# Reagir ou Antecipar? Uma Comparação entre HPA e ML para Balanceamento de Carga

Carlos Veeck, Maria Barbosa, Kelvin Dias

***Resumo*— Com o crescimento dos usuários na rede 5G e futuros sistemas 6G, é essencial garantir uma rede escalável que atenda aos requisitos mínimos de QoS, como baixa latência em *live streaming*. Contudo, a escalabilidade em ambientes virtualizados aumenta o consumo de recursos computacionais. Este artigo propõe uma solução proativa para balanceamento de carga com instanciação dinâmica de UPFs baseada em redes neurais recorrentes. A proposta é comparada com a abordagem reativa baseada no Horizontal Pod Autoscaling (HPA) em uma infraestrutrura virtualizada 5G orquestrada pelo Kubernetes.**

***Palavras-Chave*— Monitoramento, HPA, Balanceamento de Carga, Aprendizagem de Máquina.**

***Abstract*— With the growth of users in 5G and future 6G systems, it is essential to ensure a scalable network that meets minimum QoS requirements, such as low latency in live streaming. However, scalability in virtualized environments increases the consumption of computational resources. This paper proposes a proactive solution for load balancing with dynamic instantiation of UPFs based on recurrent neural networks. The proposed approach is evaluated against a reactive strategy that utilizes Horizontal Pod Autoscaling (HPA) within a virtualized 5G infrastructure orchestrated by Kubernetes.**

***Keywords*— Monitoring, HPA, load balancing, machine learning.**

## I. Introdução

Com o crescimento contínuo no número de usuários conectados à rede 5G e com a chegada iminente das redes 6G, torna-se cada vez mais essencial garantir que a infraestrutura de rede seja suficientemente escalável [1]. Essa escalabilidade é fundamental para assegurar o atendimento aos requisitos mínimos de Qualidade de Serviço (QoS), especialmente em aplicações sensíveis à latência, como transmissões ao vivo (*live streaming*), que demandam respostas em tempo quase real [2].

Entretanto, alcançar essa escalabilidade em ambientes virtualizados impõe desafios significativos, particularmente no que diz respeito ao consumo de recursos de *hardware*. Nesse contexto, surge a necessidade de realizar a instanciação dinâmica de *User Plane Functions* (UPFs) durante o tempo de operação, de forma progressiva, conforme o aumento da carga de tráfego. Essa abordagem visa otimizar o balanceamento de carga no plano de dados, contribuindo para manter o desempenho satisfatório da rede mesmo em situações de alta demanda [3][4].

Na literatura, existem diferentes abordagens para a realização do autoescalonamento de recursos da rede 5G, tais como as proativas e as reativas. As abordagens reativas estão relacionadas ao *Horizontal Pod Autoscaling* (HPA), que permite determinar previamente o momento de disparo de novos pods[1] quando um limiar é atingido. Ou seja, nessa estratégia, a carga precisa aumentar para que haja a criação de novos pods [5].

Já nas abordagens proativas [6][7][8], busca-se antecipar a previsão de aumento da carga, a fim de criar novos pods antes que o crescimento ocorra, garantindo, assim, maior estabilidade à rede. Nesse contexto, o uso de aprendizado de máquina torna-se essencial, pois possibilita antecipar cenários críticos que podem degradar a QoS das aplicações. Nesse contexto, modelos de aprendizagem de máquina são ferramentas de predição importantes para lidar com a natureza dinâmica de cargas em redes 5G e futuros sistemas 6G.

Dessa forma, as contribuições deste trabalho serão listadas a seguir:

1) Solução proativa para balanceamento de carga com instanciação dinâmica de UPFs baseada em redes neurais recorrentes, mais especificamente, o modelo *Gated Recurrent Unit* (GRU). O GRU possui a vantagem de um menor custo computacional, mantendo um desempenho comparável, ou até superior, ao de modelos mais complexos na etapa de predição.
2) Comparação da solução proposta com a abordagem reativa baseada no Horizontal Pod Autoscaling (HPA) em uma infraestrutrura virtualizada 5G baseada em Kubernetes, usando a plataforma Open Air Interface[2] (OAI).

As próximas seções estão organizadas da seguinte maneira: a Seção II apresenta os conceitos fundamentais que embasam este trabalho, bem como a solução proposta. Na Seção III, são discutidos os resultados obtidos a partir da avaliação de desempenho da solução. Por fim, a Seção IV traz as conclusões do estudo e sugestões para trabalhos futuros.

