



Designing Network Algorithms via Large Language Models

Zhiyuan He¹, Aashish Gottipati², Lili Qiu^{1,2}, Xufang Luo¹, Kenuo Xu³, Yuqing Yang¹, Francis Y. Yan^{1,4}

¹Microsoft Research, ²UT Austin, ³Peking University, ⁴UIUC

ABSTRACT

We introduce NADA, the first framework to autonomously design network algorithms by leveraging the generative capabilities of large language models (LLMs). Starting with an existing algorithm implementation, NADA enables LLMs to create a wide variety of alternative designs in the form of code blocks. It then efficiently identifies the top-performing designs through a series of filtering techniques, minimizing the need for full-scale evaluations and significantly reducing computational costs. Using adaptive bitrate (ABR) streaming as a case study, we demonstrate that NADA produces novel ABR algorithms—previously unknown to human developers—that consistently outperform the original algorithm in diverse network environments, including broadband, satellite, 4G, and 5G.

CCS CONCEPTS

• **Networks** → **Network algorithms**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Network Algorithms, Large Language Models

ACM Reference Format:

Zhiyuan He, Aashish Gottipati, Lili Qiu, Xufang Luo, Kenuo Xu, Yuqing Yang, and Francis Y. Yan. 2024. Designing Network Algorithms via Large Language Models. In *The 23rd ACM Workshop on Hot Topics in Networks (HotNets '24)*, November 18–19, 2024, Irvine, CA, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3696348.3696868>

*Lili Qiu and Francis Y. Yan are the corresponding authors.

*Aashish Gottipati and Kenuo Xu contributed to this work during their internships at Microsoft Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. *HotNets '24*, November 18–19, 2024, Irvine, CA, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1272-2/24/11

<https://doi.org/10.1145/3696348.3696868>

1 INTRODUCTION

Network control and adaptation algorithms have traditionally relied on human-designed heuristics or, more recently, reinforcement learning (RL). Notable examples of these algorithms include adaptive bitrate (ABR) streaming [8, 17, 19], congestion control (CC) [5, 18], and load balancing [7, 16].

As network technology rapidly evolves, there is a growing need for tailoring network algorithms to specific environments. For instance, ABR was originally designed for 3G and broadband networks [8, 19], but the advent of more dynamic 4G and 5G networks has prompted the development of novel, specialized ABR algorithms [6, 11, 13, 14]. Similarly, the emergence of Low Earth Orbit (LEO) satellite networks has further spurred customized algorithms [20].

However, developing new algorithms for constantly evolving network environments demands substantial expertise and effort. Motivated by the impressive generative power of large language models (LLMs), we explore the following question: *Can we leverage LLMs to automate the design of novel network algorithms tailored to diverse environments?* We propose that LLMs have the potential to dramatically accelerate innovation in network algorithm design.

The most straightforward approach to utilizing LLMs is by prompting them to generate new algorithm designs in natural language. However, after considerable experimentation, we find it challenging to have LLMs produce high-quality algorithm descriptions for a target environment (e.g., 5G or satellite networks). Although LLMs possess general knowledge about these networks, their responses (even with pseudocode) are often too broad and lack necessary details, making it difficult to validate the proposed ideas.

Rather than relying on LLMs to generate algorithm descriptions, we turn to their remarkable code generation capabilities. Recent studies have shown that LLMs are proficient at producing code from human instructions [1, 4, 21]. Nevertheless, the code they generate may fail to compile or execute, contain design flaws, or perform poorly in practice. Without efficient mechanisms to evaluate the quality of LLM-generated algorithms (code implementations), the cost of testing would be prohibitively expensive.

