NetLLM: Adapting Large Language Models for Networking

Duo Wu[1], Xianda Wang[1], Yaqi Qiao[1], Zhi Wang[2], Junchen Jiang[3], Shuguang Cui[1], Fangxin Wang[1*]
[1]SSE and FNii, The Chinese University of Hong Kong, Shenzhen
[2]SIGS, Tsinghua University [3]The University of Chicago

ABSTRACT

Many networking tasks now employ deep learning (DL) to solve complex prediction and optimization problems. However, current design philosophy of DL-based algorithms entails intensive engineering overhead due to the manual design of deep neural networks (DNNs) for different networking tasks. Besides, DNNs tend to achieve poor generalization performance on unseen data distributions/environments.

Motivated by the recent success of large language models (LLMs), this work studies the LLM adaptation for networking to explore a more sustainable design philosophy. With the powerful pre-trained knowledge, the LLM is promising to serve as the foundation model to achieve "one model for all tasks" with even better performance and stronger generalization. In pursuit of this vision, we present NetLLM, the first framework that provides a coherent design to harness the powerful capabilities of LLMs with low efforts to solve networking problems. Specifically, NetLLM empowers the LLM to effectively process multimodal data in networking and efficiently generate task-specific answers. Besides, NetLLM drastically reduces the costs of fine-tuning the LLM to acquire domain knowledge for networking. Across three networking-related use cases - viewport prediction, adaptive bitrate streaming and cluster job scheduling, we showcase that the NetLLM-adapted LLM significantly outperforms state-of-the-art algorithms.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Computing methodologies** → **Supervised learning**; **Reinforcement learning**;

KEYWORDS

Deep Learning, Network Optimization, Video Streaming, Job Scheduling, Large Language Model Adaptation

ACM Reference Format:
Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, Fangxin Wang. 2024. NetLLM: Adapting Large Language Models for Networking. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3651890.3672268

*Fangxin Wang is the corresponding author: wangfangxin@cuhk.edu.cn.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08.
https://doi.org/10.1145/3651890.3672268

1 INTRODUCTION

1.1 The Main Roadmap so far

Over the past decades, *rule-based algorithms* built on handcrafted control rules have played an important role in optimizing network systems [6, 8, 32, 107]. For instance, Copa [6] adjusts sending rates for congestion control based on measured queueing delay, while PANDA [53] switches video streaming bitrates based on heuristically estimated bandwidth. However, these algorithms heavily rely on *rule engineering*, which involves intensive human efforts to devise, implement and validate the control rules for network optimization [11, 62, 63, 66].

In recent years, the advancements of deep learning have prompted extensive research into *learning-based algorithms* for networking. These algorithms design and train deep neural networks (DNNs) with supervised learning (SL) [95] or reinforcement learning (RL) [88] techniques to automatically discover networking solutions, thus eliminating the need of rule engineering. Specifically, SL is widely adopted to train DNNs for prediction tasks in networking, such as traffic classification [54, 73] and bandwidth prediction [9, 64]. On the flip side, RL is commonly employed to solve decision-making problems in networking, including congestion control [1, 106], adaptive bitrate streaming (ABR) [44, 62] and cloud cluster job scheduling (CJS) [63, 78]. Thanks to the strong capability of DNNs in function approximation, learning-based algorithms have demonstrated significant improvement over handcrafted rule-based algorithms.

Despite their promising potential, existing learning-based algorithms still suffer from two key limitations:

- **High model engineering costs.** It has been shown that the design of DNN architecture is crucial to the final performance [67, 83]. Therefore, the focus of learning-based algorithms has shifted from rule engineering to *model engineering*. Their success is heavily dependent on engineering DNN models for the target networking tasks, which, however, can be difficult and labor-intensive due to the complex structures of DNNs [66]. To make things worse, the diversity of networking tasks also prevents sharing the same DNN model across different tasks. This necessitates designing specialized DNNs for different tasks (i.e., *one model for one task*), thus further increasing the engineering costs. Although some recent works attempt to introduce structured Transformer [94] into model design, they still necessitate manual tuning of the Transformer architecture (e.g., the number of attention blocks and attention heads) [99], or even the design of specialized tokenization scheme [36, 65] and attention mechanism [36, 57], thus leading to high engineering costs.
- **Low generalization.** DNNs trained on specific data distributions/environments may struggle to perform well or even worse than conventional rule-based algorithms on unseen data distributions/environments [105]. For example, an ABR model trained on smooth network conditions often achieves poor performance on

network environments with dynamic bandwidth fluctuations [44]. The lack of generalization can ultimately hinder the widespread deployment of learning-based algorithms in practice [103, 106], as network operators will suspect their superiority over the incumbent rule-based algorithms in production environments.

1.2 New Opportunities and Challenges

Utilizing a single generic model for different tasks has been recognized as a significant approach to mitigate the costs of handcrafting specialized DNNs for each task and enhance generalization [82]. This is exemplified by the recent popular large language models (LLMs) such as ChatGPT [72], Falcon [77] and Llama2 [92] in the field of natural language processing (NLP). With billions of parameters pre-trained on massive data to absorb extensive knowledge, LLMs have demonstrated extraordinary capacity in conversations, reasoning and text generation in NLP [96]. What's more, they also exhibit emergent abilities that were not explicitly programmed into them during pre-training, such as planning, pattern mining, problem solving and generalization to unseen conditions [102, 112]. These abilities have been proven to be transferable to other domains, including robotics [23], chip design [59], and protein structure prediction [55]. For instance, researchers have shown that LLMs can generate goal-oriented plans for robotic control, adjusting plans in response to environment changes, and generalize to previously unseen operating environments [112].

Motivated by these inspiring outcomes, we believe that LLMs can serve as foundation models for networking, as many networking tasks can also benefit from their emergent abilities. In the context of ABR, for example, the planning ability of LLMs can be utilized for better bitrate decisions to optimize the video streaming sessions based on the changing network conditions. Furthermore, their generalization capability can be harnessed to generalize across diverse network environments. Therefore, we envision LLM as the key to achieving *one model for all tasks*, with little handcraft costs and strong generalization. We try to answer the following key question: *can we embrace the era of LLMs and adapt LLMs to solve various networking tasks efficiently and effectively?*

Unfortunately, as revealed by our analysis in §3, the adaptation of LLMs for networking faces the following challenges.

- **Large input modality gap.** In networking tasks, various information observed from the system is collected as inputs for networking algorithms. However, the modalities of these inputs differ significantly from plain text, i.e., the native input modality supported by LLMs[1] [92, 108]. For example, in ABR, network throughputs and delay are often collected for bitrate decision [62], which exhibit unique time-varying data patterns typically not found in natural text. This discrepancy prevents LLMs from effectively processing the input information of networking tasks.
- **Inefficiency of answer generation.** LLMs generate answers using a language modeling (LM) head to predict words (tokens) *one by one* (see Figure 1) [46]. While this approach is well-suited in NLP, it presents several drawbacks in the networking domain. First, LLMs are prone to *hallucination* due to the inherent uncertainty of token prediction [41, 49]. Their generated answers for networking tasks may seem correct but physically invalid (e.g., a nonexistent bitrate for video download in ABR), which can eventually impair the reliability of network systems. Second, since tokens are predicted one at a time, LLMs often require multiple inferences to generate a complete answer, thus incurring high answer generation latency (i.e., the time to generate a complete answer). Consequently, LLMs may fail to quickly generate answers to respond to the system changes (e.g., switching to a lower bitrate when network bandwidth becomes scarce).
- **High adaptation costs.** The large domain gap between networking and NLP necessitates fine-tuning LLMs to acquire domain-specific knowledge for effective adaptation. However, the adaptation costs can be prohibitively high, especially when fine-tuning LLMs for decision-making tasks (e.g., ABR [44] and CJS [63]) where RL is employed to solve the system optimization problems. Specifically, RL-based decision-making tasks require the active interaction between LLMs and environments (e.g., network environments with varying bandwidth or workload patterns) to collect experiences for performance optimization [63, 103]. Due to the large parameter size of LLMs, the interaction process can be excessively time-consuming, introducing a large amount of additional training time. What's worse, the adaptation costs can further increase if fine-tuning the full parameters of LLMs, which is known to be resource-intensive [55, 59].

[1] Although multimodal LLMs have emerged recently (e.g., GPT4 [71]), their supported input modalities are still limited (mainly vision or audio), hindering directly applying them to process networking task inputs.

1.3 Design and Contributions

To overcome the aforementioned challenges, this work proposes `NetLLM`, *the first framework that efficiently adapts LLMs for networking*. `NetLLM` stands out as the first to take LLMs the extra mile in context of networking and provides a coherent design to utilize their powerful capabilities to generalize to various tasks with low efforts. It enables the efficient utilization of a single LLM (e.g., Llama2 [92]) as the foundation model without any modifications to tackle a wide range of networking tasks and meanwhile achieves enhanced performance. To accomplish this, `NetLLM` designs a multimodal encoder to enable the LLM to process multimodal data in networking and implements a networking head to improve the efficiency of answer generation. Furthermore, while fine-tuning the LLM and these two modules on the target task to acquire domain knowledge is necessary, `NetLLM` designs an efficient data-driven low-rank networking adaptation (DD-LRNA) scheme to drastically reduce the fine-tuning costs. Specifically, `NetLLM` incorporates the following three core design modules to efficiently adapt LLMs for networking.

- **Multimodal encoder.** `NetLLM` designs an efficient multimodal encoder at the input side of the LLM to effectively process the multimodal input information of networking tasks. The goal of this module is to automatically project task inputs to the same feature space as language tokens, so that the LLM can understand and utilize these inputs for task solving. To achieve this, it first uses modality-specific feature encoders to extract features from raw inputs of various modalities involved in networking. It then leverages trainable layers to project these features into token-like embedding vectors, which can be directly fed into the LLM for effective processing.

- **Networking head.** To enable efficient answer generation, NetLLM removes the default LM head used by the LLM for token prediction. Instead, it introduces various networking heads at the output side of the LLM to generate answers for specific networking tasks. Each networking head is essentially a lightweight trainable projector that maps the output features of the LLM directly into task-specific answers. In other words, they eliminate token prediction and allow direct answer generation from the valid range of possible answers (e.g., selecting a bitrate from candidate options in ABR). This enables the LLM to generate a valid answer in a single inference, thus ensuring its reliability for networking and significantly reducing generation latency.
- **DD-LRNA.** To reduce the fine-tuning costs, NetLLM designs a DD-LRNA scheme for the LLM to efficiently acquire domain knowledge for networking. Specifically, DD-LRNA incorporates a data-driven adaptation pipeline to adapt the LLM for both prediction and decision-making tasks. In particular, for decision-making tasks, it employs the efficient data-driven RL technique [3, 79, 106] to eliminate the time-consuming interaction between the LLM and environments. It collects an experience pool as training dataset with existing networking algorithms and fine-tunes the LLM over such dataset in the data-driven manner. Besides, inspired by the advanced parameter-efficient fine-tune technique [21, 29], DD-LRNA introduces a set of additional trainable low-rank matrices for the LLM to learn networking knowledge. Since the low-rank matrices only account for 0.31% of the total parameters, the fine-tune costs are greatly reduced, with the reduction of 60.9% GPU memory and 15.1% training time.

NetLLM has the following important properties. i) *Compatibility*: It is a generic framework that can be applied to adapt different LLMs for various networking tasks. ii) *Reliability*: It addresses the hallucination issue and ensures the LLM generated answers to be always valid. iii) *Efficiency*: The DD-LRNA scheme significantly reduces the costs of fine-tuning LLM to learn domain knowledge and the networking heads also reduce the answer generation latency. iv) *Transferability*: NetLLM offers an effective solution for utilizing LLMs to solve prediction and decision-making tasks at low costs. These two types of tasks, in fact, span across diverse domains such as medicine [17], finance [52], and telecommunication [111]. Hence, despite its original focus on networking, NetLLM holds the potential to be applied and transferred to other fields.

