수집하지 않으면서도, 분산된 IoT 장치의 데이터를 활용하여 모델을 학습할 수 있는 방법이 필요하다.

이러한 필요성에 따라 연합학습(Federated Learning), 분할학습(Split Learning), 분할 연합 학습(Split Federated Learning)과 같은 분산 기계학습 방식이 연구되고 있다[10]. 세 방식 모두 분산된 장치들의 원시 데이터를 중앙 서버로 수집할 필요 없이 모델을 학습할 수 있다는 공통점을 가진다. 그러나 각 방식은 모델을 어느 위치에서 학습하는지, 서버와 클라이언트가 어떤 역할을 수행하는지, 그리고 학습 과정에서 모델 파라미터 또는 모델의 중간 연산 결과 중 무엇을 전송하는지에 따라 서로 다른 구조를 가진다[10].

연합학습은 여러 클라이언트가 자신의 원시 데이터를 외부로 공유하지 않고, 서버와 협력하여 하나의 전역 모델을 학습하는 분산 기계학습 방식이다[11]. 연합학습에서 서버는 현재 전역 모델을 여러 클라이언트에 전달하고, 각 클라이언트는 자신의 로컬 데이터로 모델을 학습한다. 이후 클라이언트는 원시 데이터가 아닌 학습된 모델 파라미터만을 서버로 전송하며, 서버는 여러 클라이언트의 모델 파라미터를 집계하여 새로운 전역 모델을 생성한다. 연합학습의 장점은 원시 데이터를 클라이언트 장치 내부에 유지하면서도, 여러 IoT 장치의 데이터를 전역 모델 학습에 활용할 수 있다는 점이다[12]. 그러나 연합학습에서는 각 클라이언트가 전체 모델의 파라미터를 보유하고 학습해야 하므로, 연산 능력, 메모리 그리고 에너지 자원이 제한된 IoT 장치에서는 복잡한 모델을 학습하는 데 부담이 발생할 수 있다[13].

분할학습은 모델을 클라이언트 측 모델과 서버 측 모델로 분할하고, 두 모델이 협력하여 하나의 모델을 학습하는 분산 기계학습 방식이다[14]. 분할학습에서 모델은 분할 계층인 cut layer를 기준으로 클라이언트 측 모델과 서버 측 모델로 분할된다. 클라이언트는 cut layer까지의 연산을 수행하고, 서버는 cut layer 이후의 연산을 수행한다. 학습 과정에서 클라이언트는 자신의 로컬 데이터를 클라이언트 측 모델에 입력하여 cut layer까지 forward propagation을 수행하고, 이 과정에서 생성된 모델의 중간 연산 결과인 activation을 서버로 전송한다. 서버는 수신한 activation을 서버 측 모델의 입력으로 사용하여 서버 측 모델의 forward propagation을 수행한다. 이후 서버는 backward propagation을 통해 서버 측 모델의 파라미터를 업데이트하고, 클라이언트 측 모델의 backward propagation에 필요한

gradient를 계산하여 클라이언트로 전송한다. 클라이언트는 수신한 gradient를 통해 클라이언트 측 모델 파라미터를 업데이트하며 학습을 마무리한다. 분할 학습의 장점은 클라이언트가 모델 전체가 아닌, cut layer까지만 학습함으로써 클라이언트 장치의 학습 부담을 줄일 수 있다는 점이다[14][15][16]. 그러나 일반적인 분할학습에서는 여러 클라이언트가 서버와 순차적으로 학습하기 때문에, 학습 과정에 참여하는 IoT 장치 수가 증가할수록 전체 학습 시간이 길어진다[17]. 이와 같은 학습 시간의 증가는 IoT 장치의 에너지를 소모시킬 수 있다[6].

분할 연합 학습은 분할학습의 모델 분할 구조와 연합학습의 병렬 학습 및 모델 집계 구조를 결합한 분산 기계학습 방식이다[18]. 분할 연합 학습에서는 일반적인 분할학습과 달리 여러 클라이언트가 클라이언트 측 모델을 병렬로 학습하고, 서버는 학습된 클라이언트 측 모델 파라미터를 집계하여 전역 클라이언트 측 모델을 생성한다. 연합학습과 분할학습의 장점을 결합함으로써 분할 연합 학습은 자원이 제한된 IoT 환경에서 IoT 장치 측의 학습 부담을 줄이고 전체 학습 시간을 줄일 수 있다는 장점을 가진다[10]. 최근에는 지능형 헬스케어, 차량 엣지 컴퓨팅, 지능형 도시, IoT 기반 이상 탐지 등 다양한 서비스 분야를 대상으로 분할 연합 학습을 적용하려는 연구가 활발히 진행되고 있다[4][10]. 이러한 연구 흐름은 자원 제약적인 IoT 환경에 분산 기계학습을 적용함에 있어서 분할 연합 학습이 유망한 방식임을 보여준다[4][7][10].

이처럼 분할 연합 학습이 IoT 환경에서 유망한 분산 기계학습 방식으로 제시되고 있음에도, 이를 효과적으로 적용하기 위해서는 여러 도전 과제가 존재한다[4][10]. 그중 분할 연합 학습의 통신 오버헤드 문제는 제한된 네트워크 대역폭과 에너지 자원을 갖는 IoT 환경에서 전체 학습 시간을 지연시키고 에너지 소모를 증가시킬 수 있다[7]. 통신 오버헤드 문제는 학습 과정에 따라 크게 네 가지로 구분할 수 있다[10]. 첫 번째는 activation transmission 문제로, 이는 클라이언트가 생성한 activation을 서버로 전송하는 과정에서 발생하는 문제이다. 두 번째는 gradient transmission 문제로, 서버가 클라이언트 측 모델의 backward propagation을 위해 gradient를 클라이언트로 전송하는 과정에서 발생하는 문제이다. 세 번째는 model update 문제로, 전역 모델을 업데이트하기 위해 서버와 클라이언트가 모델 파라미터를 송수신하는 과정에서 발생하는 문제이다. 네 번째는 control message 문제로, 클라이언트 선택, 모델 집계 시점 조율 등 서버와 클라이언트 간 학습 진행을 조율하기 위한 통신 과정에서 발생하는 문제이다[10]. 최근 이러한 통신 오버헤드 문제를 해결하고자 다양한 연구가 수행되고 있다. 또한, 분할 연합 학습의 통신 오버헤드 문제를 포함하여 여러 도전 과제에 대한 연구 동향 논문이 지속적으로 발표되고 있다[4][10][19][20].

기존 연구 동향 논문들은 분할 연합 학습의 방법, 응용 분야 및 도전 과제를 포괄적으로 정리하거나, 6G 네트워크와 같은 특정 응용 분야에 한정하여 분할 연합 학습을 다루었다[19][21]. 또한, 분할 연합 학습의 통신 오버헤드를 여러 문제 유형으로 구분하고, 이를 완화하기 위한 해결 전략을 분류하였다[10]. 세부적인 통신 오버헤드 문제에 따라 선택할 수 있는 해결 전략이 다양하므로, 해결 전략의 적용에는 세부 문제의 특성에 대한 고려가 필요하다. 그러나 각 연구가 다루는 세부적인 통신 오버헤드 문제를 중심으로 해결 전략을 체계적으로 비교한 연구는 부족하다.

본 논문에서는 IoT 환경에서 분할 연합 학습을 효과적으로 적용하기 위해 통신 오버헤드 문제를 중심으로 관련 연구 동향을 조사한다. 기존 연구 동향 논문에서는 통신 오버헤드를 네 가지 문제 유형으로 구분하였다[10]. 본 논문에서는 이 네 가지 문제 유형을 바탕으로 각 연구가 다루는 세부적인 통신 오버헤드 문제를 중심으로 해결 전략을 체계적으로 비교한다. 또한, 비교 결과를 바탕으로 IoT 환경에서 분할 연합 학습을 적용할 때 고려할 수 있는 해결 전략에 대해 논의한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 IoT 환경에 대해 간략히 설명하고, 연합학습, 분할학습, 분할 연합 학습의 기본 구조와 학습 과정을 설명한다. 제 3장에서는 분할 연합 학습에서 발생하는 통신 오버헤드 문제를 activation transmission, gradient transmission, model update, control message로 구분하고, 각 문제에 대해 설명한다. 제 4장에서는 각 연구가 다루는 세부적인 통신 오버헤드 문제와 해결 전략을 체계적으로 정리한다. 마지막으로 제 5장에서는 본 논문의 내용을 요약하고, IoT 환경에서 분할 연합 학습을 적용할 때 고려할 수 있는 해결 전략에 대해 논의한다.