We present NADA (Network Algorithm Design Automation via LLMs), a generic framework aimed at automating the development of novel network algorithms using LLMs. NADA is applicable to any network algorithm that satisfies

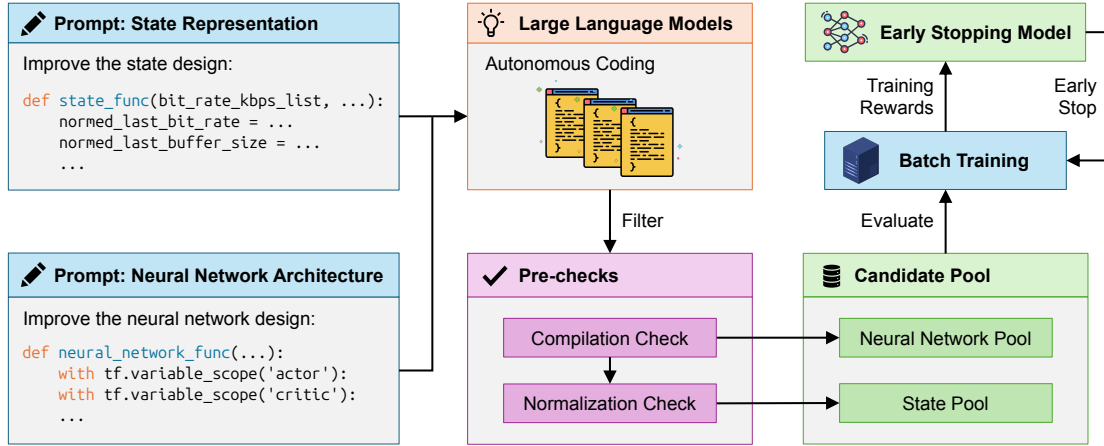


Figure 1: NADA workflow. It leverages LLMs to generate a wide range of alternative designs for a network algorithm and employs a series of filtering techniques to efficiently select the most promising designs for further evaluation.

two basic conditions: it has a functional code implementation, and its performance can be measured through a network simulator or emulator. A broad range of network algorithms satisfy these requirements, and in this paper, we use ABR algorithms in video streaming as a case study.

A widely tested ABR algorithm that meets the above criteria is Pensieve [8]. It is based on deep RL, with an architecture illustrated in Figure 2. Before applying NADA, we first identify two key components in Pensieve’s design—the RL state representation and the neural network architecture. Starting from the existing functions (code blocks) that implement these components, NADA instructs and stimulates LLMs to generate diverse design alternatives (also in the form of code blocks), using carefully crafted prompting strategies (§2.1). Next, to efficiently and accurately evaluate a large volume of LLM-generated designs without incurring excessive computational costs, NADA employs a series of filtering techniques, including a compilation check, a normalization check, and an early-stopping mechanism, to proactively terminate the evaluation of unpromising designs (§2.2). Only the remaining designs—a small subset of the total—are then evaluated at full scale (i.e., trained until convergence). This approach substantially lowers the overall computational costs.

To assess the effectiveness of NADA, we gather real-world traces from various network environments, including broadband, satellite, 4G, and 5G. In each scenario, we find that NADA is able to generate ABR algorithms that outperform Pensieve’s original design (§3). Some of these LLM-generated algorithms offer novel insights into the design of RL-based ABR algorithms, particularly with regard to normalization strategies and feature engineering for RL states (§4).

In summary, this paper outlines the process of using LLMs to design novel network algorithms (Figure 1). The proposed framework, NADA, solicits a wide array of alternative designs

from LLMs based on an existing algorithm, and then employs filtering techniques to efficiently evaluate their performance and identify the most promising designs. Using ABR as a case study, we showcase the potential of NADA to create network algorithms that outperform existing solutions. Moving forward, we plan to extend NADA to other network algorithms, such as congestion control [5], and explore its applicability to non-RL methods. We hope this work paves the way for further research and ultimately transforms how network algorithms are developed in the future.

2 OUR APPROACH

2.1 Generating Diverse Designs with LLMs

We apply NADA to the well-known ABR algorithm Pensieve [8], generating alternative algorithm designs that improve performance with the assistance of LLMs. Throughout this paper, we use Pensieve as a case study to demonstrate our methodology. However, we note that NADA is not confined to this example; it can be applied to a broader range of network algorithms, especially RL-based ones.