We have implemented NetLLM[2] for three networking tasks: viewport prediction (VP) [85] for immersive video streaming, adaptive bitrate streaming (ABR) [103], and cluster job scheduling (CJS) [63]. We believe these representative tasks cover the main input modalities of networking problems and span from prediction tasks to decision-making tasks (§3). Through extensive trace-driven simulation and real-world tests, we demonstrate the effectiveness of NetLLM in LLM adaptation for networking. We showcase that across the three use cases, the NetLLM-adapted LLM significantly outperforms state of the arts with performance improvements of 10.1-36.6% for VP, 14.5-36.6% for ABR, 6.8-41.3% for CJS, in terms of their respective performance metrics. Besides, our empirical test results also show that we can efficiently utilize the extensive knowledge of the LLM with our proposed NetLLM framework to achieve stronger generalization on unseen testing environments.

The contributions of this paper are summarized as follows:

- We identify the key challenges of adapting LLMs for networking and demonstrate that some natural alternatives fall short in addressing these challenges (§3).
- We then design NetLLM, the first LLM adaptation framework for networking that incorporates a multimodal encoder module to encode multimodal task inputs, networking head module to directly generate answers and a DD-LRNA scheme to reduce the adaptation costs (§4).
- We extensively evaluate NetLLM across three networking tasks. We showcase that the LLM adapted by our framework can significantly surpass state-of-the-art algorithms and achieve superior generalization performance. We also conduct a deep dive into NetLLM to provide an in-depth understanding of it (§5).

2 BACKGROUND

2.1 Learning-Based Networking Algorithms

Learning-based algorithms design and train deep neural networks (DNNs) to efficiently learn to solve networking tasks [2, 64, 103]. In particular, there are two learning paradigms commonly adopted to enable the learning process: supervised learning (SL) [95] and reinforcement learning (RL) [88]. SL is widely employed for prediction tasks in networking, such as traffic classification [54, 73], bandwidth prediction [64, 105] and viewport prediction [34, 85]. It trains DNNs with specific datasets to optimize a pre-defined loss function, so that once trained, DNNs can be used for efficient prediction to assist the system control. For example, Yan et al. [105] train a DNN model over real-world bandwidth datasets to predict the future bandwidth for bitrate control on video clients. On the flip side, RL is well-suited for sequential decision-making tasks in networking, including congestion control [1, 106], video streaming [44, 62] and cluster job scheduling [63, 78]. In these tasks, DNNs actively interact with the environments to collect experiences, then use them to optimize task-specific reward functions so that the performance of network systems can be optimized. For instance, Mao et al. [63] employ RL to train a DNN model to allocate resources to job requests so as to maximize the utilization of computing resources in a distributed computing cluster.

The major limitations of learning-based algorithms are two-fold. First, they require designing specialized DNN models for different networking tasks, thus entailing high model engineering overhead [66]. Second, they are prone to generalization issues, as DNNs may achieve poor performance on unseen data distributions/environments [44]. These problems, in fact, are not unique to networking and have been successfully addressed in other domains (e.g., NLP [10, 45] and robotics [23, 112]) by introducing LLMs as the foundation models to solve various downstream tasks. Hence, this work takes a pioneering step by systematically introducing LLMs into networking to address these problems.

2.2 Large Language Models

The advent of large language models (LLMs) such as ChatGPT [72], PaLM [15], Llama2 [92] and OPT [108] has profoundly revolutionized the field of natural language processing (NLP). These LLMs,

[2]The codes available at: https://github.com/duowuyms/NetLLM.

Table 1: Information of three learning-based algorithm use cases in the networking area.

Task	DNN Input	DNN Output	Objective	Learning Paradigm
Viewport Prediction (VP)	*time-series:* historical viewports; *image:* video content information	future viewports	minimize error between predicted and actual viewports	SL
Adaptive Bitrate Streaming (ABR)	*time-series:* historical throughputs, delay; *sequence:* chunk sizes at different bitrates; *scalar:* current buffer length	bitrate selected for the next video chunk	maximize user's Quality of Experience (QoE)	RL
Cluster Job Scheduling (CJS)	*graph:* DAGs describing dependency and resource demands of job execution stages	job stage to run next, number of executors allocated to the stage	minimize job completion time	RL

pre-trained over large public corpora (e.g., wikis and books) to acquire extensive knowledge, have demonstrated remarkable capability across a wide range of applications that largely affect our daily life, including dialogue systems [31], step-by-step reasoning [45], and even code generation [13].

LLMs are essentially large DNNs built on top of Transformer [94], the de facto standard architecture for sequence modeling in NLP tasks. They model the inputs and outputs as sequences of tokens representing sub-words in NLP (e.g., a word "awesome" can be split into two tokens: "aw" and "esome"). Specifically, they take as input a sequence of tokens, and generate another sequence of tokens as answer with the assistance of three key components: *tokenizer*, *vocabulary* and *language modeling (LM) head*. Figure 1 illustrates the answer generation mechanism of LLM. Given an input sentence, the tokenizer splits it into a list of tokens. Then the vocabulary is used to map each token into an embedding vector that can be understood and effectively processed by LLM. Afterwards, LLM encodes these token embeddings into high-level features, which are subsequently passed to the LM head to predict the probability distribution of next token. Note that the output tokens are generated *one by one* in the autoregressive manner [46, 61]. Both the sequence of input tokens and previously generated tokens are repeatedly fed into the LLM to predict the next token, until an end-of-sentence token (*<EOS>*) is emitted.

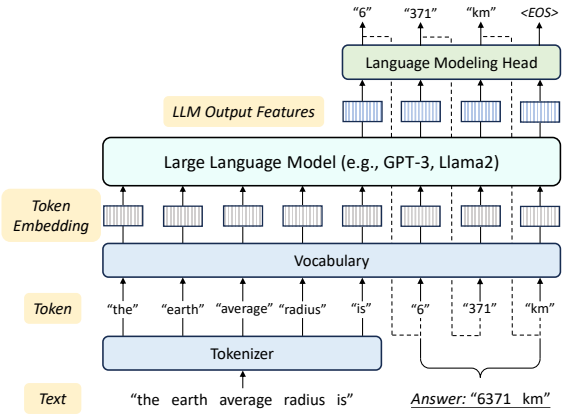


Figure 1: Illustration of the token-based answer generation mechanism of LLMs.

2.3 Domain-Adapted LLMs

The impressive performance of LLMs has sparked pioneering research to adapt LLMs for other domains [23, 55, 59, 112]. For example, PaLM-E [23] adapts PaLM [15] to generate step-by-step controlling commands for robotic manipulation. ESMFFold [55] showcases the successful applications of LLM in biological fields, which leverages LLM to predict atomic-level protein structure.

Inspired by these promising outcomes, this work explores the adaptation of LLMs for networking to address the limitations of existing learning-based algorithms, hoping to pave the way for more sustainable design philosophy of networking solutions. Unfortunately, although some prior studies have showcased that LLMs are capable of generating some technical documents for networking (e.g., description documents of digital twin for data centers [51]), none of the existing works provide an in-depth investigation on *whether and how* LLMs can be adapted to solve networking tasks. Hence, this work tries to bridge this research gap and proposes an LLM adaptation framework for networking.

3 MOTIVATION

In this section, we identify the key challenges of LLM adaptation for networking, which motivate the design of `NetLLM`. We use the following three tasks (summarized in Table 1) to make our discussion concrete:

- **Viewport prediction (VP)** serves as a fundamental building block of the emerging streaming systems of immersive videos (e.g., 360° videos [33] and volumetric videos [37]), where only the video content within the viewer's viewport (the region visible to viewer) is streamed in high quality to reduce the bandwidth consumption of video transmission [33, 57]. To accomplish this, the VP model predicts viewer's future viewport positions based on historical viewports, and potentially incorporates video content information (e.g., video frame) to enhance prediction performance [85, 98]. The goal of VP is to minimize the error between the predicted and viewer's actual viewports.
- **Adaptive bitrate streaming (ABR)** utilizes a RL model to dynamically adjust chunk-level bitrates based on the perceived network conditions and playback buffer length during the streaming session of a video [44, 103]. The objective of ABR is to maximize user's Quality of Experience (QoE), which is quantified by factors such as chunk bitrate, bitrate fluctuation, and rebuffering time.
- **Cluster job scheduling (CJS)** trains a RL scheduler to schedule incoming jobs within the distributed computing cluster [63, 78]. Each job is represented as a directed acyclic graph (DAG), which describes the dependency of each execution stage of the job and

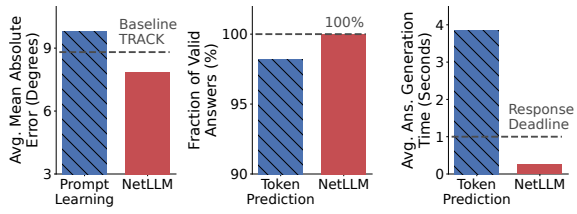


Figure 2: Illustration of the ineffectiveness for some natural alternatives with VP task as the example. *Left*: Prompt learning [60, 68] that transforms data into textual prompts achieves sub-optimal performance, while NetLLM with a multimodal encoder to encode task input data effectively outperforms baseline. *Middle, Right*: Token-based prediction with LM head fails to guarantee valid answers and produce stale responses, while NetLLM efficiently addresses these issues with the networking head module.

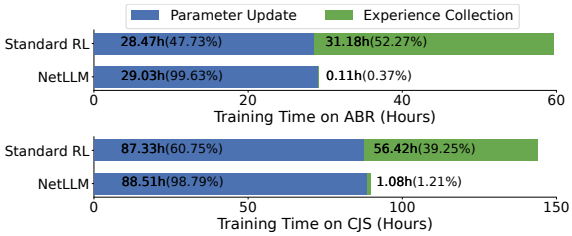


Figure 3: Using standard RL techniques [86, 93] to adapt LLM for RL-based decision-making tasks (ABR and CJS) incurs high training time due to the active environment interaction for experience collection. NetLLM eliminates this time-consuming process by designing an efficient data-driven adaptation pipeline in the DD-LRNA scheme.

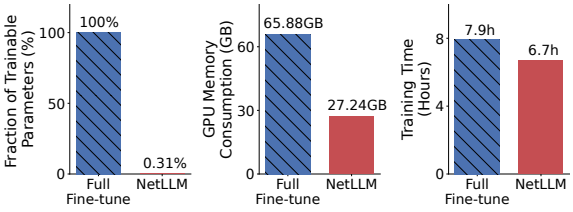


Figure 4: Illustration of the high adaptation costs of full-parameter fine-tune [16, 92] on the VP task. The DD-LRNA scheme of NetLLM efficiently reduces the costs by introducing a set of small trainable low-rank matrices.

the resource demand of each stage. The primary task of RL scheduler is to select the next stage of a job to run and allocate a number of executors (computing resources) to that particular stage. The objective is to minimize the average job completion time, so that the system-level utilization of computing resources is optimized within the cluster.

Why these tasks? We choose these tasks for several reasons.

- First, they cover the two learning paradigms commonly adopted in networking, i.e., SL for prediction tasks (VP) and RL for decision-making tasks (ABR and CJS).
- Second, they include both centralized control (CJS) and distributed control (ABR) networking tasks. Specifically, the CJS scheduler is responsible for the entire cluster, while the ABR client independently selects bitrates without considering other clients.
- Finally, they involve diverse input modalities, covering the primary data modalities in many networking tasks. For example, many continuous signals in network adaptation problems (e.g., packet loss rate in congestion control [106]) are represented as scalar data.