## II. Solução proposta

Esta seção apresenta a arquitetura geral da solução proposta, incluindo a descrição da implementação baseada em HPA e a formulação do modelo preditivo com GRU.

A arquitetura proposta neste estudo é apresentada na Figura 1.O plano de controle contempla funções de rede responsáveis pelo gerenciamento da mobilidade, estabelecimento de sessão e armazenamento das informações do usuário. No entanto, vale ressaltar que o foco central do artigo está no plano de dados, representado pela função de rede UPF. O UPF é responsável

Carlos Veeck, Centro de Informática, UFPE, Recife-Pernambuco, e-mail: chvmv@cin.ufpe.br; Maria Barbosa, Centro de Informática, UFPE, Recife-Pernambuco, e-mail: mksb@cin.ufpe.br; Kelvin Dias, Centro de Informática, UFPE, Recife-Pernambuco, e-mail: kld@cin.ufpe.br; This work was partially supported by Nº 26/23 FADE/UFPE/FINEP (01.23.0528.00 -FINEP) REF. 2849/22 - (CONVÊNIO Nº 74/2023 - UFPE) 23076.100425/2023-24

[1]https://kubernetes.io/docs/concepts/workloads/pods/

[2]https://openairinterface.org/

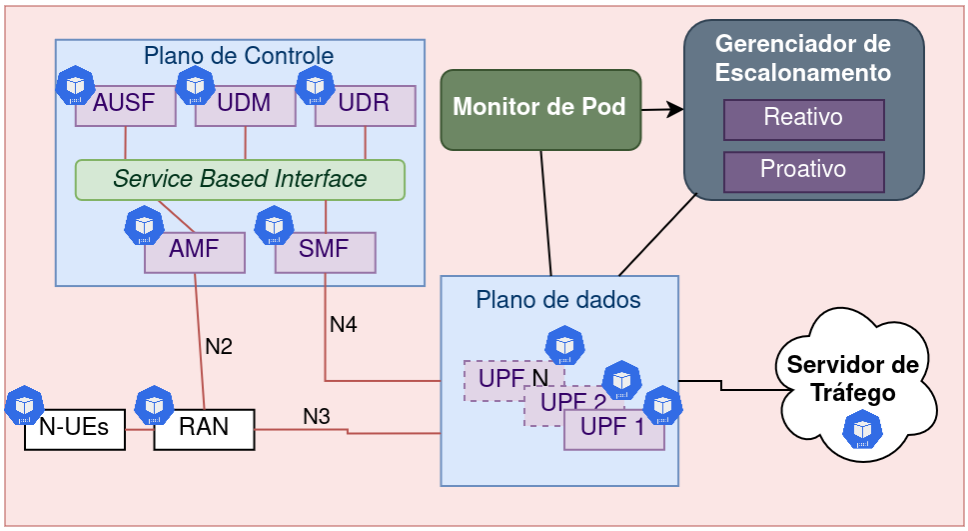


Fig. 1. Diagrama da arquitetura de rede 5G proposta.

pelo encaminhamento do tráfego e dos pacotes de dados dos usuários para a rede de dados (*internet*).

Nesta arquitetura, a criação das novas instâncias do UPF é gerenciado por um *Gerenciador de Escalonamento*, que administra réplicas dos Pods dessa função de rede com o objetivo de manter um fluxo de dados contínuo e eficiente. Contudo, para que o gerenciador possa entrar em ação, ele precisa das informações de consumo computacional dos pods UPF existentes. Sendo assim, para monitorar a carga de trabalho em cada pod, é utilizado um monitor de pods que coleta métricas por meio da API do *Metrics Server* do Kubernetes.

O Gerenciador de Escalonamento opera em dois modos distintos: um modo reativo, baseado no mecanismo nativo do Kubernetes, o HPA; e um modo proativo, que emprega um modelo de aprendizado de máquina desenvolvido especificamente para prever variações na demanda e agir de forma antecipada.

Para validar a proposta, foi desenvolvido um cenário de teste composto por um servidor de tráfego, que hospeda uma aplicação de *streaming* de vídeo implementada com Flask[3]. Nesse cenário, um equipamento transmissor envia o vídeo *Big Buck Bunny*, na resolução 320x180, para o servidor Flask, o qual é consumido simultaneamente por $n = \{5, 10\}$ equipamentos de usuário simulados (UEs) receptores.