Pensieve leverages an RL method known as the “actor critic,” as shown in Figure 2. After streaming each video chunk t , Pensieve constructs a state $s_t = (\vec{x}_t, \vec{\tau}_t, \vec{n}_t, b_t, c_t, l_t)$ to capture the surrounding network environment. In this state, the vectors \vec{x}_t , $\vec{\tau}_t$, and \vec{n}_t represent the past network throughput measurements, the previous download times of video chunks, and the available sizes of the next chunk at different bitrates, respectively. The variables b_t , c_t , and l_t correspond to the current playback buffer size, the number of remaining chunks in the video, and the last selected bitrate. Then, the state s_t is input into an actor-critic neural network. The actor network determines the probability of selecting a

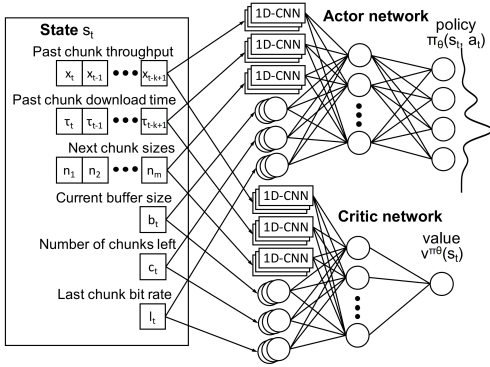


Figure 2: The original algorithm design of Pensieve [8].

particular bitrate for the next chunk, while the critic network estimates the expected reward achievable from s_t .

It can be seen from Figure 2 that the Pensieve algorithm is built around two essential components: the RL state representation and the actor-critic neural network architecture, both hand-designed and manually implemented through Python functions (code blocks). Given these existing code blocks, NADA first aims to guide LLMs in generating a wide array of alternative code blocks that potentially encapsulate novel state designs and neural network architectures. The main objective is to stimulate diversity and creativity in the algorithm designs generated by LLMs, thereby increasing the chances of producing high-quality solutions.

Through experimentation, we identified several effective prompting strategies. First, we instruct LLMs to analyze existing code, generate multiple ideas in natural language, and then select the best idea before proceeding to code generation. This method, known as Chain-of-Thought (CoT) [15], enhances the LLM’s reasoning capabilities and leads to more diverse outputs. Second, we rename the original variables, i.e., the parameters of state and neural network functions, to more semantically meaningful names. We further explain their roles both in the prompt and through detailed code comments. While not strictly necessary, this revision and annotation process helps LLMs better understand the problem and generate higher-quality solutions. Lastly, we observe that LLMs sometimes generate state designs with improperly normalized features, which hinders convergence and degrades performance. To mitigate this issue, we explicitly request proper normalization in our state generation prompts. This strategy does not apply to neural network architectures. The complete set of prompts is released at <https://github.com/hzy46/NADA>.

2.2 Filtering and Evaluating Designs

The state representations and neural network architectures generated by LLMs often fall short of expectations. Given the large number of algorithm designs produced by LLMs, the

main challenge is to efficiently and accurately evaluate them while identifying promising candidates. To address this challenge, we develop three filtering strategies aimed at reducing training and evaluation costs. The first two strategies serve as pre-checks: an initial compilation (or execution) check to filter out code with syntax errors, and an empirical heuristic to remove states with unnormalized features. The third strategy implements an early stopping mechanism, using a predictive model to terminate the training of unpromising designs before they fully complete. These techniques enable early identification of flawed designs, minimizing unnecessary computational costs without overlooking promising designs. Next, we elaborate on the design of the pre-checks and the early stopping mechanism.

The compilation check involves a trial run of the LLM-generated code. Any code that triggers an exception is immediately excluded from further consideration. Following this, a normalization check is applied to the generated states. We observe that LLMs sometimes use features like chunk sizes in bytes, which can result in abnormally large values (e.g., over one million for megabytes), hindering the convergence of the training process. To eliminate state designs with improperly normalized features, we test the code with random inputs (“fuzzing”), and check whether any output contains a feature value exceeding a predefined threshold T (set to 100 in our study). State designs that fail this test are rejected. This normalization check is applied only to state generation code, not the code that defines the neural network architecture.