In particular, we choose VP as it encompasses multiple input modalities and requires cross-modality fusion, making it more challenging for LLM adaptation than other prediction tasks that generally involve single input modality (e.g., bandwidth prediction [64]). *The characteristics of these tasks ensure that our subsequent discussion is representative and applicable to a wide range of networking scenarios.*

Challenge 1: Large modality gap. As shown in Table 1, different networking tasks have different input information of diverse modalities, spanning from time-series network throughputs to DAG data. However, most LLMs are designed to accept plain text as inputs. Due to the substantial modality gap, it is impractical to directly feed the input information of networking tasks into LLMs for effective processing.

One seemingly natural approach to tackle this challenge is *prompt learning* [60, 68, 81], which transforms data into texts through a prompt template. Specifically, it designs a textual template that provides the information of task specifications, and uses this template to transform task inputs into textual prompts to instruct the LLM to generate desired answers that solve the tasks. While this approach shows promise in other fields, it falls short in the networking domain due to the following reasons. First, it is not feasible to transform data of complex modalities (e.g., image in VP and DAG in CJS) into textual prompts. Second, even if certain data can be converted into texts (e.g., time-series viewports in VP), we empirically observe that such transformation can be sub-optimal. To give a concrete example, we use this approach to adapt Llama2-7B [92] for the VP task. We design a template to encapsulate viewport data into prompts (we exclude video content information since it cannot be incorporated into prompts directly). Based on this prompt template, we instruct Llama2 to predict the future viewports in the next 1 second based on the historical viewports in the last 1 second (detailed setup of this measurement can be found in §A.1).

Figure 2 (*left*) reports the performance of prompt-learning-adapted Llama2 in terms of mean absolute error (MAE). Lower MAE means better prediction performance. As shown, under the prompt learning framework, Llama2 achieves poor performance with 11.1% higher MAE than the state-of-the-art VP model TRACK [85]. This could be attributed to the significant modality gap between text and time-series data, as textual representation may not effectively express the characteristics of time-series data. For instance, the time-varying patterns are typically not found in natural text.

Challenge 2: Inefficiency of token-based answer generation. As introduced in §2.2, by default, LLMs generate answers with LM head by predicting next tokens in an autoregressive manner. This, however, exhibits two main drawbacks in the networking area.

First, the uncertainty of token prediction increases the risk of LLM-generated answers to be physically invalid, a phenomenon known as *hallucination* [41, 49]. To quantify this issue, we calculate the fraction of valid answers (see §A.1) when adapting Llama2 for VP task based on token prediction. Figure 2 (*middle*) shows that Llama2 fails to guarantee the generated answers to be 100% valid when using token prediction. This raises concerns regarding the reliability of deploying LLMs for real-world network systems.

Second, due to the sub-word nature of tokens, a single word may span multiple tokens. In consequence, LLMs often require multiple inferences to generate a single answer, as depicted in Figure 1. This can produce delayed or even stale answers that fail to quickly adapt to the system changes. For instance, we measure the average time for Llama2 to generate a single answer for the VP task. Figure 2 (*right*) shows that it takes up to 3.84s for per-answer generation, which significantly exceeds the 1-second response deadline required for predicting future viewports in the next second.

Note that the above problems are not unique to the networking area. Nevertheless, they reduce the efficiency of LLMs in networking, since networking tasks often require high reliability and quick responsiveness [11, 66].

Challenge 3: High adaptation costs. Many networking tasks such as ABR [103] and CJS [78] employ RL to solve complex system optimization problems, which involve active interaction between the environments (e.g., network environments with varying bandwidth or workload patterns) to collect experiences for reward optimization. In this context, simply fine-tuning LLMs for these tasks based on standard RL techniques (e.g., PPO [86] and DQN [93]) will introduce prohibitive adaptation costs due to the time-consuming process of environment interaction. To be more specific, we measure the amount of time of fine-tuning Llama2 over ABR (CJS) task for 10000 (100) iterations with standard RL. Each iteration involves interacting with the environment for one episode[3] to collect experiences, followed by optimizing rewards based on these experiences. As depicted in Figure 3, the experience collection caused by active environment interaction introduces *additional* 31.18h (56.42h), accounting for 52.27% (39.25%) of the total training time on ABR (CJS) task. While this problem has been observed in prior works [62, 63], it becomes intractable in the context of adapting LLMs for networking given their large parameter sizes.

The adaptation costs become even more expensive when fine-tuning the full parameters of LLMs [21, 29]. As shown in Figure 4, fully fine-tuning Llama2-7B for the VP task consumes 65.88GB GPU memory and 7.9h of training time. This is because full-parameter fine-tune requires extensive memory and computation to store and maintain the training states due to the large parameter size. In fact, another practical drawback associated with full-parameter fine-tune is that it may disrupt the pre-trained knowledge of LLM since it updates all the parameters of LLM. As a result, this may prevent a single LLM to share across different networking tasks.

Summary. In a nutshell, we observe three challenges of LLM adaptation for networking.

- The large modality gap makes the input information of networking tasks incompatible with the LLM, preventing the LLM from effectively processing task inputs.
- The default token-based answer generation of the LLM exhibits inefficiency in the networking domain, which reduces the reliability and responsiveness for the LLM to serve network systems.
- The large parameter size of the LLM leads to significant costs to acquire domain-specific knowledge for networking, especially for RL-based tasks which require environment interaction.

[3]An episode in RL refers to a single round of a RL model to interact with the environment from the initial state to final state. For example, in ABR task, the RL model starts streaming the first video chunk (initial state) and stops until all chunks are downloaded (final state).

4 NETLLM DESIGN

In this section, we elaborate the detailed design of `NetLLM`, an innovative LLM adaptation framework for networking that efficiently solves the aforementioned challenges. As shown in Figure 5, `NetLLM` comprises three main building blocks:

- **Multimodal encoder.** `NetLLM` solves *challenge 1* by designing an encoder module to encode multimodal input data of networking tasks into token-like embeddings, which can be effectively processed by the LLM.
- **Networking head.** To address *challenge 2*, `NetLLM` replaces the LM head for token prediction with different networking heads, which enable direct generation of a valid answer for specific tasks in a single inference.
- **Data-driven low-rank networking adaptation (DD-LRNA).** To reduce the costs of adaptation, `NetLLM` develops an efficient DD-LRNA scheme to solve *challenge 3*. It incorporates a data-driven pipeline to adapt LLMs for both prediction and decision-making tasks, and introduces different low-rank matrices to learn domain knowledge to further minimize the adaptation costs.

Note that during fine-tune, the parameters of the LLM are frozen to preserve pre-trained knowledge, while the multimodal encoder, networking heads, and low-rank matrices are tunable to optimize performance for different tasks. The details of each blocks are described as follows.

4.1 Multimodal Encoder

The key to processing task-specific information is to project the multimodal input data into token space to enable efficient utilization by the LLM. To achieve this, we design a multimodal encoder to automatically learn such projection. Figure 6 illustrates the architecture of this module, which incorporates two blocks.

Feature encoder. We first employ different feature encoders to extract features from raw input data of various modalities. A key design consideration here is the choice of feature encoder for each modality. Instead of handcrafting feature encoders from scratch, which entails high model engineering costs, we reuse the existing well-designed encoders tailored for specific modalities. For example, Decima [63] for CJS task develops a graph neural network (GNN) [101] encoder to extract features from DAG information, and Vision Transformer (ViT) [22] has been widely used to encode images into hidden features. These designs are precious research outcomes and prove effective in processing specific modalities. Therefore, we cherish and efficiently utilize these designs by integrating them into our multimodal encoder module. Specifically, we leverage the following feature encoders to encode different data modalities involved in networking: ViT for images, 1D-CNN for time-series and sequence data (e.g., historical throughputs and future chunk sizes at different bitrates in ABR), fully connected layer for scalar data (e.g., buffer occupancy in ABR), and GNN for graph information (e.g., DAGs in CJS).

Linear projection. The features extracted by encoders, however, may not align to the token space. For instance, features extracted by ViT have a dimension of 768 [22], while Llama2 requires an input dimension of 4096 [92]. To address this issue, we design a set of trainable linear layers to project the features extracted by different encoders. These layers automatically learn a highly

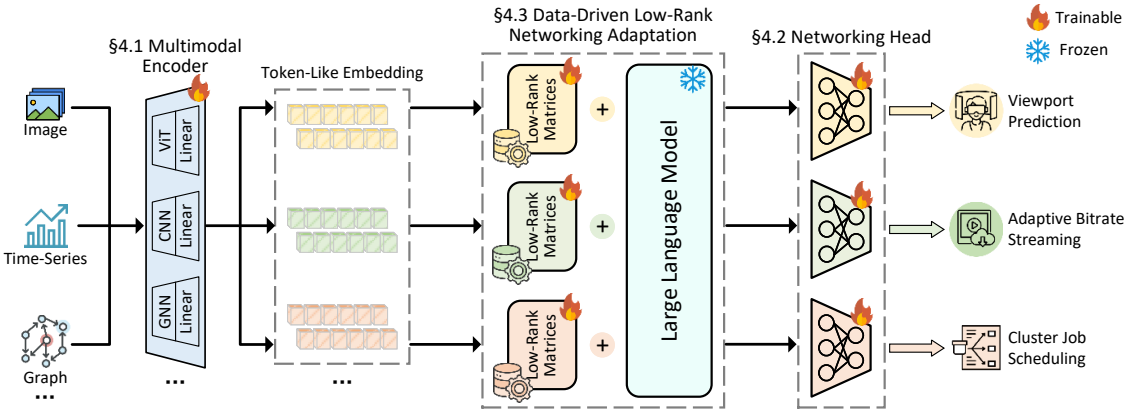


Figure 5: NetLLM consists of three core components: *multimodal encoder* to encode task inputs, *networking head* to generate task-specific answers and *data-driven low-rank networking adaptation* to efficiently learn domain knowledge for networking. The framework is illustrated with three tasks: VP, ABR and CJS, but all ideas can be easily applied to other networking tasks.

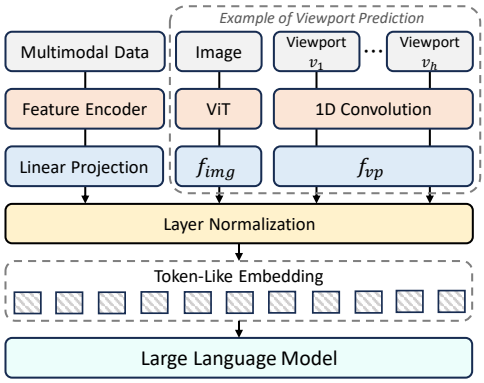


Figure 6: Illustration of the multimodal encoder of NetLLM to encode multimodal data.

efficient mapping from feature space to token space, producing a set of token-like embeddings that can be effectively utilized by the LLM. Additionally, we further enhance the projection process by normalizing the output embeddings with layer normalization [7] to ensure training stability.

Example. Figure 6 illustrates the multimodal encoder with VP task as a concrete example. The ViT and 1D convolution layer (1D-CNN) are first used to encode the image and time-series viewport data, respectively. Next, the extracted features are projected into token-like embeddings with separate linear projection layers. Finally, all embeddings are normalized through layer normalization to ensure training stability, and passed to the LLM for further processing. Figure 2 (*left*) also provides statistical results to confirm the effectiveness of multimodal encoder to project task-specific input data. As shown, empowered by this module, NetLLM significantly outperforms prompt-learning based data processing scheme [60], with the average reduction of 19.7% MAE for the VP task.

4.2 Networking Head

With the multimodal encoder, the LLM is capable to extract high-level features that encompass important task-specific information from input data of various modalities. These features are then fed into the networking head for direct answer generation. Specifically, the networking head is designed as a trainable linear layer to predict task-specific answers based on LLM output features, which can be flexibly customized according to specific networking tasks. Unlike LM head, the networking head constrains the answer generation into the valid range of possible answers, such as the range of valid viewport coordinates in VP and the set of video bitrates in ABR. In this way, all answers generated by LLM are guaranteed to be valid, ensuring the reliability of LLM for networking. Moreover, LLM can generate one answer within a single round of inference, thus significantly reducing the generation latency.

