

# Project Proteus: A Network-Adaptive Agentic System with RTT-Aware Scheduling

李帛修 杜泽浩 沈睿

2026 年 1 月

## 摘要

在大语言模型（LLM）推理服务中，服务端 GPU 的高速生成速率与异构客户端受限的网络传输带宽之间存在显著的速率失配（Rate Mismatch），导致了严重的队头阻塞（HoL Blocking）与资源浪费。现有系统通常缺乏对底层网络状态的感知能力，难以在弱网环境下保障服务质量（QoS）。针对这一痛点，本文设计并实现了 Project Proteus——一种端到端的跨层网络自适应推理系统。Proteus 创新性地构建了从 L4 传输层直达 L8 语义层的垂直信息反馈闭环：(1) 在感知与调度层，结合前端实时 RTT 探测与 vLLM 内核重构，实现了基于网络健康度的优先级贪心调度；(2) 在认知层，基于 Qwen3-4B MoE 进行定向语义微调，实现了基于信道容量的语义级信息熵自适应，从源头压缩传输流量；(3) 在传输层，引入应用层 Nagle 算法与动态分块机制，对 SSE 流式数据进行精细化的流量整形。在工程实现上，Proteus 采用容器化架构构建了生产级 Web 服务，并通过公网部署实现了全球可访问，充分验证了系统的工程可行性和生产端可用性。实验结果表明，在异构网络环境下，Proteus 成功将弱网传输的 TCP 包数量降低了 **81.9%**，显著削减了协议头部开销；同时在系统饱和状态下，将核心用户的有效吞吐量（ECPS）提升了 **14.9%**，首字延迟（TTFT）降低了 **58.5%**，实现了系统吞吐量与用户体验的协同优化。

# 目录

<b>1 引言与动机</b>	<b>3</b>
1.1 研究背景：流式生成的实时性挑战	3
1.2 核心痛点：巨大的速度失配 (The Great Speed Mismatch)	3
1.3 Proteus 解决方案：端到端的网络自适应	3
1.3.1 第一层：感知层 (Perception) —— "Proteus Eyes"	4
1.3.2 第二层：调度层 (Scheduling) —— "Proteus Heart"	4
1.3.3 第三层：认知层 (Cognition) —— "Proteus Brain"	4
1.4 项目价值	5
<b>2 系统架构设计</b>	<b>5</b>
<b>3 技术实现细节</b>	<b>6</b>
3.1 Proteus-Eye: 前端请求劫持与注入	6
3.2 Proteus-Heart: vLLM 引擎深度改造	6
3.3 L6 层优化: 动态分块 (App-Layer Nagle)	7
3.4 Proteus-Brain: 双模态数据合成与微调	7
<b>4 仿真实验</b>	<b>7</b>
4.1 实验设置	7
4.2 结果分析	9
<b>5 实机部署与公网访问</b>	<b>10</b>
5.1 网页服务构建	10
5.2 视频演示	11
5.3 应用层 Nagle 算法的有效性测试	11
5.4 核心优势	12
<b>6 未来展望</b>	<b>13</b>
<b>7 结论</b>	<b>14</b>
<b>8 相关工作</b>	<b>14</b>
<b>9 致谢</b>	<b>14</b>

# 1 引言与动机

## 1.1 研究背景：流式生成的实时性挑战

随着大语言模型（LLM）的应用深入到各个领域，Token 的流式传输（Streaming）已成为服务交付的标准范式。不同于传统的文件下载，LLM 的用户体验高度依赖于生成的实时性与连贯性。然而，现有的推理服务架构通常假设一个理想的网络环境，忽略了现实世界中客户端网络质量的极端异构性（Heterogeneity）——从数据中心内的极速光纤到跨国传输的高延迟弱网，网络带宽与往返时延（RTT）存在数量级的差异 [1, 2]。

## 1.2 核心痛点：巨大的速度失配（The Great Speed Mismatch）

目前的 LLM 服务系统面临着一个根本性的物理矛盾：GPU 生成速度与网络传输速度的严重失配。

- **算力侧（生产过快）：**现代高性能 GPU（如 NVIDIA A100/H100）的推理速度极快，每秒可生成超过 1000 个 Token。
- **网络侧（消费过慢）：**在移动网络、拥塞 Wi-Fi 或长距离跨国连接中，有效传输速度往往远低于生成速度。