Once an LLM-generated design passes both the compilation and normalization checks, NADA proceeds to train it in a network simulator (or emulator). However, RL training is computationally expensive, requiring numerous epochs to reach convergence. To reduce the cost, we introduce an early stopping model—a binary classifier—that predicts whether the training trajectory in the early stages is likely to result in a performant algorithm. Specifically, this early stopping model utilizes the training rewards from the first K episodes to learn a 1D-CNN (one-dimensional convolutional neural network) as the binary classifier. If the classifier predicts that a particular algorithm design is unlikely to rank among the top performers, NADA will early-stop its training.

Ideally, the early stopping model would filter out all but the top-performing designs, such as the top 1%. However, labeling only the top 1% of designs as positive in the training data leads to poor classification performance due to the significant class imbalance between the positive class (1%) and the negative class (99%). To address this imbalance, we employ a variant of label smoothing [9]. Instead of labeling only the top 1% as positive, we expand the positive label to the top 20%. This adjustment reduces class skew and enables the early stopping model to learn more distinguishing characteristics of high-performing designs. Then, we revert to the

Dataset	Train Traces	Train Hours	Test Traces	Test Hours	Throughput	Train Epochs	Test Interval
FCC	85	10.0	290	25.7	1.3	40,000	500
Starlink	13	0.9	12	0.8	1.6	4,000	100
4G	119	10.0	121	10.0	19.8	40,000	500
5G	117	10.0	119	10.0	30.2	40,000	500

Table 1: Network traces used in our study. “Train Traces” and “Test Traces” are the number of traces in the training and testing splits, respectively. “Train Hours” and “Test Hours” are the total duration of the traces measured in hours. “Throughput” represents the average throughput in Mbps. The last two columns show the number of training epochs and the intervals at which model checkpoints are evaluated on the corresponding test sets.

original label assignment (top 1% as positive), and fine-tune the model’s classification threshold on the training set, i.e., predicting a positive (or negative) label if the model’s output score is above (or below) the threshold. Since overlooking a performant design has a worse impact than unnecessarily evaluating a suboptimal design, the threshold is increased to maximize the true negative rate (unpromising designs correctly early-stopped) while maintaining a 0% false negative rate (top-performing designs correctly preserved).

We compare this model with alternative predictive methods and report results in §3.4. Our results indicate that the early stopping model can correctly early-stop 87% of previously unseen designs without prematurely rejecting any of the top 5 performing designs.

3 EVALUATION

3.1 Experiment Settings

We perform a trace-driven evaluation using the following trace datasets. Details are presented in Table 1.

- **FCC:** This dataset represents measurements of the U.S. broadband network as recorded by the FCC [2].
- **4G and 5G:** We create these two datasets by measuring downlink throughput from 4G and 5G networks in the U.S.
- **Starlink:** We collect throughput traces from a stationary Starlink RV terminal located in the U.S. While Starlink’s bandwidth can support high-resolution video streaming during off-peak hours, it decreases significantly during peak hours due to shared usage of satellite links. To simulate this condition, we reduce the link capacity in the Starlink traces to one-eighth of its original speed.

We adopt the same video streaming configurations as in Pensieve [8], including the same bitrate levels of {300, 750, 1200, 1850, 2850, 4300} kbps when evaluating on the FCC and Starlink datasets. However, since our 4G and 5G datasets exhibit much higher bandwidth, we elevate the bitrate ladder to {1850, 2850, 4300, 12000, 24000, 53000} kbps. This bitrate ladder follows YouTube’s recommended video encoding settings [3]. The same quality of experience (QoE) function from Pensieve (“ QoE_{lin} ”) is adopted as the reward.

NADA	Total	Compilable	Well Normalized
w/ GPT-3.5	3,000	1,237 (41.2%)	822 (27.4%)
w/ GPT-4	3,000	2,059 (68.6%)	1,505 (50.2%)

Table 2: Number of ABR designs generated by NADA using GPT-3.5 and GPT-4 that successfully pass the compilation check and the normalization check.