### A. Mecanismo Reativo

O *Horizontal Pod Autoscaler* (HPA) é um recurso nativo do Kubernetes que permite o ajuste automático da quantidade de réplicas de um pod com base na demanda por recursos computacionais no cluster. Esse mecanismo possibilita que o ambiente Kubernetes reaja dinamicamente a variações de carga, escalando horizontalmente os pods conforme necessário. A lógica de escalonamento é baseada em métricas previamente definidas pelo desenvolvedor, geralmente especificadas em um manifesto de implantação (deployment manifest). Tais métricas podem incluir o uso de CPU, memória, rede ou métricas personalizadas expostas por meio de APIs. No manifesto, é possível configurar qual pod será monitorado, os limites mínimo (minReplicas) e máximo (maxReplicas) de réplicas permitidos, além do limiar de utilização (*averageUtilization*) que determina quando o escalonamento deve ocorrer.

TABELA I
CONFIGURAÇÕES DO HPA UTILIZADAS NOS TESTES

| Parâmetro | Valor | Descrição |
|---|---|---|
| `minReplicas` | 1 | Mínimo de réplicas mantidas pelo HPA. |
| `maxReplicas` | {2,3} | Máximo de réplicas que o HPA pode escalar. |
| `averageUtilization` | 50% | Limiar de uso da CPU para escalonamento. |

### B. Mecanismo Proativo

Esta seção tem como objetivo descrever a metodologia empregada para conceber o modelo GRU, incluindo a seleção de hiperparâmetros, o número de camadas e neurônios. O modelo foi desenvolvido utilizando a linguagem de programação Python, especificamente a biblioteca Keras (3.9.2) do TensorFlow (2.19.0).

Redes Neurais Recorrentes (RNNs) são redes neurais projetadas para lidar com dados sequenciais, mantendo um estado oculto que armazena informações do passado. Isso permite que as RNNs identifiquem dependências e padrões ao longo do tempo. No entanto, enfrentam problemas como perda de memória de curto prazo e o desaparecimento do gradiente em sequências longas. Para superar essas limitações, versões aprimoradas como a GRU são utilizadas, pois possuem mecanismos que decidem quais informações devem ser mantidas para prever corretamente eventos futuros.

A fim de aprimorar o desempenho do modelo GRU para a tarefa de previsão, realizamos a busca dos hiperparâmetros utilizando a biblioteca *Optuna*. A função objetivo foi definida com base no erro quadrático médio (MSE) entre as predições e os valores reais.

Durante a busca, foram avaliadas combinações dos seguintes hiperparâmetros: tamanho da janela de entrada (*look-back*),

[3]`https://flask.palletsprojects.com/`

taxa de aprendizado, tamanho do *batch*, função de ativação, otimizador e número de épocas. A Tabela II apresenta os valores testados para cada parâmetro, com destaque em negrito para os que obtiveram o melhor desempenho.

Os melhores hiperparâmetros encontrados foram: *look-back* = 10, *learning rate* = 0.000175, *batch size* = 8, função de ativação = relu, otimizador = Adam, e número de épocas = 100. Para esses valores de *Mean Absolute Error* (MAE) foi de 0,32 e de MSE foi de 20,68.

TABELA II
VALORES TESTADOS PARA OS HIPERPARÂMETROS.

| Hiperparâmetro | Valores Testados |
|---|---|
| Look-back | 5, **10**, 15, 20, 25, 30 |
| Taxa de aprendizado | $10^{-5}$ a $10^{-2}$ |
| Batch size | **8**, 16, 32 |
| Função de ativação | **`relu`**, `tanh` |
| Otimizador | **`Adam`**, `RMSprop` |
| Épocas | 50, **100**, 150, 200 |

Após a seleção do modelo cujos hiperparâmetros apresentaram a menor taxa de erro, o mesmo foi submetido a uma validação, utilizando dados diferentes dos utilizados para o treinamento, visando verificar a eficiência do modelo. Sendo assim a Figura 2 apresenta os valores previstos pelo modelo, sobrepondo os valores reais, indicando a eficiência do modelo gerado.