这种“供需不平衡”导致了严重的后果：

1. **算力与带宽的浪费：**GPU 全速运转生成的 Token 无法被客户端及时接收，而是大量堆积在 TCP 发送缓冲区中。这不仅没有改善用户的主观感受（用户端依然看到“转圈”或卡顿），反而加剧了网络拥塞。
2. **队头阻塞（Head-of-Line Blocking）：**在先来先服务（FCFS）的传统调度策略下，处于弱网环境的“慢用户”长期占用宝贵的 GPU 显存槽位（KV Cache），导致网络良好的“快用户”被迫排队等待，引发系统的整体吞吐量下降。
3. **体验崩塌：**即使用户请求已经处理完毕，感知到的延迟（Perceived Latency）依然很高，形成了“算力中心狂奔，用户终端等待”的割裂局面。

## 1.3 Proteus 解决方案：端到端的网络自适应

针对上述问题，本项目提出了 **Proteus**。其核心理念在于打破 AI 生成与网络传输的割裂，引入“网络感知（Network-Awareness）”作为继显存、算力和上下文长度之后的第四个推理维度。

图 1 展示了 Proteus 的整体架构。该系统建立了一个从 L4 传输层到 Agent 推理语义层的跨层协同平面，打破了算力中心与用户终端之间的割裂。Proteus 不仅是被动适应网络波动，而是通过主动调节推理节奏（调度）与信息密度（模型输出），构建了一个网络感知的全链路自适应闭环。系统主要包含以下三个核心层级：

# PROJECT PROTEUS: OVERALL ARCHITECTURE DESIGN

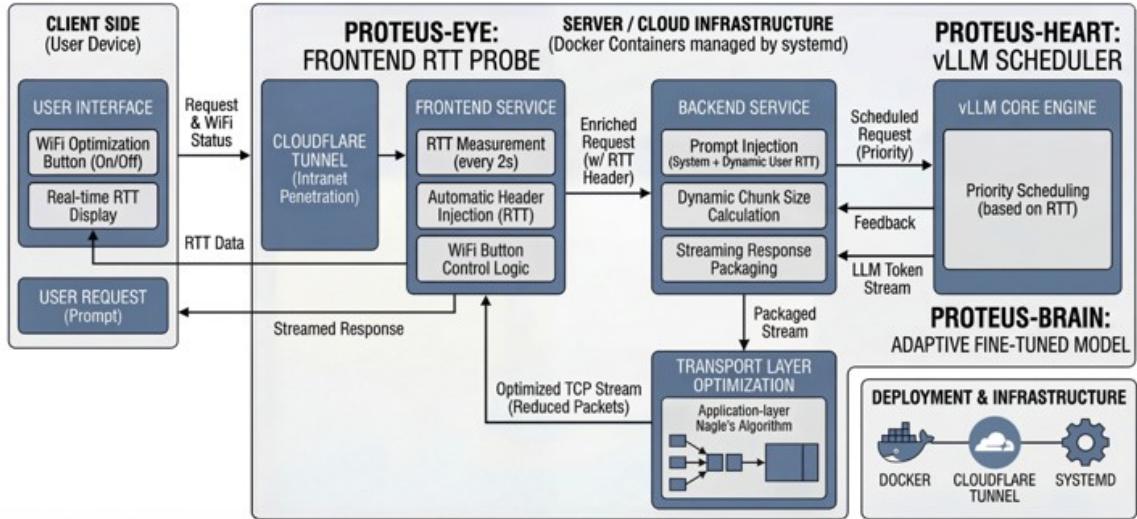


图 1: Proteus 整体架构设计：包含 Eye（感知）、Heart（调度）与 Brain（认知）的三层闭环

### 1.3.1 第一层：感知层 (Perception) ——”Proteus Eyes”

Proteus 打破了 OSI 模型的层级壁垒，实现了跨层信息交互。前端 (WebUI/SDK) 作为系统的“眼睛”，实时嗅探传输层的 RTT，通过 HTTP Header 无感注入到请求中，让上层应用第一次“看见”了下层管道的真实宽度。