On each trace dataset, we train both the original Pensieve and the novel designs generated by NADA, allowing for algorithm customization in different network environments. Two LLMs are tested with NADA—GPT-3.5 and GPT-4. To reduce the influence of random noise, we perform five independent training sessions for each design, with each session initialized using a different random seed. During each session, we periodically evaluate model checkpoints on the *test traces* and calculate the average reward from the last 10 checkpoints. The median of these smoothed rewards from the five sessions is reported as the final “test score” (or simply “score”). Table 1 lists the number of training epochs and the frequency of checkpoint testing.

3.2 Designing States

We run NADA on GPT-3.5 and GPT-4 to generate 3,000 states each. The statistics in Table 2 show that 68.6% of the state functions generated by GPT-4 are “compilable,” i.e., they execute without errors, compared with 41.2% for GPT-3.5. Meanwhile, 50.2% of the states produced by GPT-4 contain well-normalized features, whereas only 27.4% of those from GPT-3.5 do. These results highlight GPT-4’s superior capability in generating correct and desired code blocks.

The alternative state designs proposed by GPT can be non-trivial. We find that GPT introduces not only basic features, such as bitrate variance and the exponential moving average of throughput, but also imports additional Python packages to implement more advanced functionality. For instance, some states use the linear regression model from the `statsmodel` package to predict future throughput. In another example, the Savitzky-Golay filter [12] from the `scipy` package is applied to analyze buffer size trends based on



Figure 3: Test performance of the best states generated by NADA using GPT-3.5 and GPT-4, compared with the original state design throughout the training process. NADA consistently produces state representations that outperform the original design across four network trace sets in simulation.

Dataset	Method	Score	Impr.
FCC	Original	1.070	–
FCC	w/ GPT-3.5	1.089	1.7%
FCC	w/ GPT-4	1.090	1.9%
Starlink	Original	0.308	–
Starlink	w/ GPT-3.5	0.472	52.9%
Starlink	w/ GPT-4	0.482	56.3%
4G	Original	11.705	–
4G	w/ GPT-3.5	13.226	13.0%
4G	w/ GPT-4	14.973	27.9%
5G	Original	27.848	–
5G	w/ GPT-3.5	28.447	2.2%
5G	w/ GPT-4	28.636	2.8%

Table 3: Test performance of the best states generated by NADA using GPT-3.5 and GPT-4 after the training completes. Network traces are replayed in simulation.

historical data. In contrast, the original state representation in Pensieve does *not* utilize buffer size history in any form.

In Figure 3, we compare the best states generated by NADA using GPT-3.5 and GPT-4 against the original state design. The test scores (as defined in §3.1) are plotted throughout the training sessions for each network trace set. Table 3 provides a summary of the final test scores after the maximum number of training epochs. These results show that NADA, when applied with both GPT-3.5 and GPT-4, consistently generates state representations that outperform the original design, with GPT-4 demonstrating a more significant overall improvement, especially on the Starlink traces.

In addition, we conduct emulation experiments using the dash.js framework to stream video in a real web browser over Mahimahi [10]. The results for the Starlink, 4G and 5G

Dataset	Method	Score	Impr.
Starlink	Original	−0.0482	–
Starlink	w/ GPT-3.5	0.0899	286.5%
Starlink	w/ GPT-4	0.0759	257.5%
4G	Original	4.976	–
4G	w/ GPT-3.5	8.010	61.0%
4G	w/ GPT-4	9.233	85.6%
5G	Original	17.26	–
5G	w/ GPT-3.5	17.43	1.0%
5G	w/ GPT-4	21.55	24.9%

Table 4: Emulation results of the best generated states.

traces are shown in Table 4 (we did not evaluate on FCC as the simulation improvements were already statistically insignificant). Despite discrepancies in the emulation and simulation results, the optimal states generated by NADA continue to outperform the original design. In Section 4, we elaborate on the best states generated for each network scenario and provide insights into their design.