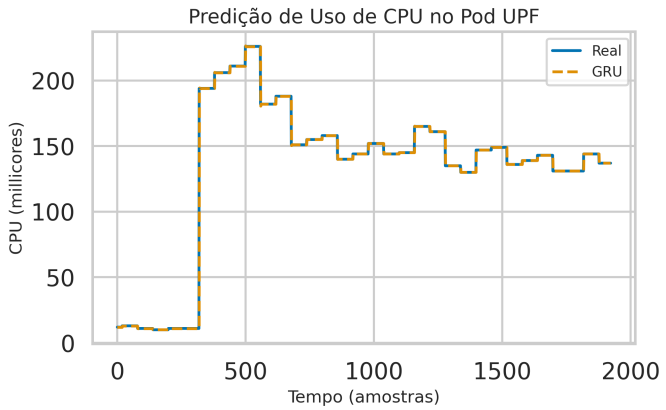


Fig. 2. Resultados na etapa de validação do modelo.

Dessa forma, o modelo treinado foi utilizado para prever os valores de uso de CPU e acionar, proativamente, a criação de novos Pods UPF. A lógica de funcionamento da solução proativa está descrita no Algoritmo 1. Inicialmente, são carregadas as informações atuais do cluster Kubernetes, como o número de Pods UPF em execução. Em seguida, é iniciado o monitoramento dos Pods UPF, permitindo verificar a utilização atual de CPU pelas funções de rede. Por fim, o modelo preditivo é carregado para realizar as estimativas e orientar o escalonamento.

Em seguida, é iniciado um loop de monitoramento, no qual os dados de uso de CPU são coletados de forma recorrente. Antes de serem utilizados pelo modelo preditivo, esses dados passam por um processo de normalização, etapa essencial para melhorar o desempenho da predição. Com os dados devidamente normalizados, eles são submetidos ao modelo GRU, que estima a carga futura. O valor predito é então convertido para a escala original. Caso esse valor ultrapasse o limiar definido na Tabela I, o mecanismo de escalonamento é acionado. Caso contrário, o loop de monitoramento continua em execução.

**Algorithm 1** Visão Geral do Sistema Proativo de Escalonamento com GRU

**Inicialização do Ambiente:**
  Carregar configurações do cluster Kubernetes
  Instanciar monitor para coleta de métricas de CPU
  Carregar modelo GRU treinado
**Ciclo de Monitoramento e Predição:**
**while** aplicação estiver ativa **do**
  Obter dados recentes de uso de CPU do pod
  Normalizar os dados para entrada do modelo
  Realizar predição da carga futura com o modelo GRU
  Inverter a normalização para obter o valor real da carga prevista
  **Tomada de Decisão:**
  **if** carga prevista excede o limiar **then**
    Escalonar o número de réplicas do UPF no cluster
    Encerrar execução (ou aguardar novo ciclo)
  **else**
    Manter configuração atual do *deployment*
  **end if**
  Esperar um intervalo fixo antes da próxima coleta
**end while**

## III. RESULTADOS

Esta seção apresentará os resultados obtidos durante a execução do escalonamento dos Pods UPF utilizando a solução proposta baseada em GRU e a comparação com a abordagem reativa HPA, nativa de Kubernetes.

### A. Metodologia de Avaliação

*a) Ambiente de Teste:* Os experimentos foram realizados em ambientes baseados em Linux, utilizando a distribuição Ubuntu 22.04. Para a conteinerização e orquestração, foi empregado o Minikube com Kubernetes na versão 1.27.4, configurado para utilizar 16 CPUs e 32 GB de memória RAM, e Helm na versão 3.11.2. Esses componentes foram escolhidos por sua estabilidade e suporte às versões desejadas tanto do Kubernetes quanto da implementação da rede 5G da OpenAirInterface (OAI), garantindo a compatibilidade e robustez necessárias para a execução dos testes neste estudo.