Ⅱ. 배경지식

2.1 IoT 환경

IoT 환경의 핵심 제약 중 하나는 IoT 장치가 제한된 에너지 자원을 가진다는 점이다[22]. 특히 IoT 장치의 에너지는 데이터 처리 연산과 서버와의 데이터 송수신 과정에서 소모된다[23][24][25]. 따라서 IoT 환경에서 분산 기계학습을 효과적으로 적용하기 위해서는 IoT 장치의 모델 연산 비용과 통신 비용을 함께 고려해야 한다.

스마트폰, 웨어러블 기기, 드론과 같은 IoT 장치는 제한된 연산 능력과 메모리 용량을 가지고 있으며, 모델을 학습하기 위해서는 많은 연산량과 메모리 사용량을 요구한다[1]. 대표적인 예시로 512MB의 메모리와 Arm Cortex A7 CPU를 가지는 Hi3516E V200 SoC 이미지 처리 프로세서가 VGG-16 이미지 분류 모델을 효과적으로 학습하는 것은 불가능하다. VGG-16 모델은 528MB의 메모리 요구량과 1초당 16억 부동소수점 연산을 요구하기 때문이다[26]. 또한 IoT 장치가 모델을 수렴시키기 위해 많은 연산을 수행할 경우 학습 완료 시간이 증가하여 에너지 소모량이 커질 수 있다[6].

IoT 환경에서는 사용하는 통신 기술에 따라 가용 대역폭과 에너지 소모 특성이 달라질 수 있다. 예를 들어 Wi-Fi, Bluetooth, LoRaWAN과 같은 통신 기술은 서로 다른 데이터 전송률과 전력 소모 특성을 가진다[7][27]. 또한, IoT 환경에서 통신 상태는 고정적이지 않고 시간에 따라 변동될 수 있다. 특히 통신 상태가 좋지 않을 때는 IoT 장치의 데이터 전송 시간이 증가하고 에너지 소모량이 커질 수 있다[9]. 또한 IoT 장치의 에너지 소모량은 연산으로 인한 소모량보다 통신으로 인한 소모량이 더 크게 나타날 수 있으므로, 통신 오버헤드를 줄이는 것이 중요하다[7].

2.2 연합학습

연합학습은 분산 기계학습 프레임워크로, 구글의 키보드 패턴 예측 및 의료, 지능형 교통, 산업 자동화 분야의 다양한 사용 사례에 성공적으로 적용되었다[28][29][30]. 서버는 분산된 클라이언트들로부터 모델을 학습하기 위해 데이터를 수집함에 있어서 원시 데이터가 아니라 클라이언트의 모델 가중치를 집계한다[11]. 이를 통해 연합학습은 데이터 프라이버시를 보호하면서 여러 클라이언트와의 학습을 가능하게 한다[10].

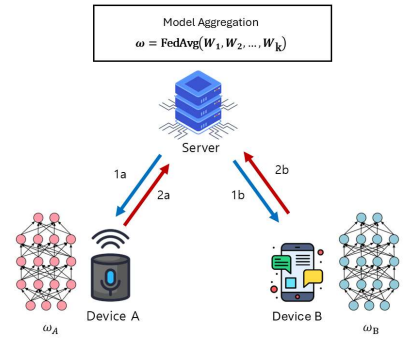


그림 1. 연합학습 과정
Fig. 1. Training process of Federated Learning

그림 1은 장치 A와 장치 B가 전역 모델 학습에 참여하는 연합학습 과정을 보여준다. 먼저 서버는 전역 모델 ω을 초기화한 후 장치 A와 장치 B에게 공유한다(그림 1.1a, 그림 1.1b). 그 다음, 각 장치들은 자신의 로컬 데이터를 이용하여 모델 가중치를 업데이트한다. 서버는 업데이트 된 모델 가중치 ω_A, ω_B를 집계한 후 전역 모델 ω을 계산한다(그림 1.2a, 그림 1.2b). 이후 계산된 전역 모델 ω를 다시 장치 A와 장치 B에게 공유하고 해당 과정을 모델이 수렴할 때까지 반복한다[20].

하지만 모델 전체를 학습하는 것은 많은 연산량을 요구한다. 특히, 자원 제약적인 IoT 장치에서는 모델을 수렴시키기 위해 많은 학습 시간이 필요하다[31]. 또한, 학습하는 모델의 파라미터 수가 많을수록 모델 가중치 크기가 커지므로 통신 오버헤드 문제가 발생한다[32].

2.3 분할학습

분할학습은 자원 제약적인 엣지 장치에서도 분산 기계학습을 가능하게 하는 방식으로서 등장한 프레임워크다[32]. 서버는 분산된 클라이언트들로부터 모델을 학습하기 위해 데이터를 수집함에 있어서 원시 데이터가 아니라 클라이언트의 중간 모델 연산 결과인 activation과 정답 label을 수신받는다[14]. 이를 통해 분할학습은 데이터 프라이버시를 보호하면서도 자원 제약적인 장치를 가진 클라이언트가 학습에 참여할 수 있도록 한다[10].

그림 2는 장치 A와 장치 B가 서버와 협력하여

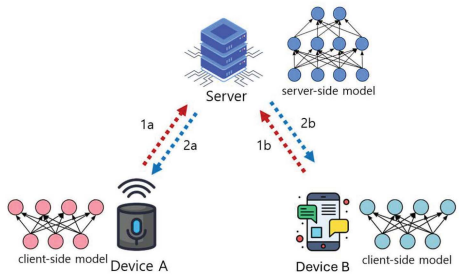


그림 2. 분할학습 과정
Fig. 2. Training process of Split Learning

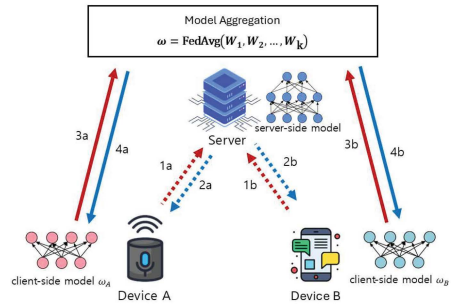


그림 3. 분할 연합 학습 과정
Fig. 3. Training process of Split Federated Learning

모델을 학습하는 분할학습 과정을 보여준다. 먼저 모델은 특정 계층인 cut layer를 기준으로 클라이언트 측 모델과 서버 측 모델로 분할된다. 그 다음 각 장치들은 자신의 로컬 데이터를 자신의 클라이언트 측 모델에 입력하여 cut layer까지 forward propagation을 수행한다. 이 과정에서 생성된 activation은 정답 label과 함께 서버로 전송된다(그림 2.1a, 그림 2.1b). 서버는 수신한 activation을 서버 측 모델의 입력으로 사용하여 forward propagation을 이어서 수행하고, 수신한 정답 label을 기반으로 손실을 계산한다. 이후 backward propagation을 통해 서버 측 모델의 가중치를 업데이트하고 서버는 cut layer에 해당하는 gradient를 각 장치로 전송한다(그림 2.2a, 그림 2.2b). 장치 A와 장치 B는 수신한 gradient를 이용하여 클라이언트 측 모델의 backward propagation을 수행하고 모델 가중치를 업데이트한다. 이러한 과정은 모델이 수렴할 때까지 반복한다.

하지만 분할학습에서는 activation과 gradient를 빈번하게 송수신해야 하므로 통신 오버헤드가 발생한다[7]. 또한, 분할학습에서는 다수의 클라이언트가 서버와 순차적으로 학습을 수행하므로, 학습에 참여하는 클라이언트 수가 증가할수록 전체 학습 시간이 길어진다[17]. 이로 인해 학습 과정에 참여하는 IoT 장치들의 에너지 소모를 증가시킬 수 있다[6].

2.4 분할 연합 학습

분할 연합 학습은 분할학습의 모델 분할 구조와 연합학습의 모델 집계 구조를 결합한 분산 기계학습 프레임워크이다[18]. 이를 통해 분할

연합 학습은 데이터 프라이버시를 보호하면서도 자원 제약적인 장치를 가진 클라이언트들이 병렬적으로 학습에 참여할 수 있도록 한다[4][18]. 특히 일반적인 분할학습과 달리 여러 클라이언트가 순차적으로 학습하지 않고 병렬로 학습을 수행하므로, 클라이언트 수 증가에 따른 전체 학습 시간 증가를 완화할 수 있다[4].