3.3 Designing Neural Networks

Due to budget constraints, our investigation into the neural network architecture is restricted to GPT-3.5. We run NADA on GPT-3.5 to generate 3,000 alternative architectures and apply the compilation check to filter out invalid designs (the normalization check is not applicable here). Among the generated neural networks, 760 architectures pass the compilation check. Figure 4 compares the most effective architectures with the original design. Notably, more pronounced improvements are observed on the Starlink, 4G, and 5G traces, whereas the improvement on FCC is not statistically significant. Overall, we find that modifying the neural

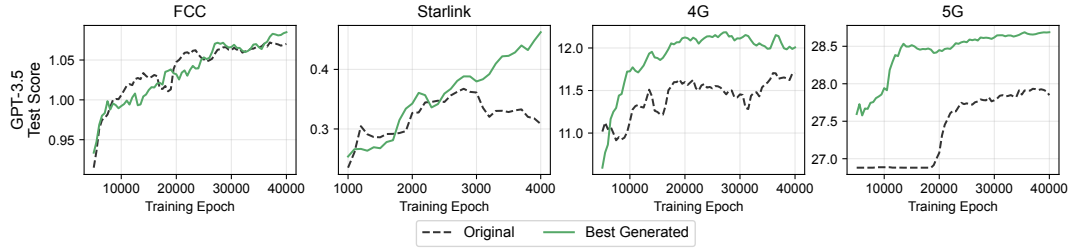


Figure 4: Test performance of the best generated neural network architectures vs. the original in simulation.

Dataset	State	Neural Net	Combined
FCC	1.7%	1.4%	2.2%
Starlink	52.9%	50.0%	61.1%
4G	13.0%	2.6%	16.5%
5G	2.2%	3.0%	3.1%

Table 5: Results of combining the states and neural networks generated by NADA with GPT-3.5.

network architecture tends to yield smaller gains than revising the state. The emulation results are omitted here.

Furthermore, we explore the performance improvements by combining novel states with newly generated neural network architectures. Specifically, we select the top 30 states and the top 30 neural networks generated by GPT-3.5, creating 900 unique combinations. Each state-architecture combination is trained five times, and the best results are presented in Table 5. We find that this combination leads to consistent improvements, with gains up to 61.1% on the Starlink traces. Nevertheless, the combined improvements are relatively modest compared with the individual gains from updating states or neural networks alone.

3.4 Early Stopping Mechanism

In this section, we introduce and assess five candidate mechanisms for early stopping during training. We consider alternative designs, including novel states or neural network architectures, that fall within the top 1% of training rewards as candidates worth full training. These top designs are labeled as positive, while the remaining ones are labeled negative. The methods tested are as follows: (1) “Reward Only”: Utilizing the first 10k training rewards to learn a 1D-CNN classifier; (2) “Text Only”: Embedding the code using OpenAI’s `text-embedding-ada-002` model as input to the trained classifier; (3) “Text + Reward”: Using the previous two features as inputs to the classifier; (4) “Heuristic Max”: Early stopping based on the maximum reward in the first 10k epochs; (5) “Heuristic Last”: Early stopping based on the reward in the final epoch.

We first collect 2000 algorithm designs along with their corresponding training metrics, including ground-truth labels, and conduct a five-fold cross validation. In each fold, 20% of the designs, or 400 samples, are used for training. We report two metrics across all validation folds and network environments: the false negative rate—fraction of top-performing designs incorrectly reject, and the true negative rate—fraction of suboptimal designs correctly stopped early.

On the samples for testing, Figure 5 shows that “Reward Only” offers the best trade-off between early stopping errors (left panel) and resource savings (right panel). Specifically, “Reward Only” successfully terminates 87% of suboptimal designs with an incorrect rejection rate of only 12% on average. We also manually confirmed that the top five algorithms are never missed. This translates to computational savings on the order of hundreds of millions of training epochs.