Example. Figure 7 compares the difference between LM head for token prediction and networking head with ABR task as the example. As depicted, the LM head generates answers by predicting next tokens autoregressively, which requires multiple rounds of inference and thus entails high generation latency. Besides, due to the inherent uncertainty of token-based prediction, the generated answers may be invalid, such as a bitrate that does not exist. In contrast, the networking head is specially designed to predict the probability distribution of bitrates, enabling direct answer generation within a single round of inference (e.g., the bitrate with the maximum probability is chosen as the answer in Figure 7). Furthermore, since the outputs of networking head are limited to the discrete set of candidate bitrates, the generated answers are guaranteed to be always valid. The superiority of networking head over LM head is also illustrated in Figure 2 (*middle, right*), where NetLLM uses it to ensure the validness of answers and quickly produces answers before the response deadline for VP task.

4.3 Data-Driven Low-Rank Networking Adaptation

In this part, we delve into the detailed design of the proposed data-driven low-rank networking adaptation (DD-LRNA) scheme to efficiently fine-tune LLMs to acquire domain knowledge. The DD-LRNA comprises the following two core designs: i) a data-driven adaptation pipeline for both prediction and decision-making networking tasks; ii) a low-rank adaptation approach to constrain the fine-tuning process to a small number of parameters for more efficient adaptation.

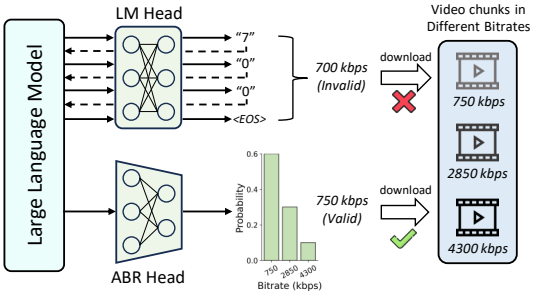


Figure 7: Comparison between LM head and networking head with ABR task as an example. For illustration, we assume that video chunks are encoded into three bitrate versions $\{750, 2850, 4300\}$ kbps.

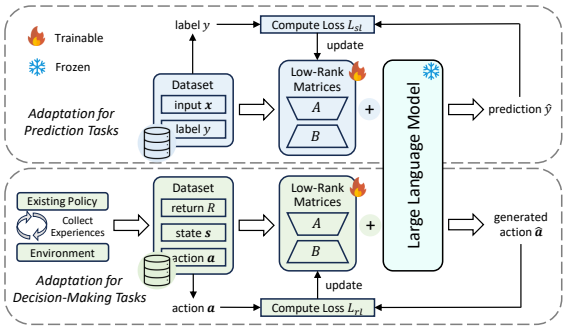


Figure 8: Illustration of the data-driven low-rank networking adaptation scheme of `NetLLM`.

Data-driven networking adaptation. In the case of prediction networking tasks, it is straightforward to fine-tune the LLM through the standard SL data-driven training pipeline. Specifically, given a task-specific dataset $\mathcal{D}_{sl} = \{\mathcal{X}, \mathcal{Y}\}$ of inputs $\boldsymbol{x} \in \mathcal{X}$ and labels $y \in \mathcal{Y}$, we leverage the multimodal encoder to encode input data $\boldsymbol{x}$, elicit prediction results $\hat{y}$ from the LLM with networking head, and compute loss for parameter update by:

$$L_{sl} = F_{sl}(y, \hat{y}) \tag{1}$$

where F_{sl} is the loss function which can be cross entropy (CE) for classification tasks (e.g., traffic classification [73]) or mean square error (MSE) for regression tasks (e.g., bandwidth prediction [64] and VP [85]).

Nevertheless, when it comes to the decision-making tasks, the traditional RL training pipeline becomes impractical due to the time-consuming process of the interaction between the LLM and environments. To tackle this challenge, we design our RL adaptation pipeline based on the efficient data-driven RL techniques [79, 106], which tackle the same problem as traditional RL but without the need of environment interaction. Specifically, we collect the experience dataset with *any* existing (non-LLM) networking algorithms, and exploit this dataset to fine-tune the LLM for reward optimization. The intuition behind is to let the LLM learn a better policy by observing the behaviors of existing algorithms [47]. That means, the LLM should not only learn from the good actions that lead to good performance, but also try to understand why some actions will lead to bad performance [106].

It is worth noting that, unlike traditional RL that requires periodic refreshing of the experience dataset during training [86, 93], our approach allows the dataset to be collected *only once* and used throughout the entire training process. As a result, the costs of adapting LLMs for RL-based networking tasks can be significantly reduced (e.g., with 51.1%/37.7% reduction of training time for ABR/CJS task under the same training iterations, as depicted in Figure 3).

The proposed RL adaptation pipeline is described as follows, which is built on top of the Transformer based data-driven RL [12, 40] that caters to the sequence modeling nature of Transformer. Given a RL-based networking task, we first employ an existing policy (e.g, Decima [63] for CJS) to collect an experience dataset which consists of experience trajectories: $\mathcal{D}_{rl} = \{\tau_1, \cdots, \tau_{|\mathcal{D}_{rl}|}\}$. Each trajectory $\tau = \{r_t, \boldsymbol{s}_t, \boldsymbol{a}_t\}_{t=1}^{T}$ is composed of rewards r, states $\boldsymbol{s}$ and actions $\boldsymbol{a}$, where T denotes the episode length. For each sample in a trajectory $\{r_t, \boldsymbol{s}_t, \boldsymbol{a}_t\} \in \tau$, we substitute the reward r_t by return $R_t = \sum_{i=t}^{T} r_i$ representing the cumulative rewards expected to receive from state $\boldsymbol{s_t}$. Additionally, considering that the state or action in some tasks may be constituted by multiple pieces of information (e.g., the state in ABR includes past network throughput and playback buffer length), we further discretize each state and action: $\boldsymbol{s}_t = \{s_t^1, \cdots, s_t^n\}, \boldsymbol{a}_t = \{a_t^1, \cdots, a_t^m\}$. This leads to the following representation of trajectory:

$$\tau = \{R_t, s_t^1, \cdots, s_t^n, a_t^1, \cdots, a_t^m\}_{t=1}^{T} \tag{2}$$

Based on the above trajectory representation, we then fine-tune the LLM to learn the distribution of returns. At each training step, we randomly sample a sequence of data from the dataset:

$$d = \{R_i, s_i^1, \cdots, s_i^n, a_i^1, \cdots, a_i^m\}_{i=t-w+1}^{t} \in \mathcal{D}_{rl} \tag{3}$$

where w is the context window to facilitate the learning of return distribution. Next, we feed data d to the LLM to generate actions $\{\hat{a}_i^1, \cdots, \hat{a}_i^m\}_{i=t-w+1}^{t}$. Notably, we consider return and each piece of state, action information as different modalities, and process them separately. Finally, the training loss is calculated by:

$$L_{rl} = \frac{1}{w} \sum_{i=1}^{w} \sum_{j=1}^{m} F_{rl}(a_i^j, \hat{a}_i^j) \tag{4}$$

where F_{rl} measures the difference between action a_i^j and the generated action $\hat{a}_i^j$, which can be CE for discrete actions or MSE for continuous actions.

The underlying rationale of the above training procedure is to train the LLM to model the distribution of returns conditioned on specific states, so that once trained, it can be used to generate a series of actions that achieve the desired returns [12]. In particular, during the inference stage, we specify a target return based on the desired performance (e.g., maximum possible return to achieve excellent performance) to trigger the LLM to generate answers.

Low-rank networking adaptation. With the data-driven adaptation pipeline in place, the LLM can now be fine-tuned for networking adaptation. Given the pre-trained parameters of the LLM denoted as Φ_0, the goal of fine-tuning is to search for the parameter update $\Delta\Phi$ such that the resulting parameters $\Phi = \Phi_0 + \Delta\Phi$ are optimized for the specific networking task. Nevertheless, due to the large parameter size of the LLM, directly fine-tuning the full parameters entails prohibitive computation costs, as the dimension of learned parameters $|\Delta\Phi|$ is equal to $|\Phi_0|$ (e.g., $|\Phi_0| = 540$ billion for PaLM [15]).

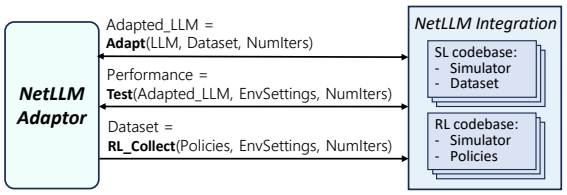


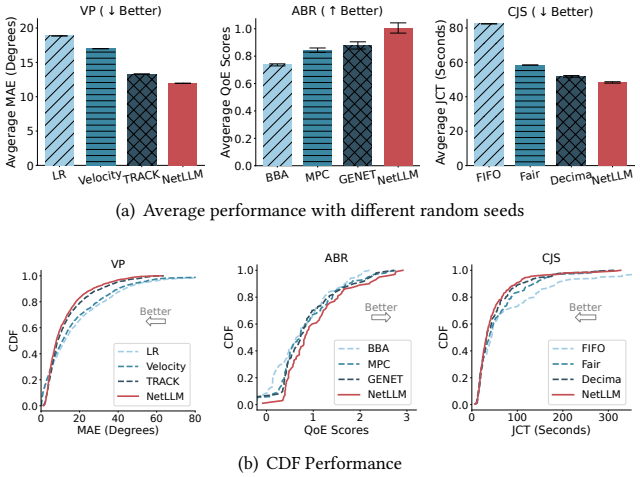
Figure 9: Components and interfaces needed to integrate NetLLM with an existing SL/RL codebase for LLM adaptation.

To combat the above limitation, we freeze the parameters of LLM and introduce additional low-rank matrices to approximate the changes needed in the LLM parameters to learn domain-specific knowledge. The underlying insight is that the parameter changes during adaptation (i.e., $\Delta\Phi$) reside on an intrinsic low rank [4, 38]. Therefore, for each pre-trained matrix $W_0 \in \Phi_0$ of dimension $d \times k$, we hypothesize the existence of a low rank $r \ll \min\{d, k\}$ and construct two low-rank matrices A, B of dimension $d \times r, r \times k$ to approximate the update of W_0, i.e., $W = W_0 + \Delta W = W_0 + AB$. During adaptation, W_0 is frozen and all parameter update is constrained on matrices A and B. As shown in Figure 4, this significantly reduces the fine-tuning costs with the reduction of 60.9% GPU memory and 15.1% training time, since the low-ranks only introduces 0.31% of trainable parameters. Another benefit of this approach is that the pre-trained knowledge of the LLM is preserved as each $W_0 \in \Phi_0$ is retained without any update. Hence, the same LLM can serve as the foundation model for different tasks, and train different copies of A, B to acquire different domain knowledge.

Putting all together. The DD-LRNA scheme is briefly summarized in Figure 8. As shown, we freeze the parameters of LLM and allocate different trainable low-rank matrices for each task. These matrices are then fine-tuned over a dataset to acquire domain-specific knowledge. For decision-making tasks, the dataset is collected by using existing algorithms to interact with environments. At each fine-tuning step, we sample a batch of data from the dataset, feed data to the LLM to generate answers, compute loss according to equation (1) for prediction tasks or equation (4) for decision-making tasks, and update the low-rank matrices through gradient descent. Note that in addition to low-rank matrices, the gradients are also propagated to update the parameters of multimodal encoder and networking head for performance optimization.