*b) Configuração dos Cenários:* Os experimentos foram conduzidos avaliando diferentes configurações de escalonamento e carga. Foram testados três tipos de cenários: sem autoscaler, HPA nativo do Kubernetes, e um escalonador baseado em rede neural recorrente do tipo GRU. Cada cenário foi avaliado com dois diferentes números de equipamento do usuário (*User Equipments* - UEs): 5 e 10 unidades. Os testes foram realizados de forma individual para cada combinação de cenário e número de UEs, com duração de 30 minutos por teste, garantindo a coleta de dados consistente e representativa

TABELA III
CONFIGURAÇÃO DO AMBIENTE DE TESTE.

| | Componente | Especificação |
|---|---|---|
| **Infraestrutura de Virtualização** | CPU | Intel Xeon Gold 5215 |
| | RAM | 96GB |
| | *Kubernetes* | 1.27.4 |
| | *Helm* | 3.11.2 |
| | Sistema Operacional | Ubuntu 22.04 LTS |
| **Rede 5G** | Plataforma | Open Air Interface |
| | gNodeB | Simulada |
| | Usuário | Simulado $N = 5, 10$ |
| | Implantação | Básica |

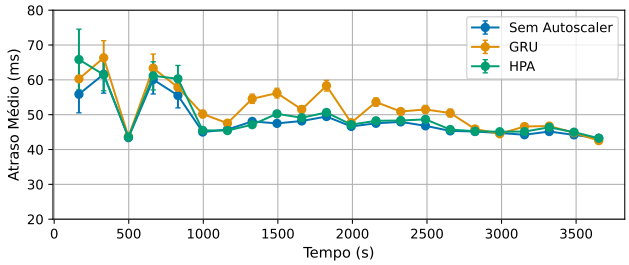


Fig. 3. Gráfico da latência em função do tempo para 5 UEs

do desempenho de cada cenário. Scripts de automação e configuração foram utilizados para provisionar as UEs de forma ágil e padronizada no ambiente de testes.

*c) Coleta de Dados:* A coleta das métricas foi realizada utilizando a Metrics Server API do Kubernetes para monitorar recursos como CPU e memória consumidos pelos pods. Além disso, foram implementados módulos personalizados para a captura de latência, fundamentais para a análise do desempenho da rede. Essa métrica customizada foi calculada diretamente na UE receptora, garantindo precisão na medição dos parâmetros de interesse. Para complementar, utilizamos um script personalizado denominado `monitor-pod.py` para a captura das métricas do UPF, com capacidade de ajuste dos intervalos de coleta, definidos em 250 ms para os testes realizados. Essa estratégia permitiu uma análise detalhada do comportamento dos diferentes escalonadores e cenários sob variadas condições de carga.

## B. Avaliação dos Mecanismos

*a) Comparação da Latência:* As figuras 3 e 4 apresentam os resultados da variação de latência ao longo do tempo para os cenários com 5 e 10 UEs, respectivamente. A análise dos gráficos de latência em função do tempo relativo mostra que, no cenário com 5 UEs, a carga do sistema é suficientemente baixa para que o uso de mecanismos de escalonamento não proporcione benefícios claros. As curvas de latência mantêm-se próximas ao longo do tempo, com variações pequenas entre as diferentes abordagens.

Por outro lado, no cenário com 10 UEs, observa-se um comportamento distinto: entre as unidades de tempo 2500 e 3000, o mecanismo com escalonamento preditivo via GRU consegue manter a latência mais controlada e estável. Isso evidencia a eficácia do modelo preditivo em antecipar aumentos de demanda e alocar recursos de forma proativa, evitando picos de latência que são perceptíveis nos outros métodos.

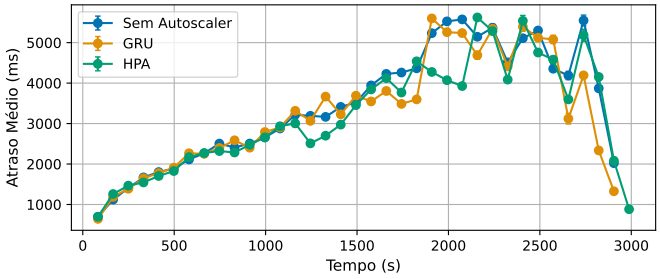


Fig. 4. Gráfico da latência em função do tempo para 10 UEs

*b) Eficiência dos Autoscalers:* Com base na análise realizada, observa-se que a eficácia dos mecanismos de escalonamento está diretamente relacionada ao nível de demanda por recursos de rede. Em momentos de pico de consumo, tanto o HPA quanto o GRU demonstram capacidade de estabilizar e controlar a latência. No entanto, soluções preditivas oferecem uma vantagem significativa, ao permitirem uma resposta mais ágil e proativa, sendo muitas vezes o fator decisivo para evitar que um pico súbito comprometa o desempenho do sistema.