4 INSIGHTS FROM GENERATED DESIGNS

In this section, we describe the best designs generated by NADA using GPT-3.5 and GPT-4 for each network environment, with a focus on the innovative state designs and the key insights gained from them. We then provide a short summary of the changes introduced in the neural network architectures before concluding this section.

FCC: On the FCC traces, we observe that the optimal states generated by NADA using both GPT-3.5 and GPT-4 involve modifying the normalization strategy for certain features. While the original normalization range is $[0, 1]$, the optimal states remap these features to $[-1, 1]$.

Starlink: NADA with GPT-3.5 exploits the smaller size of the Starlink dataset and removes two variables from the state representation: the download times of previous video chunks, and the size options for the next chunk. This approach seems to reduce overfitting and showcases NADA’s ability to autonomously customize network algorithms based on environmental complexity (reflected as the number of traces in our study), but we did not empirically verify this claim. In comparison, GPT-4 employs more aggressive normalization with an increased normalizing factor and smooths the throughput and download times.

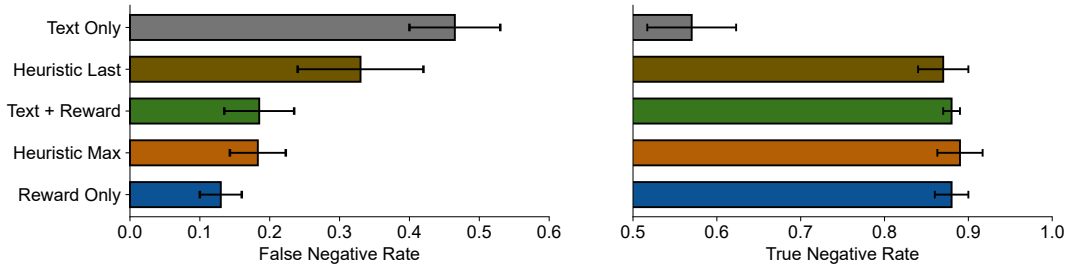


Figure 5: Comparison between different early stopping classifiers. False and true negative rates are defined in §3.4.

4G: On the 4G traces, the original state tends to favor lower bitrates, leading to lower rewards. Consequently, the optimal states generated by NADA introduce new features that promote higher bitrate selection when the video playback is sufficiently buffered. GPT-3.5, for instance, applies a linear regression model to predict the download time of future chunks and incorporates the trends of throughput and download time into the state. GPT-4 introduces the historical trend of playback buffer size, signaling the model to increase the bitrate as the buffer grows. In contrast, the original state design does not take buffer size history into account at all.

5G: The best states for the 5G traces are similar to those for 4G. GPT-3.5 introduces a predicted throughput feature, while GPT-4 adds the buffer size difference between adjacent time steps. These enhancements allow the model to make more informed bitrate decisions and achieve higher rewards.

Summary: LLMs have suggested several principles for designing the states of RL-based ABR algorithms. First, selecting an appropriate normalization strategy (with a different normalization range or normalizing factor) can enhance model performance. Second, removing unnecessary state features *might* reduce overfitting particularly in simpler target environments. Third, even though a 1D-CNN can implicitly capture past throughputs and download times, explicitly summarizing their trends or predicting future values as additional features may still provide benefits. We hypothesize that this extra layer of feature engineering helps preserve important signals amid noisy data. Finally, and perhaps most intriguingly, Pensieve has overlooked the relevance of buffer size history in ABR. In contrast, NADA reveals that incorporating features like buffer size trends or differences (between adjacent time steps) leads to noticeable improvements.

Next, we briefly summarize the key changes introduced by the best generated neural network architectures. For the FCC traces, the number of hidden neurons in the fully connected network is increased to 256, and the activation function is switched to Leaky ReLU. For Starlink, an RNN is used in place of a 1D-CNN, while in 4G, an LSTM is used instead. For the 5G dataset, the actor and critic networks share the

hidden layer but retain separate output layers. Complete results are available at <https://github.com/hzy46/NADA>.