그림 3은 장치 A와 장치 B가 협력하여 모델을 학습하는 분할 연합 학습 과정을 보여준다. 먼저 모델은 특정 계층인 cut layer를 기준으로 클라이언트 측 모델과 서버 측 모델로 분할된다. 그 다음, 각 장치들은 자신의 로컬 데이터를 클라이언트 측 모델에 입력하여 cut layer까지 forward propagation을 수행한다. 이 과정에서 생성된 activation은 정답 label과 함께 서버로 전송된다(그림 3.1a, 그림 3.1b). 서버는 수신한 activation을 서버 측 모델의 입력으로 사용하여 forward propagation을 이어서 수행하고, 수신한 정답 label을 기반으로 손실을 계산한다. 이후, backward propagation을 통해 서버 측 모델의 가중치를 업데이트하고 서버는 cut layer에 해당하는 gradient를 각 장치로 전송한다(그림 3.2a, 그림 3.2b). 장치 A와 장치 B는 수신한 gradient를 이용하여 클라이언트 측 모델의 backward propagation을 수행하고 모델 가중치를 업데이트한다. 마지막으로 일반적인 분할학습과 달리, 각 장치들은 업데이트된 클라이언트 측 모델 가중치를 서버로 전송한다(그림 3.4a, 그림3.4b). 서버는 수신한 클라이언트 측 모델 가중치를 집계하여 새로운 전역 클라이언트 측 모델을 계산하고, 이를 다음 학습 라운드에서 장치 A와 장치 B에게 다시 공유한다(그림 3.3a, 그림 3.3b). 이러한 과정은 모

델이 수렴할 때까지 반복한다[20].

하지만 분할 연합 학습에서는 일반적인 분할학습과 같이 클라이언트가 activation과 gradient를 빈번하게 송수신해야 한다. 또한 연합학습과 같이 클라이언트 측 모델 가중치도 주기적으로 송수신해야 한다[10]. 이로 인해 분할 연합 학습은 학습 과정에서 연합학습과 분할학습의 통신 오버헤드 문제를 모두 가진다. 또한, 다수의 기존 연구들이 안정적인 통신 상태를 가정하여 분할 연합 학습의 성능을 개선하려 한다[33]. 반면 IoT 환경은 불안정한 통신 상태를 가지므로 분할 연합 학습을 IoT 환경에 효과적으로 적용하기 위해서는 통신 오버헤드 문제를 고려하는 연구가 필요하다. 따라서, 본 논문에서는 통신 오버헤드 문제를 다루는 관련 연구들을 조사한다. 이를 위해 통신 오버헤드 문제가 구체적으로 무엇인지 설명한다.

Ⅲ. 분할 연합 학습의 통신 오버헤드 문제

분할 연합 학습은 클라이언트와 서버가 협력하여 하나의 모델을 학습하는 구조이기 때문에, 학습 과정에서 여러 통신 과정이 반복적으로 발생한다. 특히 제한된 네트워크 대역폭과 에너지 자원을 갖는 IoT 환경에서는 이러한 통신 오버헤드가 전체 학습 시간을 증가시키고, IoT 장치의 에너지 소모를 높이는 요인으로 작용할 수 있다. Naseh et al.은 분할 연합 학습의 통신 오버헤드가 smashed data transmission, gradient transmission, model updates, control messages에서 발생한다고 설명한다[10]. 본 논문에서는 이를 기반으로 분할 연합 학습의 통신 오버헤드 문제를 activation transmission, gradient transmission, model update, control message의 네 가지 유형으로 구분하여 설명한다. 그림 4에서는 분할 연합 학습을 진행하면서 통신 오버헤드가 발생하는 과정을 보여준다.

- Activation Transmission: 클라이언트 장치가 cut layer까지 forward propagation을 수행한 뒤 생성한 activation을 서버로 전송하는 과정에서 발생한다. 분할학습 및 분할 연합 학습에서 activation은 서버 측 모델의 입력으로 사용되므로 서버가 forward propagation과 손실 계산을 이어서 수행하기 위해서는 클라이언트가 생성한 activation이 서버로 전송되어야 한다. 특히, 분할 연합 학습에서는 여러 클라이언트가 병렬적으로 학습에 참여하므로, 각 클라이언트가 미니 배치 단위로 생성한 activation이 서버로 빈번하게 전송된다. 업링크 통신에서 더 많은 에너지를 소모하는 IoT 환경 특성상, activation transmission은 에너지 소모량을 증가시키는 주요 원인으로 작용한다[7][34].
- Gradient Transmission: 서버가 cut layer에 해당하는 gradient를 클라이언트로 전송하는 과정에서 발생한다. 분할학습 및 분할 연합 학습에서 클라이언트 측 모델은 서버 측 모델에서 계산된 gradient를 수신해야 backward propagation을 이어서 수행할 수 있다. 특히, 분할 연합 학습에서는 여러 클라이언트가 병렬적으로 학습에 참여하므로, 각 클라이언트가 미니 배치 단위로 전송한 activation에 대응하는 gradient가 서버로부터 빈번하게 전송된다. 따라서 클라이언트는 서버의 gradient를 수신 하기 위해 빈번하게 대기해야 하므로, 제한된 네트워크 대역폭을 가지는 IoT 환경에서는 gradient transmission이 전체 학습 시간을 증가시킬 수 있다[35]. 이는 gradient를 수신받는 과정과 함께 에너지 소모량을 증가시키는 주요 요인으로 작용한다[7].
- Model Update: 클라이언트 측 모델의 전역 모델을 생성하기 위해 클라이언트 측 모델 가중치를 서버와 송수신하는 과정에서 발생한다. 분할 연합 학습에서는 각 클라이언트가 activation과 gradient를 이용하여 클라이언트 측 모델을 업데이트한 뒤, 업데이트된 클라이언트 측 모델 가중치를 서버로 전송한다. 서버는 수신한 클라이언트 측 모델 가중치를 집계하여 새로운 클라이언트 측 모델의 전역 모델을 생성하고, 이를 다시 각 클라이언트에게 전송한다. 특히, 참여 클라이언트 수가 증가하거나 클라이언트 측 모델의 크기가 커질수록 서버가 수신하고 다시 전송해야 하는 모델 가중치의 통신량이 증가한다[36]. 따라서, 모든 클라이언트가 전역 모델 가중치를 수신하거나 서버가 클라이언트 측 모델 가중치를 모두 집계할 때까지 학습 라운드는 지연될 수 있다[38]. 이로 인해 제한된 네트워크 대역폭을 가지는 IoT 환경에서는 model update가 전체 학습 시간을 지연시키고, 모델 가중치 송·수신 과정에서 IoT 장치의 에너지 소모량을 증가시키는 주요 요인으로 작용한다[2].
- Control Message: 서버와 클라이언트 간 학습 진행을 조율하기 위해 필요한 제어 정보를 송·수신하는 과정에서 발생한다. Naseh et al.은

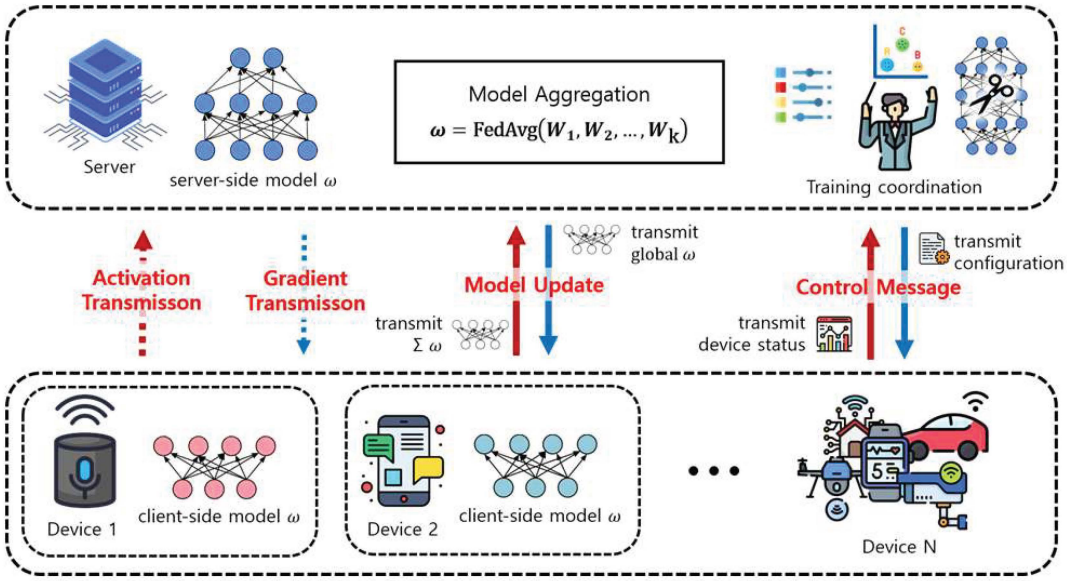


그림 4. 분할 연합 학습에서 통신 오버헤드가 발생하는 과정
Fig. 4. Process of communication overhead occurrence in split federated learning