## C. Consumo de Recurso Computacional

A fim de verificar o compromisso entre os impactos da latência e o consumo dos recursos computacionais, foi medido o uso dos núcleos da CPU do servidor. Os resultados obtidos foram apresentados na Tabela III-C, separados pelo método utilizado, número de usuários, e os seus respectivos valores máximo (máx.), mínimo (Mín.), média e desvio padrão (D.P.).

TABELA IV
ESTATÍSTICAS DE USO DE CPU (EM NÚCLEOS)

| Método | Núm. UEs | Máx. | Mín. | Média | D.P. |
|---|---|---|---|---|---|
| Sem autoscaler | 10 | 0.144 | 0.010 | 0.038928 | 0.043410 |
| | 10 | 0.146 | 0.010 | 0.028370 | 0.037109 |
| | 5 | 0.135 | 0.010 | 0.053574 | 0.038106 |
| | 5 | 0.140 | 0.010 | 0.042785 | 0.047225 |
| HPA | 10 | 0.067 | 0.010 | 0.021756 | 0.016999 |
| | 10 | 0.064 | 0.009 | 0.019837 | 0.016113 |
| | 5 | 0.064 | 0.009 | 0.028052 | 0.022492 |
| | 5 | 0.013 | 0.010 | 0.011236 | 0.000620 |
| GRU | 10 | 0.093 | 0.010 | 0.040762 | 0.025996 |
| | 10 | 0.092 | 0.010 | 0.027412 | 0.024324 |
| | 5 | 0.093 | 0.010 | 0.052444 | 0.031913 |
| | 5 | 0.095 | 0.010 | 0.043458 | 0.032882 |

Os resultados indicam que o uso de CPU, para o método HPA apresentou o menor consumo médio e a menor variabilidade, demonstrando maior eficiência e estabilidade, ideal para ambientes que priorizam escalabilidade e economia de recursos. O modelo GRU teve desempenho semelhante, apresentou também uma baixa variabilidade, indicando um controle mais eficiente do uso de CPU. Já o método Sem autoscaler apresentou maior variabilidade e consumo médio mais alto, sendo o menos eficiente dos três.

### *D. Lições Aprendidas*

A partir dos experimentos conduzidos, foram identificadas algumas lições relevantes para a otimização de aplicações sensíveis à latência em ambientes orquestrados com Kubernetes.

Em primeiro lugar, observou-se que em ambientes com baixa demanda por recursos ou paralelismo, o uso de mecanismos de escalonamento não necessariamente resulta em ganhos de desempenho. Nessas situações, o custo computacional e a complexidade adicionada pelo escalonamento podem não se justificar, visto que os recursos disponíveis já são suficientes para atender à carga com estabilidade.

Por outro lado, em cenários de maior demanda, mecanismos preditivos como o GRU se mostraram mais eficazes do que abordagens reativas como o HPA, especialmente nos momentos iniciais de variações abruptas de carga, nos quais a agilidade da resposta é crucial para manter a latência sob controle.

Outro ponto essencial identificado foi a importância da forma como a aplicação é desenvolvida e instrumentada para expor métricas relevantes. Aplicações que fornecem métricas detalhadas e consistentes permitem a construção de estratégias de escalonamento mais inteligentes, com maior capacidade de adaptação e eficiência.

Além disso, a frequência de coleta de métricas demonstrou ter impacto direto na sensibilidade e precisão do mecanismo de resposta. O uso de ferramentas customizadas, como o `monitor-pod.py`, permitiu configurar intervalos de amostragem mais curtos (como 250 ms), possibilitando uma visualização mais refinada do comportamento da rede. Essa granularidade contribuiu para decisões mais rápidas e eficazes dos escalonadores.

Essas observações demonstram que o escalonamento eficiente depende não apenas do mecanismo adotado, mas também do perfil de carga da aplicação, da qualidade e frequência das métricas disponíveis e do comportamento dinâmico do sistema ao longo do tempo.

## IV. Conclusão

Este trabalho propôs uma arquitetura adaptativa para o escalonamento da função de rede UPF no plano de dados de redes 5G, explorando tanto abordagens reativas, com o uso do Horizontal Pod Autoscaler (HPA) do Kubernetes, quanto preditivas, com um modelo baseado em GRU. Através de testes simulados com diferentes cargas de tráfego (5 e 10 UEs), foi possível avaliar o impacto desses mecanismos sobre métricas críticas de desempenho.