4.4 Implementation

NetLLM is fully implemented in Python and Bash, and can be easily integrated into existing SL/RL codebases to adapt LLMs for networking tasks. As depicted in Figure 9, it interacts with an existing codebase with three APIs. First, *Adapt* triggers NetLLM to use the provided dataset to adapt the LLM to learn domain-specific knowledge for the target task, and returns the snapshot of the adapted LLM. Second, *Test* evaluates the performance of the adapted LLM on the testing environments generated with the given simulation settings. Finally, for RL-based tasks without an available dataset for adaptation, NetLLM offers the *RL_Collect* API to collect the experience dataset by using the given RL policies to interact with the environments. Afterwards, the collected dataset can be plugged into the *Adapt* API to adapt the LLM.



(a) Average performance with different random seeds

(b) CDF Performance

Figure 10: Comparing NetLLM-adapted Llama2 for VP, ABR, and CJS, with baselines in testing environments generated with the same settings as training environments.

We have integrated NetLLM into three existing codebases for VP [99], ABR [103], and CJS [30], and implemented the above APIs based on the functionalities provided in the codebases. More details of implementation are provided in §A.2.

5 EVALUATION

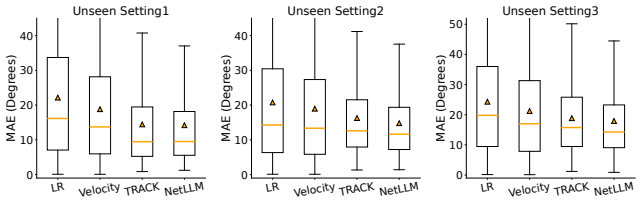
5.1 Setup

Simulation setup. By default, we utilize Llama2-7B [92] as the foundation LLM. We then use NetLLM to adapt Llama2 for three networking tasks VP [99], ABR [103], and CJS [63]. We generate different simulation environments with real-world and synthetic datasets for training and testing, following the settings described in §A.4 and Table 2, 3, 4. These settings cover the key factors that affect the model performance. For instance, in ABR task, our environment settings consider the range and changing frequency of bandwidth as well as video bitrates.

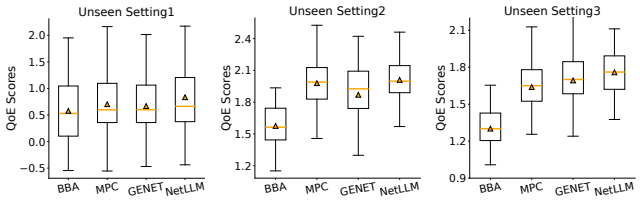
Baselines. We implement three state-of-the-art learning-based algorithms for comparison: TRACK [85] for VP, GENET [103] for ABR and Decima [63] for CJS. We choose these baselines because they have open source implementation. In addition, we also compare with other rule-based (non-DNN) algorithms for each task: linear regression (labeled "LR") [80] and velocity-based prediction (labeled "Velocity") [24] for VP, BBA [39] and MPC [107] for ABR, first-in-first-out (labeled "FIFO") and fair scheduling (labeled "Fair") [87] for CJS. Appendix §A.3 provides the brief overview of all baselines.

Metrics. For performance metrics, we consider mean absolute error (MAE) for VP, Quality of Experience (QoE) scores for ABR, job completion time (JCT) for CJS. Lower MAE, higher QoE and lower JCT indicate better performance. In particular, following the same formula in [62, 103], QoE is calculated as the weighted linear combination of bitrate, rebuffering time and bitrate changes. Appendix §A.6 provides the details of three metrics.

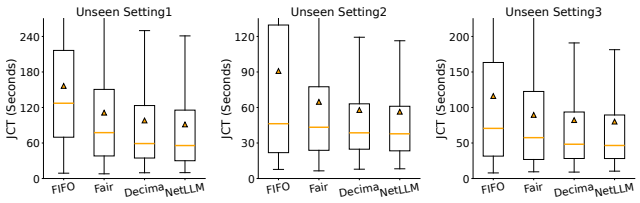
Hardware settings. We conduct experiments on a Linux server equipped with eight Intel(R) Xeon(R) Gold 5318Y CPUs and two NVIDIA 40GB A100 GPUs.



(a) VP (↓ Better)



(b) ABR (↑ Better)



(c) CJS (↓ Better)

Figure 11: Comparing the generalization performance of NetLLM-adapted Llama2 for VP, ABR, and CJS, with baselines in testing environments generated with settings different from training environments. The shape of box shows the distribution and the triangle in each box denotes average.

5.2 General Evaluation

In this part, we first compare NetLLM-adapted Llama2 with other methods across three different tasks over the testing environments with the same settings as training environments (see §A.4). In other words, for each task, we adapt Llama2 and train learning-based algorithms over the environment generated with the target setting, and test all methods in the new environment from the same setting.

Figure 10 presents the performance of each method for the corresponding tasks. As shown in Figure 10, NetLLM-adapted Llama2 consistently outperforms other methods across all cases. It surpasses all baselines by reducing 10.1-36.6% of MAE for VP, improving 14.5-36.6% of QoE for ABR and reducing 6.8-41.3% of JCT for CJS. Figure 10 also provides more detailed results in the form of CDF for each task. It can be seen that a large proportion of NetLLM-adapted Llama2 is concentrated in the range of lower MAE, higher QoE and lower JCT. For instance, for CJS task, the 90th percentile JCT of Llama2 is 97.3 seconds, while this value dramatically increases to 109.3 seconds for Decima, 135.6 seconds for Fair and 187.5 seconds for FIFO. The above outcomes highlight the effectiveness of NetLLM in LLM adaption for networking.

It can be seen that there is a clear ranking among the three baselines on each task, where the learning-based algorithms consistently yield improvement over traditional rule-based algorithms. This can be attributed to the inherent strength of DNNs in fitting complex functions for prediction. However, the LLM demonstrates even more powerful capabilities in function approximation, pattern mining and long-term planning, owing to its large parameter size and large-scale pre-training. Hence, by effectively utilizing the strengths of LLM to solve networking tasks, NetLLM achieves superior performance compared to the other learning-based algorithms. In addition, it is worth mentioning that the performance gain of learning-based algorithms relies on engineering specialized DNN models for the target tasks. In contrast, NetLLM efficiently utilizes the LLM as the foundation model for task solving. That said, it uses the same LLM to solve various networking tasks without any further modification on the model, thus significantly reducing the overhead of model engineering.

5.3 Generalization

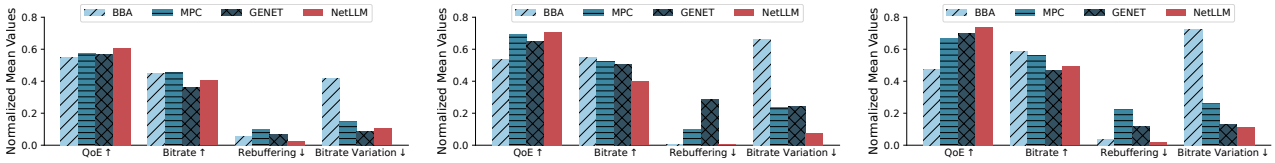
Next, we evaluate the generalization performance of all methods for each task in testing environments generated with various settings different from the training environments (see §A.4). As depicted in Figure 11, NetLLM-adapted Llama2 consistently outperforms baselines in terms of average values and distributions across all cases. For instance, compared to the learning-based algorithms, it reduces the MAE by 1.7-9.1%, improves the QoE by 3.9-24.8% and reduces the JCT by 2.5-6.8% on average. This suggests that, enabled by NetLLM, Llama2 demonstrates superior generalization performance.

From Figure 11, we also notice that learning-based algorithms do not always outperform conventional rule-based algorithms for the ABR task. Figure 12 breaks down the QoE scores of all ABR methods for more detailed analysis. As shown, GENET is surpassed by MPC with 5.2%/5.9% lower average QoE on unseen setting 1/2. More specifically, on unseen setting 1, where the streaming video is different from training one, GENET fails to optimize video bitrates and thus achieves worse performance than MPC. On the other hand, GENET struggles to adapt to the dynamic fluctuations of bandwidth on unseen setting 2, where the testing bandwidth traces change more frequently than training ones. It may inappropriately select high bitrates when the current bandwidth resources become scarce, and thus produces the highest rebuffering time among other methods. In contrast, NetLLM-adapted Llama2 strikes a good balance between the three QoE factors and thus achieves the highest QoE scores on all settings. These cases exemplify that conventional DNN models may perform poorly in unseen environments. In comparison, leveraging our NetLLM framework, we can indeed efficiently utilize the extensive knowledge of the LLM to achieve stronger generalization.

Real-world tests. As a final test of generalization, we evaluate NetLLM-adapted Llama2 in a real-world client-server ABR system under different network connections (see §A.5 for detailed setup). The results are reported in Figure 14. On each network, the adapted Llama2 outperforms the baselines. This indicates that the LLM adapted by NetLLM is able to generalize to real-world scenarios.

5.4 Deep Dive

Importance of pre-trained and domain knowledge. To gain a deeper understanding of why LLMs can be adapted for networking, we investigate the importance of both the pre-trained and learned domain knowledge of LLMs in networking adaptation. We use



(a) ABR Unseen Setting 1 (b) ABR Unseen Setting 2 (c) ABR Unseen Setting 3

Figure 12: Comparing NetLLM-adapted Llama2 with baselines for ABR by breaking down their performance on individual QoE factors in different unseen environments. Results are normalized through min-max. Arrow ↑ / ↓ means higher/lower is better.

Figure 13: Exploring the importance of pre-trained and learned domain-specific knowledge of LLM in networking adaptation.

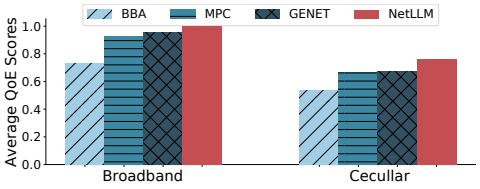


Figure 14: Comparing NetLLM-adapted Llama2 with baselines for ABR on real-world environments with different network connections.

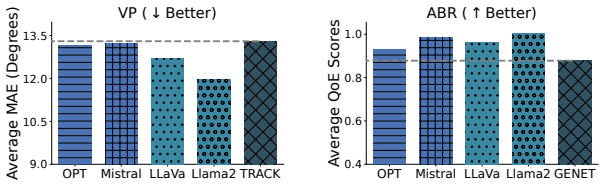


Figure 15: Comparing the performance of different LLMs adapted by NetLLM for VP and ABR with learning-based algorithms.

Llama2-7B as the LLM for our exploration. First, we disable the pre-trained weights of Llama2 that represent its pre-trained knowledge, randomly initialize its weights and train it from scratch for each task. As depicted in Figure 13, the absence of pre-trained knowledge leads to dramatic performance drop across all tasks. This indicates that while LLMs are pre-trained over text corpora to acquire language knowledge, their emergent abilities (e.g., planning [102], pattern mining [112]) are indeed universal and applicable across domains, including networking. For instance, the pattern mining ability of LLM can be utilized to mine complex changing patterns of viewport movement for accurate viewport prediction. Hence, the pre-trained knowledge of LLMs is crucial for networking adaptation.

Next, we preserve the pre-trained knowledge of Llama2 but disable the low-rank matrices that represent the learned domain knowledge. As reported in Figure 13, the absence of domain knowledge also results in performance degradation on each task, which highlights the importance of NetLLM to acquire domain knowledge.

Impacts of different types of LLMs. To validate whether NetLLM is applicable to various LLMs, we employ it to adapt three additional LLMs besides Llama2 for the VP and ABR tasks: OPT [108], Mistral [42] and LLaVa [56]. The size of each LLM is set to 7B for fair comparison. It is worth mentioning that LLaVa is a multimodal LLM trained on the combination of image and text corpora. We select LLaVa for our evaluation to investigate whether the pre-trained knowledge of multimodal fusion is applicable for networking.

As shown in Figure 15, all the adapted LLMs outperform the state of the arts on both tasks, which confirms the compatibility of NetLLM. Interestingly, we observe that the multimodal LLaVa performs worse than to single-modal Llama2. This suggests that the knowledge acquired by LLaVa in multimodal fusion during pre-training may not be directly beneficial in the networking domain[4].