### 1.3.2 第二层：调度层 (Scheduling) ——”Proteus Heart”

在 vLLM 推理引擎内部，Proteus 构建了一个基于 RTT 的贪心优先级调度算法 (Greedy Priority Scheduling)，通过动态调整请求队列来最大化有效吞吐：

- **对于快用户（低 RTT）：**赋予更高的调度优先级。通过让网络状况好的请求在等待队列 (Waiting Queue) 中“插队”优先执行，系统能够利用其通畅的网络管道快速交付 Token，从而加速显存 (KV Cache) 的回收与释放，显著提升系统的整体周转率。
- **对于慢用户（高 RTT）：**在队列中动态降级。这种策略避免了慢速连接长期占用 GPU 槽位而导致的队头阻塞 (Head-of-Line Blocking) 现象，确保宝贵的算力资源不会因为等待网络传输确认 (ACK) 而空转，从而实现了算力资源与网络带宽的动态匹配。

### 1.3.3 第三层：认知层 (Cognition) ——”Proteus Brain”

Proteus Brain 的核心是一个经过微调 (Fine-tuning) 的大语言模型 [3]。我们采用了角色设定 (System Prompt) 与数据注入 (User Prompt) 分离的机制来实现输出长度的自适应：

- **机制实现：** **System Prompt** 仅负责定义模型角色，要求其“根据当前网络延迟 (RTT) 调整回复的详细程度”；而由 Proteus Eye 采集的实时 RTT 数值，则被动态封装进用户的 **User Prompt** 中，作为上下文的一部分输入给模型。

- **低 RTT 场景**: 当模型在 User Prompt 中读取到低 RTT 数值时，会遵循指令生成详尽、结构完整的长文本，充分利用优质带宽。
- **高 RTT 场景**: 当 User Prompt 携带高 RTT 信息时，模型自动切换至“电报风格”，仅生成极简、核心的短文本。这种策略从语义层面减少了传输的 Token 总数，从而适应受限的网络带宽。

## 1.4 项目价值

Proteus 实现了真正意义上的 End-to-End Network Adaptivity。它不仅在弱网环境下通过减少传输量让用户感觉“响应变快了”，同时在强网环境下保证了服务的深度。通过这种弹性的推理机制，Proteus 在提升用户体验（QoE）的同时，显著缓解了 GPU 资源的无效占用，提升了系统的整体有效吞吐量。