control message를 분할 연합 학습의 통신 오버헤드 중 하나로 제시하고, 이를 학습 과정의 동기화 및 조율에 대한 오버헤드 문제로 설명한다[10]. 분할 연합 학습에서는 학습 진행을 위해 참여 클라이언트 선택, cut layer 설정, 모델 집계 및 전역 모델 공유와 같은 제어 과정이 필요하다[38][39]. 따라서, 클라이언트 수가 증가하거나 클라이언트의 참여 상태가 자주 변하는 가변적인 IoT 환경에서는 클라이언트 선택, 자원 할당, 데이터 동기화와 같은 클라이언트 관리 과정이 복잡해질 수 있다. 이로 인해 모델 동기화와 학습 과정 조율에 어려움이 발생하여 전체 학습 시간이 지연될 수 있다[10][19].

Ⅳ. 통신 오버헤드 최적화 연구 동향

분할 연합 학습은 다양한 분야에서 높은 활용 가능성을 보이며, IoT 환경에도 이를 적용하려는 연구가 활발히 진행되고 있다. 그러나 분할 연합 학습을 IoT 환경에 효과적으로 적용하는 데 있어서 통신 오버헤드 문제가 핵심 제약으로 작용한다. 본 장에서는 이러한 통신 오버헤드 문제를 다룬 기존 연구들을 조사한다. 이를 위해 3장에서 구분한 통신 오버헤드 문제를 중심으로 기존 연구들을 분류하고, 각 연구의 해결 전략과 성능을 설명한다. 표 1에서는 조사한 연구들을 통신 오버헤드 문제별로 구분하고, 각 문제에 대응하는 해결 전략과 각 연구에서 제시한 주요 성능 결과 및 정확도에 미치는 영향을 정리한다. 다만, 표 1에 정리한 주요 성능 결과는 각 연구에서 서로 다른 비교 대상, 데이터셋, 모델 및 평가 지표를 사용하여 산출되었으므로, 각 연구에서 제시한 주요 성능 결과를 직접 비교하기에는 한계가 있다.

Xing et al.은 에너지 하베스팅 기반 IoT 장치에서 activation 및 gradient 전송으로 인한 통신 에너지 소모를 줄이고자 하였다[7]. 이를 위해 제안 기법은 IoT 장치의 에너지 상태와 모델 훈련 손실 변화를 기반으로 activation 및 gradient 전송 여부를 선택적으로 결정한다. 구체적으로, IoT 장치는 현재 에포크(epoch)에서 모델 업데이트에 필요한 에너지가 충분하고 이전 업데이트 시점 대비 훈련 손실 변화가 특정 임계값 이상일 때 activation을 서버로 전송한다. 서버는 해당 activation을 이용하여 서버 측 학습을 수행한 뒤, IoT 장치가 backward propagation을 수행할 수 있도록 gradient를 전송한다. 반면, 위 조건을 만족하지 않아 클라이언트 측 모델이 업데이트되지 않는 경우에는 gradient 전송을 생략하고, 서버는 이전 에포크에서 저장한 activation을 재사용한다. 이를 통해 제안 기법은 activation 및 gradient 전송 횟수를 줄이고, 통신 에

표 1. 분할 연합 학습에서 통신 오버헤드 최적화 연구 비교
Table 1. Comparison of communication overhead optimization research in Split Federated Learning

Research	Problem	Solution strategy	Performance gain	Accuracy impact
[7]	activation transmission	selective transmission, caching and reuse	energy consumption ↓ 28.8 - 80.5%	maintained
	gradient transmission	selective transmission		
[38]	activation transmission	dynamic split	convergence time ↓ 7.7×	maintained
	gradient transmission			
	model update	aggregation interval adjust		
[39]	activation transmission	dynamic split	training time ↓, energy consumption ↓	maintained
	gradient transmission	dynamic split, gradient aggregation		
	model update	client scheduling		
	control message			
[40]	activation transmission	selective transmission	communication load ↓ 49.7 - 98.4%	2.6% - 7.3% ↓
	gradient transmission			
[41]	gradient transmission	gradient aggregation	training latency ↓, per-round latency ↓ 50%	maintained
[42]	activation transmission	dynamic split	training time ↓ 48%, energy consumption ↓ 62.8%	maximum 32% ↑
	gradient transmission			
	model update			
[43]	control message	client scheduling, client clustering	idle time ↓ 100ms	9% ↑
[44]	activation transmission	compression	communication load ↓ 45.9 - 95.9%	0.2% - 18.5% ↓
	gradient transmission	selective transmission		
[45]	activation transmission	compression	communication load ↓ 4×	maintained
	gradient transmission			
[46]	gradient transmission	gradient aggregation	training latency ↓, communication load ↓	maximum 5.8% ↑
	model update	aggregation interval adjust		

너지 소모를 감소시킨다. 실험 결과, 제안 기법은 모델 업데이트에 필요한 에너지가 충분할 때마다 업데이트를 수행하는 energy-aware selective updating 방식과 비교하여 정확도 감소를 0.5% 이내로 유지하였다[37]. 또한, 통신으로 인한 에너지 소모를 28.8%에서 80.5%까지 줄였다.

Lin et al.은 자원 제약 엣지 네트워크 환경에서 activation, gradient, 모델 가중치 전송으로 인해 발생하는 통신 오버헤드와 학습 지연을 줄이고자 하였다[38]. 이를 위해 제안 기법은 학습 수렴에 필요한 조건을 만족하면서 전체 학습 시간을 줄이도록 cut layer와 클라이언트 측 모델 집계 간격을 적응적으로 조정한다. 제안 기법은 각 엣지 장치의 연산 능력, 네트워크 처리량, 계층별 activation 및 gradient 크기, 모델 가중치 크기를 고려하여 cut layer와 클라이언트 측 모델 집계 간격을 결정한다. 이후 학습 수렴 조건을 만족하는 범위에서 전체 수렴 시간을 최소화하는 cut layer와 클라이언트 측 모델 집계 간격을 선택한다. 이를 통해 네트워크 및 연산 자원 상태에 따라 activation과 gradient의 전송 크기, 모델 가중치 전송 빈도를 함께 조절하여 통신 오버헤드와 학습 시간을 감소시킨다. 실험 결과, cut layer와 클라이언트 측 모델 집계 간격을 임의로 결정하는 기존 기법과 비교하여 정확도를 유지하면서, 수렴 속도를 최소 7.7배 향상시켰다.

Ao et al.은 자원 제약적인 무선 네트워크 환경에서 activation, gradient, 모델 가중치 전송과 동기화 및 조정 과정에서 발생하는 오버헤드로 인해 증가하는 학습 시간과 에너지 소모를 줄이고자 하였다[39]. 저자들은 이를 해결하고자 학습 지연과 에너지 소모의 가중합을 기준으로 cut layer와 학습 참여 클라이언트를 선택한다. cut layer 선택을 통해 클라이언트와 서버 간에 전송되는 activation 및 gradient 크기와 장치 측 연산량을 조절한다. 또한, 각 학습 라운드에서 클라이언트의 연산 성능과 무선 채널 상태를 고려하여 학습에 참여할 클라이언트를 선택함으로써, 학습 속도가 느린 클라이언트로 인해 서버와 다른 클라이언트가 다음 학습 단계로 넘어가지 못하고 대기하는 시간을 줄였다. 추가로, 저자들이 제안한 기법은 서버 측 backward propagation 이후 각 클라이언트의 gradient를 평균화하여 클라이언트로 전송하고, 연속으로 선택된 클라이언트에는 장치 측 모델을 다시 전송하지 않는다. 또한, 라운드 종료 후에는 채널 상태가 가장 좋은 클라이언트 하나만 모델 가중치를 서버로 전송한다. 이를 통해, activation 및 gradient 전송 크기, 모델 가중치 전송 빈도, 그리고 학습 과정에서 발생하는 대기 시간을 함께 줄여 학습 지연과 에너지 소모를 감소시킨다. 실험 결과, 저자들이 제안한 기법은 기존 연합학습 및 기존 순차적 분할학습 방식과 비슷한 정확도에 도달하면서, 동일 정확도 도달에 필요한 전체 학습 시간을 줄였다.