Impacts of different sizes of LLMs. Next, we investigate the impacts of LLM sizes on the adaptation performance. We select OPT [108] as the foundational model for this investigation, which offers different versions with varying parameter sizes. The results are presented in Figure 16. As shown, when the parameter size exceeds 1B, the adapted OPT achieves superior or comparable performance to the advanced learning-based algorithms. However, for the ABR task, OPT-0.35B performs significantly worse than all baselines, potentially due to the limited common knowledge to generalize across tasks. This suggests that, in practice, LLMs with parameter sizes greater than 1B are suitable for networking adaptation, while those smaller than 1B may not be the optimal choices for adaptation.

Computation overhead. To measure the overhead of deploying NetLLM-adapted LLM to solve networking tasks, we profile the LLM answer generation process. Overall, loading a 7B LLM like Llama2-7B requires 29 GB memory and takes about 0.1s~0.3s to generate one answer. The computation overhead can be reduced by utilizing the advanced model compression techniques [19, 104] (discussed in §6), or employing smaller LLMs such as OPT-1.3B which also achieves superior performance over baselines (see Figure 16). Specifically, for OPT-1.3B, it only takes 7GB to load the model, which can be accommodated by commercial GPUs like NVIDIA 10GB 3080. Besides, it takes about 0.04s for OPT-1.3B to generate one answer, which is acceptable for many networking tasks.

6 DISCUSSION

Q1: What considerations are needed when adapting LLMs for specific networking tasks using NetLLM?

While NetLLM's overall design is independent of specific networking tasks, some considerations are needed when applying it for LLM adaptation. Specifically, when adapting LLMs for the target

[4]We leave deeper investigation of the efficacy of multimodal LLMs in networking for future work.

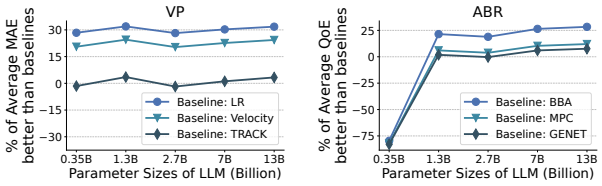


Figure 16: Exploring the impacts of LLM sizes in networking adaptation, with OPT [108] as the foundation model.

networking task, the creation of a new networking head is necessary, and the selection of a modality-specific feature encoder is also needed when dealing with a new modality. However, NetLLM minimizes the ad-hoc design costs associated with these considerations. On one hand, the networking head in NetLLM is essentially a simple linear layer that can be easily customized based on the task output space. For example, in VP task, the networking head is simply designed with three neurons to output the viewport coordinates, i.e., roll, pitch and yaw (see §A.2). On the other hand, NetLLM facilitates efficiency by allowing the reuse of existing encoders to process specific modalities, eliminating the need of handcrafting encoders from scratch. Furthermore, as the current LLM research landscape actively explores multimodality [28, 50, 71, 91], NetLLM stands to benefit from the advancements in this field. With the development of more generic and powerful LLMs that support more modalities, NetLLM can utilize their built-in encoders to effectively process multimodal data.

Q2: How does NetLLM compare to retrieval-augmented generation (RAG)?

Retrieval-augmented generation (RAG) [48, 74, 100] has been recently proposed to enhance the capabilities of LLMs. RAG constructs a large corpus as an external knowledge base to store domain-specific knowledge. During the inference stage, relevant information is retrieved from the knowledge base and attended to the input context of LLMs to provide domain knowledge. Although RAG has shown efficacy in improving the performance of LLMs in NLP tasks, it faces challenges when applied in the networking field. This is because, unlike NLP where knowledge can be stored explicitly in the textual form (e.g., the names of presidents of different countries), representing domain knowledge in networking as plain texts is challenging due to the abstract and implicit nature of networking knowledge (e.g., the ABR policy to dynamically adjust bitrates based on the changing network conditions). As a result, constructing an external knowledge base for RAG is rather challenging in the field of networking. In contrast, NetLLM takes a different approach by designing an efficient DD-LRNA module, which enables LLMs to automatically and effectively learn domain-specific knowledge for networking. With this module, NetLLM facilitates efficient acquisition of networking knowledge without relying on the construction of an external knowledge base.

Q3: How to reduce the computation overhead of LLMs?

There is a wealth of research in model compression [19, 104] which can be leveraged to reduce the computation overhead of LLMs, including model pruning [58, 109], quantization [89, 90] and knowledge distillation [69, 75]. For example, SparseGPT [26] demonstrates that LLMs can be pruned to ignore at least 50% parameters with minimal performance loss. OPTQ [27] uses quantization to decrease the bit-width of OPT-1.3B from originally 16 bits to 4 bits with negligible performance degradation while significantly reducing the model size by 4×. These active lines of research can be integrated into NetLLM to reduce the overhead of LLMs when deploying them for networking in practice. While the trade-off between performance and resource consumption should be considered when applying these techniques, we leave further exploration of this trade-off for future work.

Q4: Why can LLMs be useful in networking?

In §5.4, we have identified from the high-level perspective that the pre-trained knowledge of LLMs is one of the dominant factors to their success in networking. However, further investigations into the internal working mechanisms of LLMs are crucial to improve their explainability. Gaining deeper insights into the explainability of LLMs enables researchers to comprehend their capabilities, limitations, and areas for improvement [5, 110]. This understanding, in turn, paves the way for the development of more reliable and secure LLM-based networking systems that can be trustfully deployed in real-world scenarios. Therefore, a significant future research direction lies in designing an interpretable system to elucidate the behaviors of LLMs in the context of networking, which will greatly facilitate the effective utilization of LLMs in networking.

7 CONCLUDING REMARKS

In this paper, we for the first time explore the utilization of LLMs as foundation models for networking to reduce handcraft costs involved in algorithm design and achieve strong generalization. To achieve this, we propose NetLLM, the first framework that efficiently adapts LLMs for different networking tasks. Across three use cases in networking, we show that NetLLM enables the effective utilization of a single LLM to achieve superior performance and generalization in multiple networking tasks. While NetLLM by no means is the final answer, we hope that it serves as a stepping stone towards a more sustainable design philosophy for future networking algorithms and demonstrates the potential of adapting LLMs for networking.

Ethics: This work does not raise any ethical issues.

ACKNOWLEDGEMENTS

We thank the anonymous SIGCOMM reviewers and our shepherd, Shay Vargaftik, for their invaluable feedbacks. This work was supported in part by NSFC with Grant No. 62293482, the Basic Research Project No. HZQB-KCZYZ-2021067 of Hetao Shenzhen-HK S&T Cooperation Zone, NSFC with Grant No. 62102342, the Shenzhen Science and Technology Program with Grant No. RCBS20221008093120047, the Young Elite Scientists Sponsorship Program of CAST with Grant No. 2022QNRC001, the Shenzhen Outstanding Talents Training Fund 202002, the Guangdong Research Projects No. 2017ZT07X152 and No. 2019CX01X104, the Guangdong Provincial Key Laboratory of Future Networks of Intelligence (Grant No. 2022B1212010001), and the Shenzhen Key Laboratory of Big Data and Artificial Intelligence (Grant No. ZDSYS201707251409055). Zhi Wang's work was supported in part by Shenzhen Science and Technology Program (Grant No. JCYJ20220818101014030). Junchen Jiang's work was supported in part by NSF CNS 2146496, 1901466, 2313190, 2131826.

REFERENCES

[1] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the 2020 ACM SIGCOMM Conference*. 632–647.

[2] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. 2021. Wanna make your tcp scheme great for cellular networks? Let machines do it for you! *IEEE Journal on Selected Areas in Communications* 39, 1 (2021), 265–279.

[3] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. 2020. An optimistic perspective on offline reinforcement learning. In *Proceedings of the 2020 International Conference on Machine Learning*. PMLR, 104–114.

[4] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255* (2020).

[5] J Alammar. 2021. Ecco: An open source library for the explainability of transformer language models. In *Proceedings of the 59th Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. 249–257.

[6] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *2018 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 329–342.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450* (2016).

[8] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2015. Information-agnostic flow scheduling for commodity data centers. In *2015 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 455–468.

[9] Abdelhak Bentaleb, Christian Timmerer, Ali C. Begen, and Roger Zimmermann. 2019. bandwidth prediction in low-latency chunked streaming. In *Proceedings of the 2019 ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 7–13.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, Vol. 33. 1877–1901.

[11] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. 2018. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 ACM SIGCOMM Conference*. 191–205.

[12] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems* 34 (2021), 15084–15097.

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[14] Tatsuhiro Chiba and Tamiya Onodera. 2016. Workload characterization and optimization of TPC-H queries on Apache Spark. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 112–121.

[15] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[16] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416* (2022).

[17] Jan Clusmann, Fiona R Kolbinger, Hannah Sophie Muti, Zunamys I Carrero, Jan-Niklas Eckardt, Narmin Ghaffari Laleh, Chiara Maria Lavinia Löffler, Sophie-Caroline Schwarzkopf, Michaela Unger, Gregory P Veldhuizen, et al. 2023. The future landscape of large language models in medicine. *Communications medicine* 3, 1 (2023), 141.

[18] Federal Communications Commission. 2016. Raw data - measuring broadband america. (2016). https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016

[19] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.

[20] Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Haitao Zheng, and Maosong Sun. 2022. OpenPrompt: An open-source framework for prompt-learning. In *Proceedings of the 2022 Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 105–113.

[21] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence* 5, 3 (2023), 220–235.

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2021).

[23] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378* (2023).

[24] Xianglong Feng, Zeyang Bao, and Sheng Wei. 2021. Liveobj: Object semantics-based viewport prediction for live mobile virtual reality streaming. *IEEE Transactions on Visualization and Computer Graphics* 27, 5 (2021), 2736–2745.

[25] DASH Industry Form. 2016. Reference Client 2.4.0. (2016). http://mediapm.edgesuite.net/dash/public/nightly/samples/dash-if-reference-player/index.html.

[26] Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 2023 International Conference on Machine Learning*. PMLR, 10323–10337.

[27] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Optq: Accurate quantization for generative pre-trained transformers. In *Proceedings of the 2022 International Conference on Learning Representations*.

[28] Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, et al. 2023. Mme: A comprehensive evaluation benchmark for multimodal large language models. *arXiv preprint arXiv:2306.13394* (2023).

[29] Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. *Proceedings of the 2023 AAAI Conference on Artificial Intelligence*, 12799–12807.

[30] Archie Gertsman. 2024. spark-sched-sim: An apache spark job scheduling simulator, implemented as a Gymnasium environment. (2024). https://github.com/ArchieGertsman/spark-sched-sim

[31] Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. 2022. Improving alignment of dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375* (2022).

[32] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *Proceedings of the 2014 ACM SIGCOMM Conference*. 455–466.

[33] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: Optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of the 2019 ACM SIGCOMM Conference*. 394–407.