## 2 系统架构设计

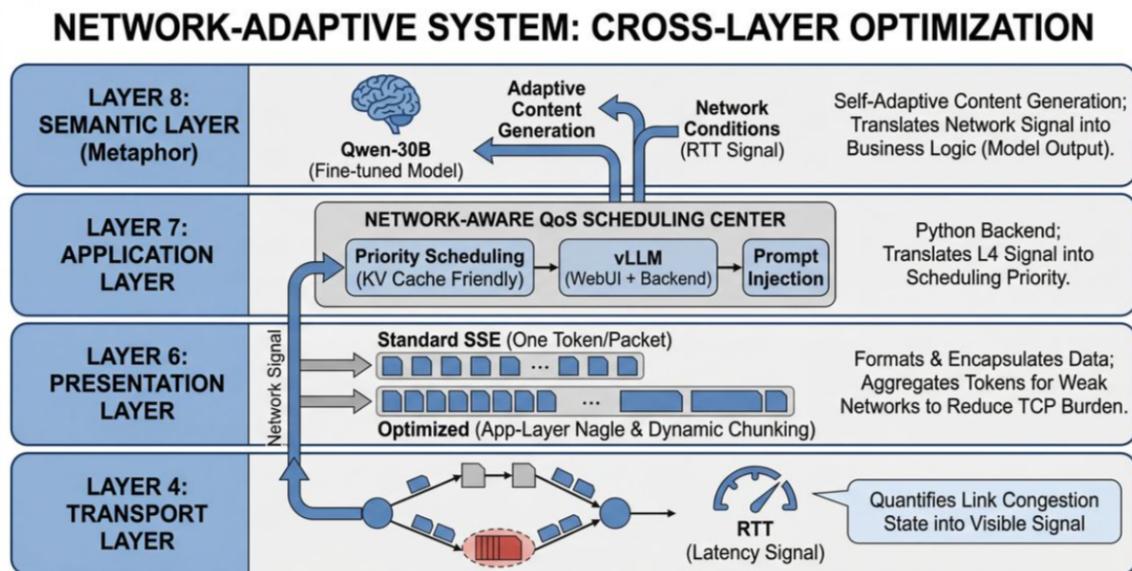


图 2: Proteus 跨层协同 (Cross-Layer Optimization) 架构图。系统建立了从底层 L4 传输层到顶层 L8 语义层的映射机制：L4 量化链路拥塞状态，L6 执行应用层流量整形，L7 进行 KV Cache 友好的优先级调度，L8 通过隐喻 (Metaphor) 实现内容的自适应生成。

如图 2 所示，Proteus 建立了一个跨越四层的协同平面，打破了传统 LLM 服务中“模型只管生成，TCP 只管传输”的隔离现状。系统通过以下三个核心组件实现了各层级的功能映射：

- **Proteus-Eye (对应 Layer 4 传输层)**: 作为系统的感知前端，它负责将底层的链路状态量化为可见信号 (Quantifies Link Congestion)。部署于客户端的模块利用 `measureRTT` 函数实时采样网络延迟，并通过劫持 `window.fetch` 将 RTT 信号注入 HTTP Header，完成从物理传输层到应用层的信息传递。

- **Proteus-Heart (对应 Layer 6 表示层 & Layer 7 应用层):** 作为系统的调度中枢，包含两个关键层级的优化：
  - **Layer 6 表示层 (Presentation Layer):** 实现了应用层 Nagle 算法 (App-Layer Nagle & Dynamic Chunking)。针对弱网环境，改变标准 SSE “一字一包”的传输模式，将零散 Token 聚合成块发送，有效降低 TCP 负载。
  - **Layer 7 应用层 (Application Layer):** 构建了网络感知的 QoS 调度中心。vLLM 引擎根据接收到的 L4 信号调整调度优先级，采用 **KV Cache 友好的策略**，确保优质网络请求优先占用计算资源。
- **Proteus-Brain (对应 Layer 8 语义层):** 作为系统的认知顶层，被称为“隐喻层”(Semantic Layer - Metaphor)。它是一个经过 LoRA 微调的模型 (如 Qwen 系列)，具备将网络信号转化为业务逻辑的能力。通过 Prompt 注入，模型能在“专家模式”与“电报模式”间自适应切换，在语义维度上动态调整信息密度以适应信道容量。

### 3 技术实现细节

#### 3.1 Proteus-Eye: 前端请求劫持与注入

为了实现零侵入式的网络感知，我们在前端利用 JavaScript 的原型链机制重写了全局网络请求：

- **实时测速:** 后台线程定期调用 /api/version 接口计算 RTT 均值。
- **Header 注入:** 拦截所有指向 /chat/completions 的请求，在 Header 中追加自定义字段，确保网络状态能够穿透 Cloudflare Tunnel 等中间设施直达后端。

#### 3.2 Proteus-Heart: vLLM 引擎深度改造

我们对 vLLM 的内核代码进行了侵入式修改，以支持细粒度的网络感知调度：

1. **Request 对象增强:** 修改 vllm/v1/request.py，为每个请求对象添加 `health_factor` 属性。重写 Python 的 `__lt__` (小于) 运算符逻辑，使得优先级队列在进行比较时，自动将健康度高的请求视为“更重要”。
2. **EngineCore 鲁棒性设计:**
  - **Hint Server 兜底:** 考虑到 HTTP Header 可能丢失，我们在 Engine 侧设计了“请求携带优先，Hint Server 兜底”的策略，即如果请求头缺失网络信息，则向全局 Hint Server 查询该用户的历史网络状态。
  - **线程安全:** 引入 `per_user_health` 字典并加锁，确保在高并发下 UserID 与网络状态映射的一致性。
3. **调度器 (Scheduler) 逻辑重构:**