Mu et al.은 분할 연합 학습에서 activation 및 gradient 전송으로 인해 발생하는 통신 오버헤드를 줄이고자 하였다[40]. 이를 위해 저자들은 클라이언트 측 모델에 보조 네트워크를 추가하여, 클라이언트가 서버로부터 gradient를 수신하지 않고도 모델을 업데이트하는 방안을 제안했다. 클라이언트는 모든 미니배치마다 activation을 서버로 전송하지 않고, 미리 설정된 미니배치 간격마다 activation을 서버로 전송한다. 반면, activation을 전송하지 않는 미니배치에서는 보조 네트워크의 손실 함수를 기반으로 로컬 업데이트를 수행한다. 이를 통해 제안 기법은 activation 전송 횟수를 줄이고, gradient 전송을 제거하여 통신 오버헤드를 감소시킨다. 실험 결과, 제안 기법은 기존 분할 연합 학습과 비교하여, 약 2.6%에서 7.3%의 정확도 감소와 함께 통신량을 49.7%에서 98.4%까지 줄였다.

Lin et al.은 자원 제약적인 무선 엣지 네트워크 환경에서 gradient 전송으로 인해 발생하는 통신 오버헤드와 학습 지연을 줄이고자 하였다[41]. 이를 위해, 저자들은 서버 측 모델의 마지막 계층에서 일부 gradient를 먼저 집계한 뒤 backward propagation을 수행하는 방식을 제안했다. 서버는 집계 비율에 따라 각 엣지 장치의 미니배치 내 일부 gradient를 장치 간 평균화하고, 나머지 gradient는 개별적으로 유지한다. 이후, 집계된 gradient는 모든 엣지 장치로 전송하고, 집계되지 않은 gradient는 해당 엣지 장치로 각각 전송한다. 이를 통해 제안 기법은 backward propagation 과정에서 처리해야 하는 gradient 수와 엣지 장치로 전송해야 하는 gradient 크기를 줄여 통신 오버헤드와 학습 지연을 감소시킨다. 실험 결과, 제안 기법은 분할학습 및 분할 연합 학습과 유사한 정확도를 유지하면서, 목표 정확도 도달에 필요한 학습 시간을 줄였다.

Samikwa et al.은 서로 다른 IoT 기기를 사용하는 이기종 IoT 환경 및 데이터 분포가 다른 non-IID 환경에서 activation, gradient, 모델 가중치 전송으로 인해 발생하는 통신 오버헤드와 학습 지연을 줄이고 정확도 저하를 완화하는 방안을 제안했다[42]. 저자들은 각 학습 라운드에서 네트워크 처리량과 모델의 계층별 연산 시간을 측정하고, 이를 기반으로 IoT 장치별 모델 분할 지점을 동적으로 결정한다. 또한, 장치가 보유한 데이터 분포의 유사성을 반영하기 위해, 클라이언트 측 모델 계층의 유사도를 기반으로 학습 참여 장치를 클러스터링한다. 모델 분할 지점이 달라지면 IoT 장치와 서버에서 각각 연산되는 모델의 계층 수가 달라진다. 이에 따라 서버로 전송해야 하는 activation 크기, IoT 장치로 전송해야 하는 gradient 크기, 그리고 모델 가중치 크기도 함께 달라진다. 이를 통해 제안 기법은 IoT 장치의 자원 상태에 따라 모델 분할 지점을 동적으로 결정하여 activation, gradient 및 모델 가중치 전송으로 인한 통신 오버헤드와 학습 시간을 줄인다. 또한, 유사한 데이터 특성을 가진 장치들을 함께 학습시켜 non-IID 환경에서의 정확도를 개선한다. 실험 결과, 제안 기법은 non-IID 환경에서 기존 분할 연합 학습 및 연합학습과 비교하여 정확도를 최대 32% 향상시키고, IoT 장치의 에너지 소모를 최대 62.8% 감소시키며, 학습 시간을 최대 48% 줄였다.

Wazzeh et al.은 자원 제약적인 IoT 환경에서 클라이언트의 자원 이질성과 무작위 클라이언트 선택으로 인해 발생하는 동기화 및 조정 오버헤드와 학습 지연을 줄이고자 하였다[43]. 제안 기법은 자원 상태가 유사한 클라이언트를 클러스터로 그룹화하고, 각 학습 라운드에서 자원 가용성과 보유한 데이터 특성을 고려하여 학습 참여 클라이언트를 선택한다. 자원에 따라 장치를 클러스터링할 때 자원이 부족한 장치들은 제외시킨다. 이후, 자원 사용량 예측 모델을 통해 각 클라이언트의 동적 자원 사용량을 추정하고, 유전 알고리즘을 이용하여 고유 사용자 수, 데이터 샘플 수, 선택 이력, 처리 성능 차이를 고려해 각 클러스터에서 학습에 참여할 클라이언트 집합을 선택한다. 이를 통해, 제안 기법은 자원 상태가 유사한 클라이언트들이 함께 학습에 참여하도록 하여 학습 속도가 느린 클라이언트로 인해 발생하는 대기 시간을 줄이고 학습 과정에서 발생하는 동기화 및 조정 오버헤드를 완화한다. 실험 결과, 무작위 클라이언트 선택을 사용하는 기존 클러스터 기반 분할 연합 학습보다 정확도를 약 9% 향상시켰으며, 평균 대기 시간을 약 100 ms 줄였다.

Ayad et al.은 IoT 환경에서 분할학습 적용 시 activation 및 gradient 전송으로 인해 발생하는 통신 오버헤드를 줄이고자 하였다[44]. 이를 위해 제안 기법은 오토인코더를 이용하여 activation을 압축하고 적응형 임계값을 기반으로 gradient 전송 여부를 선택적으로 결정한다. 클라이언트는 cut layer에서 생성된 activation을 인코더를 통해 더 작은 잠재 벡터로 변환한 뒤 서버로 전송하고, 서버는 디코더를 통해 이를 복원하여 서버 측 학습을 수행한다. 서버는 forward propagation 이후 계산된 손실이 임계값보다 큰 경우에만 gradient를 클라이언트로 전송하고, 손실이 임계값보다 작은 경우에는 gradient 전송을 생략한다. 이를 통해 제안 기법은 activation 전송 크기와 gradient 전송 횟수를 함께 줄여 통신 오버헤드를 감소시킨다. 실험 결과, 저자들은 약 0.2%에서 18.5%의 정확도 감소와 함께 통신량을 45.9%에서 95.9%까지 줄였다.

Mudvari et al.은 IoT 및 모바일 엣지 환경에서 분할학습 적용 시 activation 및 gradient 전송으로 인해 발생하는 통신 오버헤드와 학습 지연을 줄이고자 하였다[45]. 저자들은 압축-복원 모듈을 이용하여 cut layer에서 생성된 activation을 압축하고, 학습 과정에서 압축 수준을 적응적으로 조정하는 방안을 제시했다. 클라이언트는 cut layer까지 forward propagation을 수행한 뒤 생성된 activation을 압축하여 서버로 전송하고, 서버는 이를 복원한 뒤 서버 측 모델의 forward propagation과 backward propagation을 수행한다. 이후 서버는 앞서 적용된 activation의 압축 수준에 따라 크기가 압축된 gradient를 클라이언트로 전송한다. 이를 통해 학습 초기에는 높은 압축률을 적용하여 통신량을 줄이고, 학습 후반에는 압축 수준을 완화하여 정확도 저하를 줄인다. 따라서, 제안 기법은 activation 전송 크기와 gradient 전송 크기를 함께 줄여 통신 오버헤드와 학습 지연을 감소시킨다. 실험 결과, 일반적인 분할학습 방식과 비교하여 정확도 저하 없이 학습 과정의 통신량을 약 4배 줄였다.