[34] Quentin Guimard, Lucile Sassatelli, Francesco Marchetti, Federico Becattini, Lorenzo Seidenari, and Alberto Del Bimbo. 2022. Deep variational learning for multiple trajectory prediction of 360° head movements. In *Proceedings of the*

2022 ACM Multimedia Systems Conference. 12–26.
[35] Antonio Gulli and Sujit Pal. 2017. *Deep learning with Keras*. Packt Publishing Ltd.
[36] Satyandra Guthula, Navya Battula, Roman Beltiukov, Wenbo Guo, and Arpit Gupta. 2023. netFound: Foundation Model for Network Security. *arXiv preprint arXiv:2310.17025* (2023).
[37] Bo Han, Yu Liu, and Feng Qian. 2020. Vivo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 2020 ACM Annual International Conference on Mobile Computing and Networking*. Article 11, 13 pages.
[38] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
[39] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM SIGCOMM Conference*. 187–198.
[40] Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline reinforcement learning as one big sequence modeling problem. *Advances in Neural Information Processing Systems* 34 (2021), 1273–1286.
[41] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12, Article 248 (2023), 38 pages.
[42] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825* (2023).
[43] Yili Jin, Junhua Liu, Fangxin Wang, and Shuguang Cui. 2022. Where are you looking? A large-scale dataset of head and gaze behavior for 360-degree videos and a pilot study. In *Proceedings of the 2022 ACM International Conference on Multimedia*. 1025–1034.
[44] Nuowen Kan, Yuankun Jiang, Chenglin Li, Wenrui Dai, Junni Zou, and Hongkai Xiong. 2022. Improving generalization for neural adaptive video streaming via meta reinforcement learning. In *Proceedings of the 2022 ACM International Conference on Multimedia*. 3006–3016.
[45] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems* 35 (2022), 22199–22213.
[46] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 2023 ACM Symposium on Operating Systems Principles*. 611–626.
[47] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).
[48] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.
[49] Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 6449–6464.
[50] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597* (2023).
[51] Minghao Li, Ruihang Wang, Xin Zhou, Zhaomeng Zhu, Yonggang Wen, and Rui Tan. 2023. Chattwin: Toward automated digital twin generation for data center via large language models. In *Proceedings of the 2023 ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 208–211.
[52] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large Language Models in Finance: A Survey. In *Proceedings of the ACM International Conference on AI in Finance*. Association for Computing Machinery, New York, NY, USA, 374–382.
[53] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.
[54] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the 2022 ACM Web Conference*. 633–642.
[55] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. 2023. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science* 379, 6637 (2023), 1123–1130.
[56] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *arXiv preprint arXiv:2304.08485* (2023).
[57] Junhua Liu, Boxiang Zhu, Fangxin Wang, Yili Jin, Wenyi Zhang, Zihan Xu, and Shuguang Cui. 2023. Cav3: Cache-assisted viewport adaptive volumetric video streaming. In *Proceedings of the 2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 173–183.
[58] Jing Liu, Bohan Zhuang, Zhuangwei Zhuang, Yong Guo, Junzhou Huang, Jinhui Zhu, and Mingkui Tan. 2022. Discrimination-aware network pruning for deep model compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2022), 4035–4051.
[59] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, Bonita Bhaskaran, Bryan Catanzaro, Arjun Chaudhuri, Sharon Clay, Bill Dally, Laura Dang, Parikshit Deshpande, Siddhanth Dhodhi, Sameer Halepete, Eric Hill, Jiashang Hu, Sumit Jain, Brucek Khailany, George Kokai, Kishor Kunal, Xiaowei Li, Charley Lind, Hao Liu, Stuart Oberman, Sujeet Omar, Sreedhar Pratty, Jonathan Raiman, Ambar Sarkar, Zhengjiang Shao, Hanfei Sun, Pratik P Suthar, Varun Tej, Walker Turner, Kaizhe Xu, and Haoxing Ren. 2023. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176* (2023).
[60] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.
[61] Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, Ganesh Ananthanarayanan, and Junchen Jiang. 2023. CacheGen: Fast Context Loading for Language Model Applications. *arXiv preprint arXiv:2310.07240* (2023).
[62] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the 2017 ACM SIGCOMM Conference*. 197–210.
[63] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the 2019 ACM SIGCOMM Conference*. 270–288.
[64] Lifan Mei, Runchen Hu, Houwei Cao, Yong Liu, Zifan Han, Feng Li, and Jin Li. 2020. Realtime mobile bandwidth prediction using lstm neural network and bayesian fusion. *Computer Networks* 182 (2020), 107515.
[65] Xuying Meng, Chungang Lin, Yequan Wang, and Yujun Zhang. 2023. Netgpt: Generative pretrained transformer for network traffic. *arXiv preprint arXiv:2304.09513* (2023).
[66] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting deep learning-based networking systems. In *Proceedings of the 2020 ACM SIGCOMM Conference*. 154–171.
[67] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2024. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing (Second Edition)* (second edition ed.), Robert Kozma, Cesare Alippi, Yoonsuck Choe, and Francesco Carlo Morabito (Eds.). Academic Press, 269–287.
[68] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.
[69] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2020. Improved knowledge distillation via teacher assistant. In *Proceedings of the 2020 AAAI Conference on Artificial Intelligence*. 5191–5198.
[70] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: accurate record-and-replay for http. In *2015 USENIX Annual Technical Conference (USENIX ATC)*. 417–429.
[71] OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
[72] OpenAI. 2024. Chatgpt. (2024). https://chat.openai.com/chat
[73] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. 2018. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials* 21, 2 (2018), 1988–2014.
[74] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* (2024), 1–20.
[75] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. 2019. Relational knowledge distillation. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3967–3976.
[76] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison,

Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019).

[77] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116* (2023).

[78] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2021. Dl2: A deep learning-driven scheduler for deep learning clusters. *IEEE Transactions on Parallel and Distributed Systems* 32, 8 (2021), 1947–1960.

[79] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. 2023. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems* (2023), 1–21.

[80] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 2018 ACM Annual International Conference on Mobile Computing and Networking*. 99–114.

[81] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[82] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A generalist agent. *arXiv preprint arXiv:2205.06175* (2022).

[83] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Comput. Surv.* 54, 4, Article 76 (may 2021), 34 pages.

[84] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 2013 ACM Multimedia Systems Conference*. 114–118.

[85] Miguel Fabián Romero Rondón, Lucile Sassatelli, Ramón Aparicio-Pardo, and Frédéric Precioso. 2022. Track: A new method from a re-examination of deep architectures for head motion prediction in 360° videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 9 (2022), 5681–5699.

[86] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[87] Apache Spark. 2024. Job Scheduling - Spark 3.5.0 Documentation. (2024). https://spark.apache.org/docs/latest/job-scheduling.html Accessed: 2024-01-08.

[88] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[89] Chen Tang, Kai Ouyang, Zhi Wang, Yifei Zhu, Wen Ji, Yaowei Wang, and Wenwu Zhu. 2022. Mixed-precision neural network quantization via learned layer-wise importance. In *Proceedings of the 2022 European Conference on Computer Vision*. Springer Nature Switzerland, 259–275.

[90] Chen Tang, Haoyu Zhai, Kai Ouyang, Zhi Wang, Yifei Zhu, and Wenwu Zhu. 2022. Arbitrary bit-width network: A joint layer-wise quantization and adaptive inference approach. In *Proceedings of the 2022 ACM International Conference on Multimedia*. 2899–2908.

[91] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[92] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[93] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the 2016 AAAI Conference on Artificial Intelligence*, Vol. 30.

[94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).

[95] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. 2020. Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Networks* 125 (2020), 258–280.

[96] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652* (2022).

[97] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A dataset for exploring user behaviors in VR spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. 193–198.

[98] Chenglei Wu, Ruixiao Zhang, Zhi Wang, and Lifeng Sun. 2020. A spherical convolution approach for learning long term viewport prediction in 360 immersive video. In *Proceedings of the 2020 AAAI Conference on Artificial Intelligence*. 14003–14040.

[99] Duo Wu, Panlong Wu, Miao Zhang, and Fangxin Wang. 2023. Mansy: Generalizing neural adaptive immersive video streaming with ensemble and representation learning. *arXiv preprint arXiv:2311.06812* (2023).

[100] Yuxiang Wu, Yu Zhao, Baotian Hu, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2022. An Efficient Memory-Augmented Transformer for Knowledge-Intensive NLP Tasks. In *Proceedings of the 2022 EMNLP Conference*. 5184–5196.

[101] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

[102] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

[103] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. 2022. Genet: Automatic curriculum generation for learning adaptation in networking. In *Proceedings of the 2022 ACM SIGCOMM Conference*. 397–413.

[104] Canwen Xu and Julian McAuley. 2023. A survey on model compression and acceleration for pretrained language models. In *Proceedings of the 2023 AAAI Conference on Artificial Intelligence*. 10566–10575.

[105] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: A randomized experiment in video streaming. In *2020 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 495–511.

[106] Chen-Yu Yen, Soheil Abbasloo, and H Jonathan Chao. 2023. Computers Can Learn from the Heuristic Designs and Master Internet Congestion Control. In *Proceedings of the 2023 ACM SIGCOMM Conference*. 255–274.

[107] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM SIGCOMM Conference*. 325–338.

[108] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[109] Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. 2022. Advancing model pruning via bi-level optimization. *Advances in Neural Information Processing Systems* 35 (2022), 18309–18326.

[110] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology* 15, 2 (2024), 1–38.

[111] Hao Zhou, Chengming Hu, Ye Yuan, Yufei Cui, Yili Jin, Can Chen, Haolun Wu, Dun Yuan, Li Jiang, Di Wu, et al. 2024. Large Language Model (LLM) for Telecommunications: A Comprehensive Survey on Principles, Key Techniques, and Opportunities. *arXiv preprint arXiv:2405.10825* (2024).

[112] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 2023 PMLR Conference on Robot Learning (CoRL)*. 2165–2183.

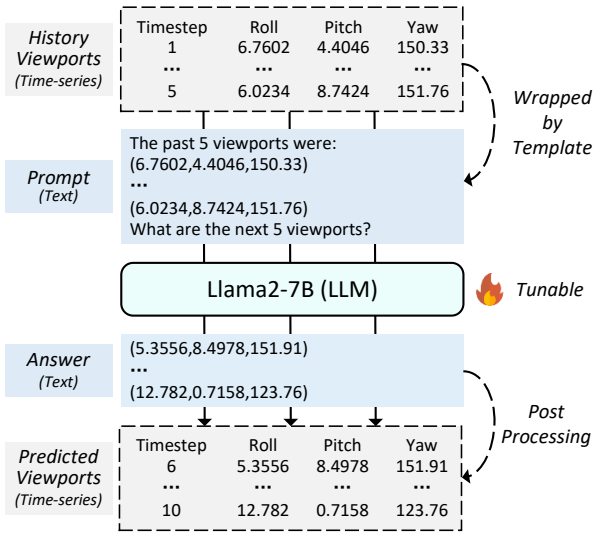


Figure 17: Illustration of using prompt learning [60] to adapt the Llama2-7B LLM [92] for the VP task.

A APPENDICES

Appendices are supporting material that has not been peer-reviewed.

A.1 Details of Figure 2

The details to produce the results of "Prompt Learning" and "Token Prediction" in Figure 2 are illustrated in Figure 17 and described as follows. We use prompt learning [60] to adapt the Llama2-7B LLM for the VP task. Specifically, we design a *prompt template* to encapsulate time-series viewports into textual prompts, and instruct Llama2 [92] to generate answers for the VP task based on *token prediction*. We conduct our measurements on an existing immersive video viewport dataset [43]. As a motivating example, we use Llama2 to predict future viewports in the next 1 second based on past viewports in the last 1 second. Following previous works [34, 85], the viewport sampling rate is set to 5Hz, thus the future viewports and historical viewports are both 5-sample long. Note that Llama2 initially achieves poor performance without any fine-tuning. Hence, we further use OpenPrompt [20], an open-source prompt learning framework, to fine-tune Llama2 for 100000 iterations over the viewport dataset.

Note that in Figure 2 (*middle*), we calculate the fraction of valid answers generated by token prediction. In our practical implementation, we consider an answer as valid if we can extract viewports from it following a series of pre-defined string parsing operations. On the other hand, an invalid answer is often the case where it contains invalid characters (e.g., unexpected punctuation) or misses some values. While designing complex rules to post-process answers can alleviate the issue of invalid answers, it introduces the overhead of answer engineering [51].

A.2 Details of `NetLLM` Implementation

We have integrated `NetLLM` into three existing codebases for VP [99], ABR [103], and CJS [30]. We have implemented the APIs in Figure 9 based on the functionalities provided in the codebases. Additional details of `NetLLM` implementation are explained as follows.