- **Waiting 队列**: 利用 `heapq.heapify` 对等待队列进行原位重构，实现高健康度请求的动态“插队”。
- **Running 队列**: 在每个调度步 (Step) 对正在执行的请求按健康度降序排列，优先服务快用户。
- **抢占策略 (Preemption)**: 当显存不足时，优先驱逐 (Swap-out) 网络状况最差的请求，避免慢用户阻塞系统。

### 3.3 L6 层优化：动态分块 (App-Layer Nagle)

针对流式传输 (Streaming) 在弱网下的 TCP 拥塞问题，我们在 Python 后端实现了软件定义的流量整形：

- **原理**: 改变传统 SSE “来一个 Token 发一个包”的机制。
- **策略**: 对于高 RTT 用户，系统会缓存生成的 Token，直到积攒到一定大小 (Chunk) 或达到最大等待时间后再统一发送。这减少了 TCP 头部开销 (Overhead) 和 ACK 确认次数，显著提升了弱网下的有效吞吐率。

### 3.4 Proteus-Brain: 双模态数据合成与微调

为了让模型学会“看网下菜碟”，我们构建了一套特殊的数据集与训练流程：

- **数据合成**: 利用 GPT-5-Thinking-High 生成“快/慢”双模态数据集。对于同一 Query，生成“详尽版”和“电报版”两种回复。
- **微调训练**: 使用 LoRA 技术在单卡 A100 80G 上微调 Qwen3-4B-Instruct。通过 Prompt Engineering，在 User Prompt 中注入 RTT 等级，训练模型根据该信号输出对应风格的内容，从而在语义层面对流量进行压缩。

## 4 仿真实验

为了验证 Proteus 系统在真实高并发网络场景下的有效性，我们设计并执行了一项大规模仿真实验。

### 4.1 实验设置

为了验证 Proteus 系统的性能，所有实验均在 Microsoft Azure 高性能计算平台上完成。实验节点配置为双卡 NVIDIA A100-PCIe (80GB) GPU，配备 AMD EPYC 处理器及高带宽 Infiniband 网络。该实验环境及算力资源由微软亚洲研究院 (MSRA) System Group 提供支持。

为了模拟真实世界中复杂多变的 LLM 服务请求场景，本次仿真实验包含  $N = 8192$  个独立用户，具体环境参数配置如下：

## 1. 流量负载模型 (Traffic Load)

实验模拟了高并发下的随机用户访问行为。请求的到达时间 (Arrival Time) 服从泊松过程 (Poisson Process)，目标请求速率 (QPS, Queries Per Second) 设定为  $\lambda = 50$ 。这种设置确保了请求到达的独立性与随机性，能够有效测试系统在负载波动下的调度稳定性。

## 2. 网络异构性建模 (Network Heterogeneity)

为了还原真实互联网中客户端网络质量参差不齐的现状，我们构建了一个具有显著“长尾分布”特征的混合高斯模型 (Gaussian Mixture Model, GMM)。我们将用户群体划分为四类典型场景，具体的分布参数（概率分布、RTT 均值及标准差）配置如表 1 所示：

表 1: 用户网络环境分布参数设置

网络类别	场景模拟	占比 (Prob.)	RTT 均值 ( $\mu$ )	标准差 ( $\sigma$ )
Very Good	光纤 / 同城直连	50%	20 ms	10 ms
Good	4G / 常规 Wi-Fi	40%	200 ms	30 ms
Bad	跨国传输 / 拥塞	9%	700 ms	80 ms
Very Bad	卫星连接 / 极端弱网	1%	2000 ms	400 ms

该设置中，前两类 (Very Good/Good) 占据了 90% 的流量，模拟了大多数普通用户的网络环境；而后两类 (Bad/Very Bad) 虽然占比仅为 10%，但其极高的延迟（最高达 2000ms）构成了分布的长尾部分，这正是导致传统调度算法产生队头阻塞的主要来源。