Oh et al.은 분할 연합 학습에서 gradient 전송과 클라이언트 측 모델 집계로 인해 발생하는 통신 오버헤드와 학습 지연을 줄이고자 하였다[46]. 이를 위해 제안 기법은 서버에서 클라이언트로 전송되는 gradient를 평균화하고, 클라이언트 측 모델 집계 간격을 조정한다. 서버는 클라이언트별로 계산된 gradient와 여러 클라이언트의 평균 gradient를 클라이언트들에게 함께 전송한다. 이때, 클라이언트별로 서로 다른 gradient를 각각 전송하는 과정에서 발생하는 다운링크 통신 오버헤드를 줄인다. 또한, 클라이언트 측 모델 집계 간격을 조정하여 클라이언트가 모델 가중치를 서버로 업로드하는 과정에서 발생하는 업링크 통신 비용을 조절한다. 한편, 제안 기법은 클라이언트가 업로드한 activation을 서버에서 조합하여 서버 측 학습에 활용함으로써 정확도를 개선한다. 이를 통해 gradient 전송, 모델 집계 과정에서 발생하는 통신 오버헤드, 그리고 학습 지연을 감소시키고 정확도를 개선한다. 실험 결과, 기존 분할 연합 학습 기법과 비교하여 정확도를 최대 5.8% 향상됐으며, gradient 전송과 모델 집계 과정에서 발생하는 통신량과 학습 지연을 줄였다.

V. 결 론

본 논문에서는 IoT 환경에서 분할 연합 학습을 적용할 때 핵심 제약으로 작용하는 통신 오버헤드 문제를 중심으로 관련 연구들을 조사하였다. 기존 연구 동향 논문은 통신 오버헤드를 smashed data transmission, gradient transmission, model update,

control message의 네 가지 문제 유형으로 구분하고, 이를 완화하기 위한 해결 전략을 분류하며 관련 기법이 정확도에 미치는 영향을 비교하였다. 본 논문은 동일한 네 가지 문제 유형을 바탕으로 각 연구가 다루는 세부적인 통신 오버헤드 문제와 해결 전략을 체계적으로 비교하였다. 또한, 각 연구의 주요 성능 결과와 정확도에 미치는 영향을 정리하였다. 다만, 각 연구는 서로 다른 항목을 평가하고 결과를 서로 다른 단위로 제시하므로, 해결 전략 간 성능을 직접 비교하기에는 한계가 있다.

관련 연구들은 선택적 전송, 압축, 동적 분할, gradient 집계, 클라이언트 선택 및 클러스터링, 모델 집계 간격 조정과 같은 다양한 전략을 제안하였다. 조사 결과, 기존 연구들은 주로 activation transmission 및 gradient transmission 문제 최적화에 집중되어 있었으며, model update와 control message 문제를 직접적으로 다루는 연구는 상대적으로 부족하였다. 따라서, IoT 환경에서 분할 연합 학습을 효과적으로 적용하기 위해서는 activation 및 gradient 전송뿐만 아니라, 모델 가중치 송수신과 학습 과정 조율에서 발생하는 통신 오버헤드를 함께 고려하는 연구가 필요하다.

향후 연구로는 model update와 control message 문제를 중심으로, 통신 오버헤드 최적화 방법을 연구할 계획이다. 또한, 실제 IoT 환경에서는 네트워크 상태와 장치의 자원 상태가 시간에 따라 변할 수 있으므로, 이를 반영하여 activation 전송, gradient 전송, 모델 가중치 송수신, 학습 과정 조율에 따른 통신 오버헤드를 함께 줄일 수 있는 방법을 연구하고자 한다. 이를 통해 자원 제약적인 IoT 환경에서 분할 연합 학습의 통신 효율성을 향상시키고, 실제 서비스 환경에서의 적용 가능성을 높이고자 한다.

References

[1] I. Siam, H. Ahn, L. Liu, S. Alam, H. Shen, Z. Cao, N. Shroff, B. Krishnamachari, M. Srivastava, and M. Zhang, "Artificial intelligence of things: A survey," A*CM Transactions on Sensor Networks*, Vol. 21, No. 1, Article 9, pp. 1 - 75, Jan. 2025.

[2] E. Dritsas, M. Trigka, "Federated learning for iot: A survey of techniques, challenges, and applications," *Journal of Sensor and Actuator Networks*, Vol. 14, No. 1, pp. 1 - 25, Jan. 2025.

[3] H. Fu, J. Rao, F. Deng, Y. Wang, B. Zhao, Z. Liu, H. Guan, P. H. Malinowski, and L. Xu, "AIoT: Artificial intelligence and the internet of things for monitoring and prognosis of systems and structures," *IEEE Transactions on Instrumentation and Measurement*, Vol. 74, Article 9700232, pp. 1 - 32, Jun. 2025.

[4] Q. Duan, S. Hu, R. Deng, and Z. Lu, "Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: State-of-the-art and future directions," Sensors, Vol. 22, No. 16, pp. 1 - 32, Aug. 2022.

[5] A. Arzovs, J. Judvaitis, K. Nesenbergs, and L. Selavo, "Distributed learning in the iot - Edge - Cloud continuum," *Machine Learning and Knowledge Extraction,* Vol. 6, No. 1, pp. 283 - 315, Feb. 2024.

[6] Y. Gao, M. Kim, C. Thapa, S. Abuadbba, Z. Zhang, S. A. Camtepe, H. Kim, and S. Nepal, "Evaluation and optimization of distributed machine learning techniques for Internet of Things," *IEEE Transactions on Computers*, Vol. 71, No. 10, pp. 2538 - 2552, Oct. 2022.

[7] X. Chen, J. Li, and C. Chakrabarti, "Energy and loss-aware selective updating for splitfed learning with energy harvesting-powered devices," *Journal of Signal Processing Systems*, Vol. 94, No. 10, pp. 961 - 975, Oct. 2022.

[8] B. Radovič, M. Canini, and V. Pejović, "Review and comparative evaluation of resource-adaptive collaborative training for heterogeneous edge devices," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, Vol. 10, No. 1, pp. 1 - 35, Mar. 2025.

[9] E. Samikwa, A. Di Maio, and T. Braun, "ARES: Adaptive resource-aware split learning for Internet of Things," *Computer Networks*, Vol. 218, Art. No. 109380, pp. 1 - 13, Dec. 2022.

[10] D. Naseh, A. Bozorgchenani, S. S. Shinde, and D. Tarchi, "Unlocking distributed intelligence: A comprehensive survey on federated split learning's evolution, challenges, and future frontiers," *Computer Networks*, Vol. 285,

Article 112402, pp. 1 - 32, May 2026.

[11] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492,* pp. 1 - 10, Oct. 2016.

[12] C. Thapa, M. A. P. Chamikara, and S. A. Camtepe, "Advancements of federated learning towards privacy preservation: From federated learning to split learning," *arXiv preprint arXiv:2011.14818,* Nov. 2020.

[13] J. Hu, Y. Liang, Y. Chen, G. Liu, W. Chen, and L. Duan, "Performance optimization of split federated learning in heterogeneous edge computing environments," *IEEE Transactions on Industrial Informatics*, Vol. 22, No. 1, pp. 154 - 164, Jan. 2026.

[14] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, Vol. 116, No. 15, pp. 1 - 8, Aug. 2018.

[15] M. Le, T. Huynh-The, T. Do-Duy, T. V. Vu, W. J. Hwang, and Q. V. Pham, "Applications of distributed machine learning for the internet-of-things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, Vol. 27, No. 2, pp. 1053 - 1100, Apr. 2023.

[16] B. Radovič, M. Canini, S. Horváth, V. Pejović, and P. Vepakomma, "Towards a unified framework for split learning," in *Proc. of the 5th Workshop on Machine Learning and Systems*, Rotterdam, Netherlands, pp. 183 - 191, Apr. 2025.

[17] S. Oh, J. Park, P. Vepakomma, S. Baek, R. Raskar, M. Bennis, and S.-L. Kim, "LocFedMix-SL: Localize, federate, and mix for improved scalability, convergence, and latency in split learning," in *Proc. of the ACM Web Conference 2022*, Lyon, France, pp. 3347 - 3357, Apr. 2022.

[18] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," in *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 36, No. 8, Virtual, pp. 8485 - 8493, Jun. 2022.

[19] G. S. Hukkeri, R. H. GOUDAR, G. M. DHANANJAYA, V. N. RATHOD, and S. ANKALAKI, "A comprehensive survey on split-fed learning: Methods, innovations, and future directions," *IEEE Access*, Vol. 13, pp. 46312 - 46333, Mar. 2025.