5 DISCUSSION

Through our exploration of applying NADA to ABR algorithms, we have learned the following lessons. First, directly applying LLMs to optimize large, complex programs proves challenging, while optimizing individual functions (e.g., states or neural networks) enables more manageable and targeted improvements. Second, LLMs can generate creative design alternatives, but not all suggestions are useful. Therefore, it is essential to develop efficient filtering mechanisms to quickly assess the quality of LLM-generated designs.

Our work demonstrates the potential of LLMs in designing network algorithms, and we identify several promising future directions: (1) We plan to extend the case study from ABR to other network algorithms, such as congestion control. (2) While our current focus is on enhancing RL-based algorithms, we believe NADA can be adapted to generate other types of network algorithms, although different filtering techniques may be required. (3) LLMs have demonstrated the ability to propose creative algorithmic modifications, but their proposals lack completeness and rigor. Moving forward, we aim to integrate LLMs with program synthesis or neural architecture search (NAS) to systematically explore the design space. (4) Lastly, we intend to refine our prompting strategies and more effectively harness the reasoning capabilities of LLMs to reduce the number of initial candidates, while still maintaining diversity and quality. This will improve the overall efficiency of our framework.

6 CONCLUSION

In this paper, we presented NADA, a framework that leverages LLMs to develop novel network algorithms tailored to diverse network environments. Using ABR as a case study, we demonstrated that NADA effectively generated novel RL state designs and neural network architectures that consistently outperformed the original ABR design in different network environments. In future work, we plan to extend our framework beyond ABR to other network algorithms.

REFERENCES

- [1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
- [2] FCC. 2024. Measuring Broadband America. <https://www.fcc.gov/general/measuring-broadband-america>. [Accessed 10-03-2024].
- [3] Google. 2024. YouTube recommended upload encoding settings. <https://support.google.com/youtube/answer/1722171?hl=en>. [Accessed 10-03-2024].
- [4] Sirui Hong, Xiwu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352* (2023).
- [5] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*.
- [6] Dhyanjay Kumar, S. Aishwarya, A. Srinivasan, and L. Arun Raj. 2016. Adaptive video streaming over HTTP using stochastic bitrate prediction in 4G wireless networks. In *2016 ITU Kaleidoscope: ICTs for a Sustainable World (ITU WT)*.
- [7] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Mehrdad Khani Shirkoobi, Songtao He, Vikram Nathan, et al. 2019. Park: An open platform for learning-augmented computer systems. *Advances in Neural Information Processing Systems* (2019).
- [8] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*.
- [9] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. 2019. When does label smoothing help? *Advances in Neural Information Processing Systems* (2019).
- [10] 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 '15)*.
- [11] Eman Ramadan, Arvind Narayanan, Udhaya Kumar Dayalan, Rostand AK Fezeu, Feng Qian, and Zhi-Li Zhang. 2021. Case for 5G-aware video streaming applications. In *Proceedings of the 1st workshop on 5G measurements, modeling, and use cases*.
- [12] Abraham Savitzky and Marcel J. E. Golay. 1964. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry* (1964).
- [13] Anh-Tien Tran, Nhu-Ngoc Dao, and Sungrae Cho. 2020. Bitrate adaptation for video streaming services in edge caching systems. *IEEE Access* (2020).
- [14] Mehmet Fatih Tuysuz and Mehmet Emin Aydin. 2020. QoE-based mobility-aware collaborative video streaming on the edge of 5G. *IEEE Transactions on Industrial Informatics* (2020).
- [15] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* (2022).
- [16] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. 2022. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*.
- [17] 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 *USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*.
- [18] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *USENIX Annual Technical Conference (ATC '18)*.
- [19] 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 ACM SIGCOMM 2015 Conference*.
- [20] Jinwei Zhao and Jianping Pan. 2023. QoE-driven joint decision-making for multipath adaptive video streaming. In *2023 IEEE Global Communications Conference (GLOBECOM)*.
- [21] Shuyan Zhou, Uri Alon, Frank F. Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. 2022. DocPrompting: Generating code by retrieving the docs. *arXiv preprint arXiv:2207.05987* (2022).