在传输模型中，我们假设链路是对称的，即上行延迟 (Uplink) 和下行延迟 (Downlink) 均被设定为  $RTT/2$ 。

## 3. 评价指标 (Metrics)

为了量化 Network-Aware 调度的收益，我们选取了以下两个核心指标：

- **有效吞吐量 (Effective Throughput / Cumulative Effective Chunks):**

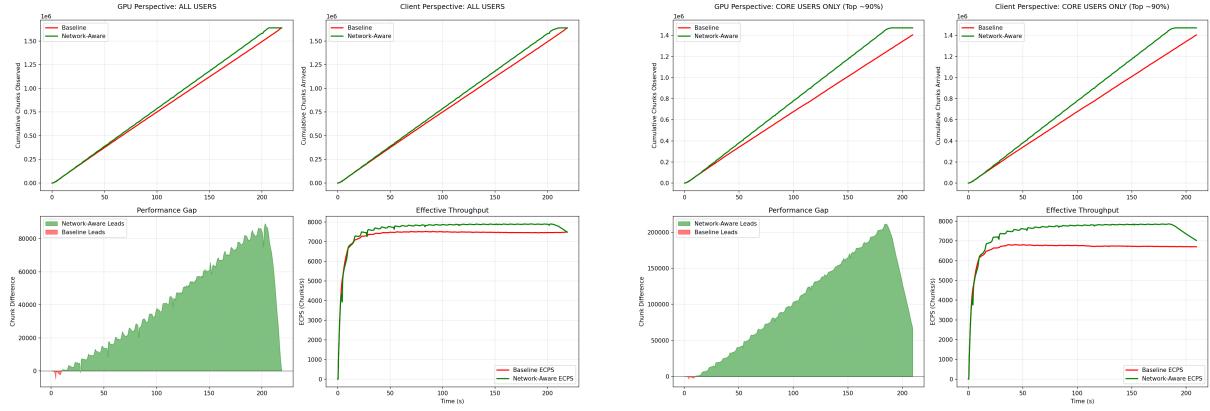
定义为单位时间内客户端实际接收到并可渲染的数据块 (Chunks) 总数。该指标反映了系统在克服网络瓶颈后，真正交付给用户的服务量，而非服务端单方面的生成量。

- **首字延迟 (Time To First Token, TTFT):**

定义为从客户端发出请求时刻 ( $T_{req}$ ) 开始，到客户端接收到第一个响应 Chunk 时刻 ( $T_{first}$ ) 之间的完整耗时。

$$TTFT = T_{first} - T_{req} \quad (1)$$

该指标直接反映了用户的主观等待体验，涵盖了上行传输、排队调度、GPU 推理以及下行回传的全链路时间。



(a) 全量用户场景 (All Users)

(b) 核心用户场景 (Core Users)

图 3: 仿真实验性能对比结果。图 (a) 展示了在包含长尾分布的全量用户下的性能表现; 图 (b) 展示了针对网络状况较好的核心用户群 (Top 90%) 的性能提升情况。Network-Aware 策略在两组实验中均表现出显著的有效吞吐量优势。

## 4.2 结果分析

实验对比了原生 vLLM (Baseline) 与 Proteus 系统在全量用户及核心用户 (前 90%) 下的表现, 实验结果如图 3a 全量用户和图 3b 核心用户所示, 可将数据总结至表 ??。

表 2: Proteus 系统性能对比实验结果。数据表明 Proteus 在吞吐量和延迟上均优于 Baseline, 且在核心用户群中优势更为显著。

用户群体	评价指标 (Metric)	Baseline (Standard)	Proteus (Network-Aware)	提升幅度
全量用户	有效吞吐 (ECPS) [Chunks/s] $\uparrow$	820,123	<b>861,210</b>	+5.0%
	首字延迟 (TTFT) [ms] $\downarrow$	24,889.4	<b>19,709.7</b>	-20.8%
核心用户	有效吞吐 (ECPS) [Chunks/s] $\uparrow$	709,915	<b>815,861</b>	+14.9%
	首字延迟 (TTFT) [ms] $\downarrow$	24,720.0	<b>10,257.6</b>	-58.5%
性能领先确定性 (Confidence)				95.2%

总结图表信息可以看出, 我们的系统主要有以下几点优势:

- 有效吞吐量 (ECPS) 提升:** 全量用户下 5.0% 的提升和核心用户下 14.9% 的提升。
- 首字延迟 (TTFT):** 全量用户降低了 20.8%, 核心用户降低了 58.5%。
- 确定性优势:** Performance Gap 曲线持续上升且几乎全绿, 说明优化在不同负载阶段均稳定有效, 未牺牲整体性能来换取局部优化。