Os resultados demonstraram que, em cenários com baixa demanda por paralelismo e recursos (5 UEs), o uso de escalonadores não trouxe benefícios significativos, indicando que a ativação de mecanismos de escalonamento deve considerar o nível de carga do sistema. Por outro lado, em ambientes com maior concorrência por recursos (10 UEs), observou-se que ambos os mecanismos contribuíram para a estabilidade da latência, sendo que o modelo preditivo apresentou desempenho superior nos momentos iniciais de sobrecarga, por antecipar picos e responder de forma mais ágil.

Além disso, o estudo evidenciou a importância de um gerenciamento mais refinado das funções de rede e da infraestrutura subjacente, indo além das configurações padrão do Kubernetes. Lições aprendidas incluem a necessidade de adaptar a estratégia de escalonamento à carga prevista e à natureza da aplicação, destacando que soluções preditivas podem ser determinantes para garantir qualidade de serviço em redes móveis de nova geração.

Como trabalhos futuros, pretende-se desenvolver um sistema de balanceamento de carga mais eficiente do que o nativo do Kubernetes, a fim de viabilizar testes em cenários de carga significativamente mais elevada. Além disso, buscaremos aperfeiçoar o modelo de inteligência artificial, tornando o mecanismo de escalonamento ainda mais dinâmico e responsivo às variações do ambiente.

## Agradecimentos

Este trabalho foi apoiado pelo projeto Nº 26/23 FADE/UFPE/FINEP (01.23.0528.00 -FINEP) REF. 2849/22 - (CONVÊNIO Nº 74/2023 - UFPE) 23076.100425/2023-24, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Bolsa de Produtividade em Pesquisa (Processo nº 313083/2023-1) e Fundação de Amparo à Pesquisa do Estado de Pernambuco (FACEPE) (Processo nº IBPG-0130-1.03/23).

## Referências

[1] S. Chen, Y.-C. Liang, S. Sun, S. Kang, W. Cheng e M. Peng, "Vision, Requirements, and Technology Trend of 6G: How to Tackle the Challenges of System Coverage, Capacity, User Data-Rate and Movement Speed,"arXiv preprint arXiv:2002.04929, fev. 2020.

[2] M. Bennis, M. Debbah e H. V. Poor, "Ultra-Reliable and Low-Latency Wireless Communication: Tail, Risk and Scale,"arXiv preprint arXiv:1801.01270, jan. 2018.

[3] R. Li, B. Decocq, A. Barros, Y. -P. Fang and Z. Zeng, "Estimating 5G Network Service Resilience Against Short Timescale Traffic Variation,"in IEEE Transactions on Network and Service Management, vol. 20, no. 3, pp. 2230-2243, Sept. 2023, doi: 10.1109/TNSM.2023.3269673.

[4] T. V. Kiran Buyakar, H. Agarwal, B. R. Tamma and A. A. Franklin, "Prototyping and Load Balancing the Service Based Architecture of 5G Core Using NFV,"2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 2019, pp. 228-232, doi: 10.1109/NETSOFT.2019.8806648.

[5] T. Van Do, N. H. Do, C. Rotter, T. V. Lakshman, C. Biro and T. Bérczes, "Properties of Horizontal Pod Autoscaling Algorithms and Application for Scaling Cloud-Native Network Functions,"in IEEE Transactions on Network and Service Management, vol. 22, no. 2, pp. 1889-1898, April 2025, doi: 10.1109/TNSM.2025.3532121.

[6] J. E. Joyce and S. Sebastian, "Enhancing Kubernetes Auto-Scaling: Leveraging Metrics for Improved Workload Performance,"2023 Global Conference on Information Technologies and Communications (GCITC), Bangalore, India, 2023, pp. 1-7, doi: 10.1109/GCITC60406.2023.10426170.

[7] Jon Larrea, Andrew E. Ferguson, and Mahesh K. Marina. 2023. CoreKube: An Efficient, Autoscaling and Resilient Mobile Core System. In Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23). Association for Computing Machinery, New York, NY, USA, Article 25, 1–15. https://doi.org/10.1145/3570361.3592522

[8] H. T. Nguyen, T. Van Do and C. Rotter, "Scaling UPF Instances in 5G/6G Core With Deep Reinforcement Learning,"in IEEE Access, vol. 9, pp. 165892-165906, 2021, doi: 10.1109/ACCESS.2021.3135315.