[20] W. Zhang, D. Zhan, H. Yu, L. Zhang, B. Zhao, X. Du, and Z. Tian, "Distributed machine learning for next-generation communication networks: A survey on privacy, fairness, efficiency, and trade-offs," *Information Fusion*, Vol. 126, Article 103657, pp. 1 - 28, Feb. 2026.

[21] H. HAFI, B. BRIK, P. A. FRANGOUDIS, A. KSENTINI, and M. BAGAA, "Split Federated Learning for 6G Enabled-Networks: Requirements, Challenges, and Future Directions," *IEEE Access*, Vol. 12, pp. 9890 - 9930, Jan. 2024.

[22] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, Vol. 6, No. 3, pp. 4921 - 4934, Jun. 2019.

[23] S. Liu, B. Guo, C. Fang, Z. Wang, S. Luo, Z. Zhou, and Z. Yu, "Enabling resource-efficient AIoT system with cross-level optimization: A survey," *IEEE Communications Surveys & Tutorials,* Vol. 26, No. 1, pp. 389 - 427, Sept. 2024.

[24] I. Nikoloska and N. Zlatanov, "Data selection scheme for energy efficient supervised learning at IoT nodes," *IEEE Communications Letters*, Vol. 25, No. 3, pp. 859 - 863, Mar. 2021.

[25] H. Yin, Z. Wang, and N. K. Jha, "A hierarchical inference model for internet-of-things," *IEEE Transactions on Multi-Scale Computing Systems,* Vol. 4, No. 3, pp. 260 - 271, Sept. 2018.

[26] Y. Wang, C. Yang, S. Lan, L. Zhu, and Y. Zhang, "End-Edge-Cloud collaborative computing for deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, Vol. 26, No. 4, pp. 2647 - 2683,

Apr. 2024.

[27] A. P. Behera, P. Daubaris, I. Bravo, J. Gallego, R. Morabito, J. Widmer, and J. P. Champati, "Exploring the boundaries of on-device inference: When tiny falls short, go hierarchical," *IEEE Internet of Things Journal*, Vol. 12, No. 18, pp. 37456 - 37470, Sept. 2025.

[28] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, pp. 1 - 9, Dec. 2018.

[29] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., "Advances and open problems in federated learning," Foundations and Trends in Machine Learning, Vol. 14, No. 1-2, pp. 1 - 210, Jun. 2021.

[30] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Transactions on Communications*, Vol. 68, No. 2, pp. 1146 - 1159, Feb. 2020.

[31] A. Li, J. Sun, X. Zeng, M. Zhang, H. Li, and Y. Chen, "FedMask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking," in *Proc. of the 19th ACM Conference on Embedded Networked Sensor Systems*, Coimbra, Portugal, pp. 42 - 55, Nov. 2021.

[32] J. Park, S. Samarakoon, A. Elgabli, J. Kim, M. Bennis, and S.-Lyun Kim, "Communication-efficient and distributed learning over wireless networks: Principles and applications," *Proceedings of the IEEE*, vol. 109, No. 5, pp. 796 - 819, May 2021.

[33] S. Rahman, J. Arora, S. Pal, and C. Karmakar, "DR-SFL: Dynamic and resource-aware split federated learning for semantic-enabled 6G industrial iot systems," *IEEE Communications Standards Magazine*, Vol. 10, No. 1, pp. 1 - 8, Jun. 2026.

[34] J. Lee, M. Seif, J. Cho, and H. V. Poor, "Exploring the privacy-energy consumption tradeoff for split federated learning," *IEEE Network*, vol. 38, no. 6, pp. 388 - 395, Nov. 2024.

[35] D. J. Han, H. I. Bhatti, J. Lee, and J. Moon, "Accelerating federated learning with split learning on locally generated losses," in P*roc. of the ICML 2021 workshop on federated learning for user privacy and data confidentiality*, Vol. 2, No. 3, Virtual, pp. 1 - 9, Sept. 2021.

[36] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," *arXiv preprint arXiv:1909.09145*, pp. 1 - 5, Sept. 2019.

[37] X. Chen, J. Li, and C. Chakrabarti, "Communication and computation reduction for split learning using asynchronous training," in *Proc. of the 2021 IEEE Workshop on Signal Processing Systems*, Coimbra, Portugal, pp. 76 - 81, Oct. 2021.

[38] Z. Lin, G. Qu, W. Wei, X. Chen, and K. K. Leung, "AdaptSFL: Adaptive split federated learning in resource-constrained edge networks," *IEEE Transactions on Networking*, Vol. 33, No. 6, pp. 2993 - 3008, Dec. 2025.

[39] H. Ao, H. Tian, and W. Ni, "Federated split learning for edge intelligence in resource-constrained wireless networks," I*EEE Transactions on Consumer Electronics,* Vol. 71, No. 2, pp. 4451 - 4463, May 2025.

[40] Y. Mu and C. Shen, "Communication and storage efficient federated split learning," in *Proc. of the IEEE International Conference on Communications*, Rome, Italy, pp. 2976 - 2981, Oct. 2023.

[41] Z. Lin, G. Zhu, Y. Deng, X. Chen, Y. Gao, K. Huang, and Y. Fang, "Efficient parallel split learning over resource-constrained wireless edge networks," *IEEE Transactions on Mobile Computing*, Vol. 23, No. 10, pp. 9224 - 9239, Oct. 2024.

[42] E. Samikwa, A. Di Maio, and T. Braun, "DFL:

Dynamic federated split learning in heterogeneous iot," *IEEE Transactions on Machine Learning in Communications and Networking*, Vol. 2, pp. 733 - 752, Jun. 2024.

[43] M. Wazzeh, M. Arafeh, H. Sami, H. Ould-Slimane, C. Talhi, A. Mourad, and H. Otrok, "CRSFL: Cluster-based resource-aware split federated learning for continuous authentication," *Journal of Network and Computer Applications*, Vol. 231, Article 103987, pp. 1 - 17, Aug. 2024.

[44] A. Ayad, M. Renner, and A. Schmeink, "Improving the communication and computation efficiency of split learning for iot applications," in *Proc. of the 2021 IEEE Global Communications Conference*, Madrid, Spain, pp. 1 - 6, Dec. 2021.

[45] A. Mudvari, A. Vainio, I. Ofeidis, S. Tarkoma, and L. Tassiulas, "Adaptive compression-aware split learning and inference for enhanced network efficiency," *ACM Transactions on Internet Technology*, Vol. 24, No. 4, pp. 1 - 26, Nov. 2024.

[46] S. Oh, H. Nam, J. Park, P. Vepakomma, R. Raskar, M. Bennis, and S. L. Kim, "Mix2SFL: Two-way mixup for scalable, accurate, and communication-efficient split federated learning," *IEEE Transactions on Big Data*, Vol. 10, No. 3, pp. 238 - 248, Jun. 2024.

정 한 길 (Han-Gil Jeong)



2026년 2월 : 강원대학교 컴퓨터과학 졸업

2026년 2월 - 현재 : 강원대학교 데이터사이언스학과 석사 과정

<관심분야> 분할 연합 학습, 시스템 최적화

김 종 욱 (Jong-Wouk Kim)



2019년 2월 : 강원대학교 컴퓨터과학 졸업

2021년 2월 : 강원대학교 컴퓨터과학 석사

2021년 3월 - 현재 : 강원대학교 컴퓨터과학 박사과정

<관심분야> 네트워크 보안, 악성코드 분석, 딥러닝

최 미 정 (Mi-Jung Choi)



1998년 2월 : 이화여자대학교 공학사 졸업

2000년 2월 : 포항공과대학교 공학석사

2004년 2월 : 포항공과대학교 공학박사

2004년 - 2005년 : 프랑스 INRIA 연구소 박사 후 연구원

2005년 - 2006년 : 캐나다 워터루대학 박사 후 연구원

2008년 - 현재 : 강원대학교 컴퓨터학부 컴퓨터공학 전공 교수

<관심분야> 네트워크 관리, 네트워크 보안

KNOM Review '26-01 Vol.29 No.01
https://doi.org/10.22670/knom.2026.29.1.59

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 가능함을 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 지능평면 기술 개발, 50%)과 삼성전자의 지원(50%)을 받아 수행되었습니다.
◆ First Author : Pohang University of Science and Technology Institute of Artificial Intelligence, obiwan96@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. 서 론

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, 인텐트 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 자동화에 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.