Table 2: Summary of setting information in VP simulation. *hw*/*pw* is short for historical window/prediction window.

Setting	Viewport Dataset	Prediction Setup
default train	*Jin2022*	*hw* =2s, *pw* = 4s
default test	*Jin2022*	*hw* = 2s, *pw* = 4s
unseen setting1	*Jin2022*	*hw* = 4s, *pw* = 6s
unseen setting2	*Wu2017*	*hw* = 2s, *pw* = 4s
unseen setting3	*Wu2017*	*hw* = 4s, *pw* = 6s

For the multimodal encoder, we utilize ViT [22] to encode images, and 1D-CNN [62] to encode time-series and sequence data (e.g., historical throughputs and future chunk sizes at different bitrates in ABR). We leverage fully connected layer to extract features from scalar data (e.g., buffer occupancy in ABR), and use GNN [63, 101] to process graph information (e.g., DAGs in CJS). By default, the multimodal encoders are trainable, except that the parameters of ViT are frozen. This is because ViT has open source pre-trained weights which can be used to effectively extract image features.

The networking heads can be easily customized according to the target networking tasks. Specifically, we design the VP head to predict the viewport coordinates of roll, pitch, and yaw values. The ABR head is designed to output the probability distribution of candidate bitrates. As for CJS task, we design two heads for action generation: one to determine the next job stage to run and the other to decide the number of executor resources allocated to that stage.

Regarding the DD-LRNA scheme, we configure the context window w for learning return distribution as 10 and 20 for ABR and CJS, respectively. We then set the rank r of low-rank matrices to be 32, 128 and 128, for VP, ABR and CJS, respectively. While additional tuning of w, r may be beneficial, we empirically find that `NetLLM` performs well across a wide range of hyperparameter values (generally, $w \geq 10$ and $r \geq 32$ will yield good performance). Thus, we do not employ sophisticated methods to tune these hyperparameters and keep them fixed throughout the experiments in §5. As for experience collection, we use GENET [103] and Decima [63] to collect experience datasets for the RL-based ABR and CJS tasks, respectively. While using more algorithms to interact with the environments for more epochs to expend the datasets may yield potential benefits, we leave this for future exploration.

A.3 Overview of Baselines

In our evaluation, we compare the performance of the LLM adapted by our `NetLLM` framework with three baselines for each task, including state-of-the-art learning-based algorithms and rule-based algorithms. The following provides an overview of each baseline used in our evaluation.

Baselines for VP. We implement the following three baselines for performance comparison for the VP task: TRACK [85], linear regression (labeled "LR") [80], and velocity-based prediction (labeled "Velocity") [24]. TRACK [85] is a learning-based algorithm that designs a DNN model based on Long Short Term Memory (LSTM) architecture for VP. It considers both viewer's historical viewports and video saliency map as inputs to achieve state-of-the-art performance, where saliency map is an image that describes viewer's potential attention on the video content. LR [80] assumes the movement of viewer's viewports as a linear function related

Table 3: Summary of setting information in ABR simulation.

Setting	Video Dataset	Bandwidth Traces
default train	*Envivio-Dash3*	*FCC*
default test	*Envivio-Dash3*	*FCC*
unseen setting1	*Envivio-Dash3*	*SynthTrace*
unseen setting2	*SynthVideo*	*FCC*
unseen setting3	*SynthVideo*	*SynthTrace*

Table 4: Summary of setting information in CJS simulation.

setting	Job Requests	Executor Resources(k)
default train	200	50
default test	200	50
unseen setting1	200	30
unseen setting2	450	50
unseen setting3	450	30

to time, then uses linear regression to estimate such function for predicting viewer's viewports. Velocity [24] calculates the moving speed of viewer's historical viewports and uses it to estimate the positions of viewer's future viewports.

Note that since the open-source codes of TRACK are originally written in Keras [35], we carefully convert its codes into PyTorch [76] to make TRACK compatible with the VP codebase [99]. We have ensured that our implementation preserves the same functionality of TRACK as its original implementation. Besides, as TRACK does not offer pre-trained model weights, we re-train it from scratch on our datasets with the same training hyperparameters described in TRACK's paper and codes. As for rule-based algorithms LR and Velocity, which do not provide open source implementation, we implement them ourselves by strictly following the same ideas and formulas presented in their respective papers.

Baselines for ABR. As for ABR, the following baselines are implemented for comparison: GENET [103], BBA [39] and MPC [107]. GENET [103] is a RL-based streaming algorithm improved over Pensieve [62]. It introduces a curriculum learning technique to facilitate the RL training process to improve convergence performance. BBA [39] considers buffer occupancy as a critical signal for bitrate control and designs an algorithm to maintain the playback buffer occupancy at a desired level. MPC [107] leverages both throughput estimates and buffer occupancy to choose bitrates by optimizing a given QoE metric over a future chunk horizon.

To implement the aforementioned ABR baselines, we utilize the open-source codes of GENET [103], which already include the implementation of the three algorithms. Furthermore, we re-use the pre-trained model weights of GENET[5] for our experiments.

Baselines for CJS. The following three baselines are implemented for the CJS task: Decima [63], first-in-first-out scheduling (labeled "FIFO") [87] and fair scheduling (labeled "Fair") [87]. Decima [63] is a RL model for job scheduling in the distributed computing cluster, which develops a graph neural network (GNN) to efficiently process DAG information of job properties (e.g., resource demands and dependency). Both FIFO and Fair are two common scheduling algorithms used by data processing system Spark [87]. The former schedules jobs in the order of their arrival and allocates the requested amount of resources to each job, while the latter schedules jobs in a "round robin" fashion to ensure that each job receives a roughly equal share of the cluster.

We utilize the PyTorch re-implementation of Decima [30] for our experiments as the original implementation is somewhat outdated. Furthermore, we make use of the pre-trained model weights of Decima[6] provided in [30]. Additionally, we adopt the implementation of FIFO and Fair from the same source [30].

A.4 Details of Simulation Settings

We generate different simulation environments with real-world and synthetic datasets for training and testing to comprehensively evaluate the performance of the LLM adapted by NetLLM against baselines. The detailed simulation settings for each task are explained as follows.

VP simulation. As shown in Table 2, by default, we train and test each method on a large-scale viewport dataset *Jin2022* [43] which records the viewport traces from 84 viewers[7] watching 27 60-second immersive videos. We randomly select 15 videos and 42 viewers for training, 6 videos and 21 viewers for validation, 6 videos and 21 viewers for testing. This results in a total of 882 traces for experiments. The historical window (hw) and prediction window (pw) are set to be 2 seconds and 4 seconds, respectively, for the default training and testing settings.

When evaluating generalization performance, we test each method on a new viewport dataset (i.e., new data distributions) and/or with a new prediction setup (i.e., increasing prediction difficulty). For instance, on *unseen setting2*, we evaluate each method on the new *Wu2017* dataset [97]. This dataset contains 9 videos[8] with an average length of 242 seconds watched by 48 viewers. We randomly sample 4 videos and 9 viewers from the dataset, resulting in 36 long viewport traces for testing generalization. As for *unseen setting1*, we increase pw to increase the prediction difficulty for each method. Following the setting in [80], we roughly set $hw \approx pw / 2$ across all settings. Changing the coefficient does not qualitatively affect the results.

ABR simulation. Table 3 summarizes the simulation settings for ABR. By default, we train and test all methods to stream the *Envivio-Dash3* video from the DASH-246 JavaScript reference client [25], whose format follows the GENET [103] and Pensieve [62] setting. We use the broadband *FCC* [18] traces as the default bandwidth dataset. In particular, we use the same traces for training and validation as those used by GENET, which compromise 235 traces for training and 150 traces for validation. Then, we randomly sample 100 traces from the remaining dataset for testing. This results in the use of more than 90 hours of bandwidth traces for experiments. To simulate environments for generalization testing, we follow the method in Pensieve [62] to generate a synthetic video

[5]https://github.com/GenetProject/Genet/tree/main/src/emulator/abr/pensieve/data/mahimahi_new_best_models/ADR_model

[6]https://github.com/ArchieGertsman/spark-sched-sim/tree/main/models/decima

[7]The *Jin2022* dataset originally contains the viewport traces from 100 viewers [43]. We filter out those incomplete ones (i.e., less than 60 seconds in duration) and finally use the traces from 84 viewers for experiments.

[8]The *Wu2017* dataset originally includes 18 videos. We use the first 9 videos as viewers are free to look around when watching these videos.

SynthVideo which shares a similar format of *Envivio-Dash3* but with a larger video bitrate. Besides, we also generate a new bandwidth dataset *SynthTrace* with 100 traces according to the method in Pensieve [62], which exhibits a larger bandwidth range and more dynamic fluctuation patterns than *FCC*.

CJS simulation. Table 4 provides the detailed information of the CJS simulation. Following Decima [63], we simulate different workload traces using a real-world dataset TPC-H [14] which contains job requests of large data volumes, high executor demands, and high degree of complexity. To be consistent with the settings used by the pre-trained Decima in [30], we set the number of job requests to be 200 and the number of executor resources (representing computation resources) to be 50k units as the default training and testing settings. To evaluate the generalization performance of each method, we simulate various unseen harder workloads by increasing the number of job requests and reducing the number of executor resources, as also done in Decima [63]. Note that in each setting the job requests are randomly sampled from the TPC-H dataset. Besides, when evaluating on the default testing setting, we have ensured that the job requests are different from those in the training setting. This can be easily done by setting different random seeds for data sampling.

A.5 Real-world ABR Testbed Setup

We leverage the testbed from GENET [103] to test the `NetLLM`-adapted Llama2 in a real-world client-server ABR system. The testbed modifies dash.js (version 2.4) to support BBA, MPC and GENET streaming algorithms. We further modify the dash.js to support the adapted Llama2. In our real-world tests, the client video player is a Google Chrome browser (version 87) and the video server (Apache version 2.7) runs on the same machine as the client. All tests are performed on our Linux server, with two different ports to emulate the ABR client and video server. We then use Mahimahi [70] to emulate different network environments from the broadband traces [18] and cellular mobile traces [84], along with an 80ms RTT, between the client and server. In particular, we randomly sample 100 traces from both the broadband and cellular mobile bandwidth datasets for network environment emulation.

A.6 Evaluation Metrics

Metric for VP. We use mean absolute error (MAE) as the evaluation metric for the VP task. Let $\boldsymbol{v}_p = (\alpha_t, \beta_t, \zeta_t)$ denote the viewport coordinate at timestep t, where α, β, ζ represent the roll, pitch and yaw values, respectively. Given $\boldsymbol{v}_t^p, \boldsymbol{v}_t^g$ as the predicted and ground-truth viewports and H as the prediction horizon, the MAE is calculated by:

$$MAE = \frac{1}{H}\sum_{t=1}^{H}\frac{|\alpha_t^p - \alpha_t^g| + |\beta_t^p - \beta_t^g| + |\zeta_t^p - \zeta_t^g)|}{3}$$

Metric for ABR. We use quality of experience (QoE) as the evaluation metric for the ABR task, which is defined as the weighted linear combination of three metrics [62, 103]:

$$QoE = \frac{\sum_{i=1}^{C}(Bitrate_i - \lambda Rebuf_i - \gamma BitrateChange_i)}{C}$$

where $Bitrate_i$ is the bitrate in Mbps of chunk i, $Rebuf_i$ is the rebuffering time in seconds of downloading chunk i, $BitrateChange_i$ is the bitrate change in Mbps between consequtive chunks, C is the number of chunks of the video and λ, γ are the weight parameters. Following Pensieve [62], we set $\lambda = 4.3$ and $\gamma = 1$.

Metric for CJS. We use job completion time (JCT) [63] as the evaluation metric for the CJS task. Let t_s denote the arrival time of a job and t_e denote the finishing time of a job. The JCT is calculated by:

$$JCT = t_e - t_s$$