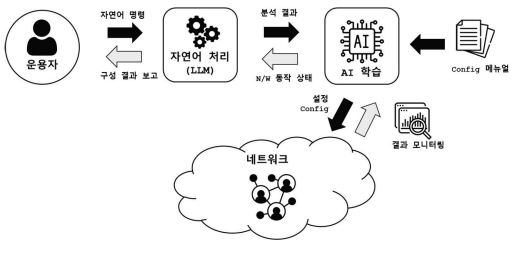


그림 1. 의도 기반 네트워킹-네트워크 설정 개요
Fig. 1. IBN-Configuration Outline

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 명령으로 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. Dzeparoska 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 파인튜닝하여 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 함수를 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

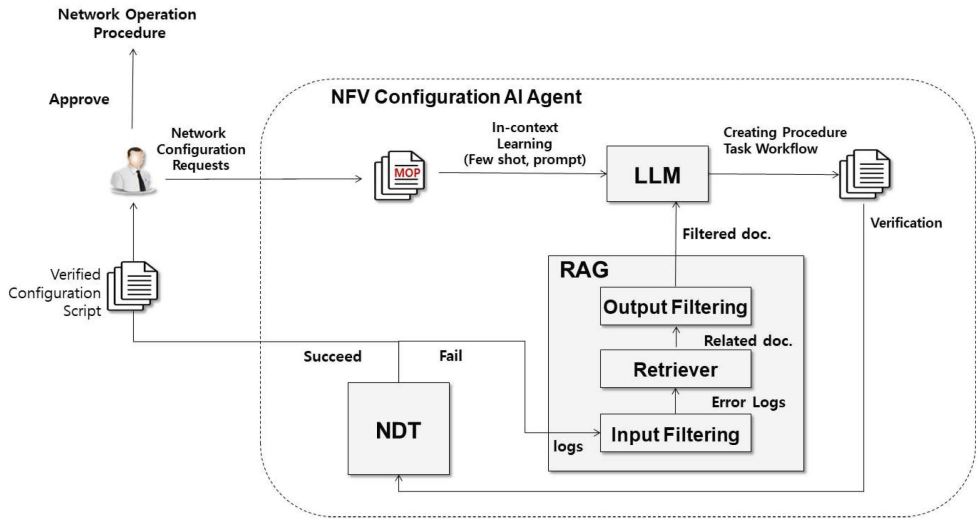


그림 2. LLM 기반 NFV 구성 에이전트 구조도
Fig. 2. LLM-based NFV Configuration AI Agent Structure

에 관한 도큐먼트 (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 테스팅 모듈이 LLM의 응답에서 코드를 추출하고 NDT와 통신하여 실행함으로서 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 전달함으로서 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 제안하는 AI Agent는 LLM이 문제를 인지하고 자가 수정할 수 있도록 추가적인 동작을 지시함으로서 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 전달함으로서 이를 고칠 수 있도록 한다.

우리는 해당 과정을 위해 사전 연구 [10]에서 테스트 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 도큐먼트에 없을 수 있으며, 로그와 가장 유사한 도큐먼트를 찾는 것이 목표이다. 이 때문에 기존 QA 중심 RAG를 그대로 적용하기에는 한계가 있으며, 본 연구에서는 오류 해결을 위해 문서 구조를 유지하면서 의

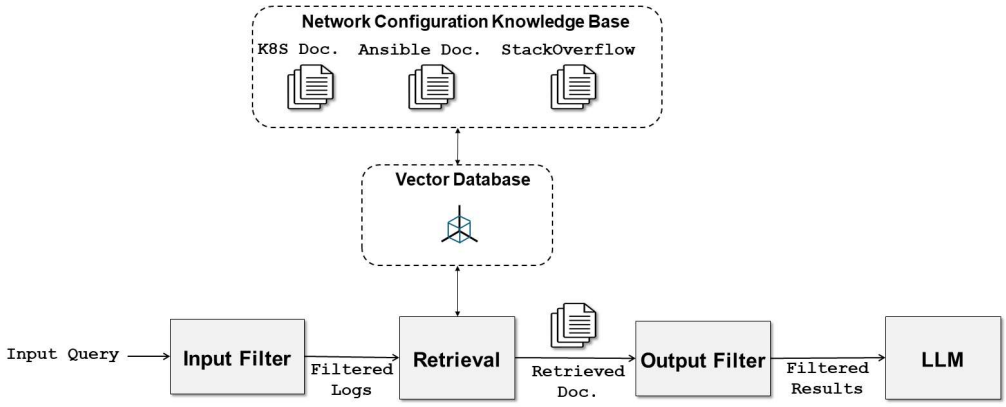


그림 3. 입/출력 필터 포함 RAG 구조도
Fig. 3. RAG Structure with Input/Output Filter

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 지식베이스 내 모든 문서의 제목을 벡터화한다. Stack Overflow 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 실험하는 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 거리가 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 희귀도를 정량화하기 위한 **Log-TF-IDF**를 제안하였다. 전통적 TF-IDF가 문서 내 단어의 중요도를 측정한다면, Log-TF-IDF는 이를 **정상 로그 내 이벤트 희귀도**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 w가 얼마나 자주 등장하는지를 수치화

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

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

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

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.

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

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.

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

이를 통해 계산되는 Log-TF-IDF는 다음과 같다.

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

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 메세지와 logarithm을 구별하기 위해 로그 메시지를 뜻하는 로그는 첫글자를 대문자로 표기함.

```
ErrorList ← []
for Log ∈ InputLogs do
    LogEvent, IsNew ← LogParser(Log)
    if IsNew then
        Log_TF_IDF ← CalculateTF_IDF(LogEvent)
    else
        Log_TF_IDF ← TF_IDF_Dict[LogEvent]
    end if
    if Log_TF_IDF ≥ TF_IDF_Threshold then
        Append [Log] to ErrorList
    end if
end for
if ErrorList is Empty then
    ErrorList ← InputLogs
end if
InputVector ← EmbeddingModel(ErrorList)
MaxSim ← 0
for (Vector, Doc) ∈ VectorKB do
    sim ← VectorSim(Vector, InputVector)
    if sim > MaxSim then
        MaxSim ← sim
        QueriedDoc ← Doc
    end if
end for
Related, FilteredOutput ← JudgeLLM(QueriedDoc)
if Related and MaxSim > Sim_Threshold then
    return FilteredOutput
end if
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 크롤링하여 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 메타데이터를 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.

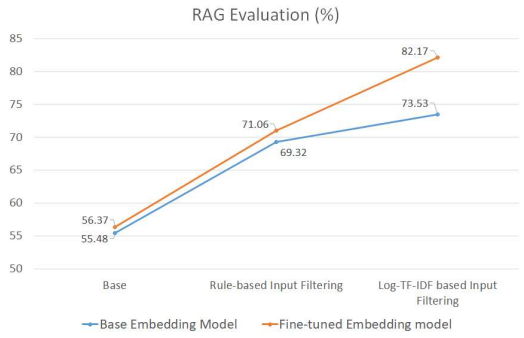


그림 4. RAG 검증 실험 결과
Fig. 4. RAG Evaluation Test Results

2) RAG 적용 결과 검증

이어서 제안한 RAG 모델을 통해 AI Agent가 자가 수정이 가능한지 확인하여보았다. 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

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 높은데, 이는 RAG 실험 과정에서도 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 과정 등을 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 Dzeparoska, 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 Dzeparoska, 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

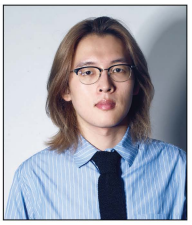
[17] Mojan Javaheripi, Sebastien Bubeck, Marah Abdin, Jyoti Aneja, Caio Cesar T. Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, Suriya Gunasekar, Yin Tat Lee, Yuanzhi Li, Anh Nguyen, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Adam Taumann Kalai, 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 크립토블록체인연구센터 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

한국통신학회 통신망운용관리연구회
KNOM Review Vol.29/ No.1

ISSN : 2287-1543(Online)
2026년 8월 31일 인쇄
2026년 8월 31일 발행

발행인/ 김우태
편집인/ 김명섭 편집위원장
발행처/ 한국통신학회 통신망운용관리연구회
서울시 서초구 서초동 1330-8 현대기림오피스텔 1504동 6호
전 화 : 02-3453-5555
홈페이지 : www.knom.or.kr/knom-review/