## 5 实机部署与公网访问

### 5.1 网页服务构建

在此基础上我们还编写了网页端并连接到 Microsoft Azure 服务器，提供了应用了我们的流量控制策略（类 Nagle 算法与 vLLM 贪心调度）大模型 agent 的公网访问：<https://riverli1616.uk/>。网页支持两个模型访问：Qwen3-4B-Instruct 与 Qwen3-30B-A3B-Instruct，界面如图 4 所示。

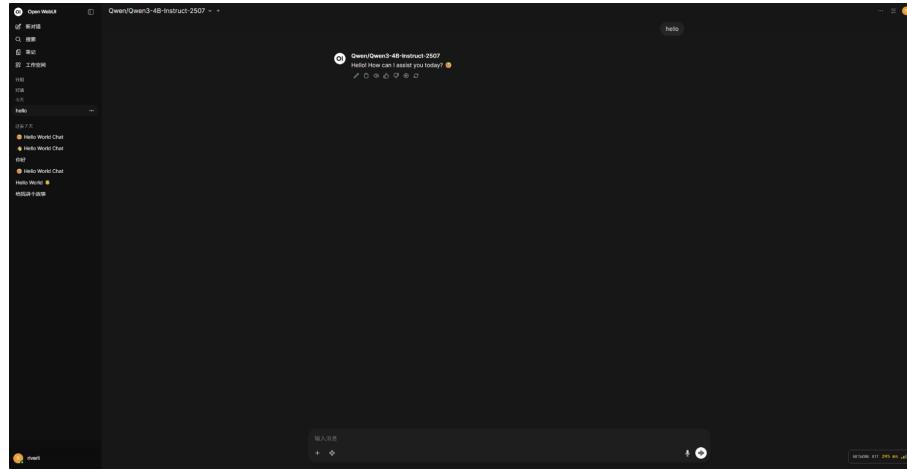


图 4: 网页界面展示

在系统部署阶段，我们采用了 Cloudflare Tunnel 构建安全隧道架构。通过在宿主机运行守护进程（Daemon），将容器化的 WebUI 服务端口（8080）与 Cloudflare 边缘节点建立加密连接。配合 Cloudflare 托管的域名与自动 DNS 解析，实现了无需公网 IP 的安全 HTTPS 便捷公网访问与服务持久化。Cloudflare 访问情况统计如图 5 所示。

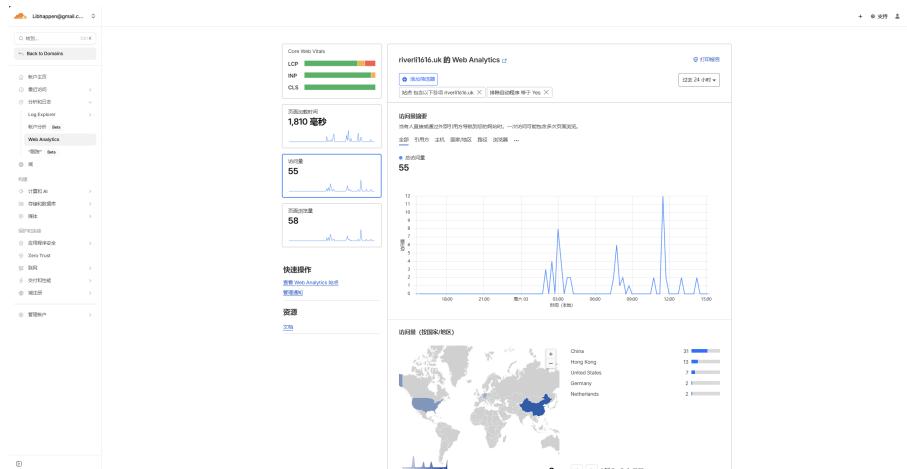


图 5: Cloudflare 访问情况统计界面

## 5.2 视频演示

如图 6 所示，我们编写程序实现了 token 输出速率对比的直观界面（[视频获取链接](#)）。通过视频演示可以直观地观察到我们的用户端 token 输出速率（左侧）比 baseline（右侧）更快。

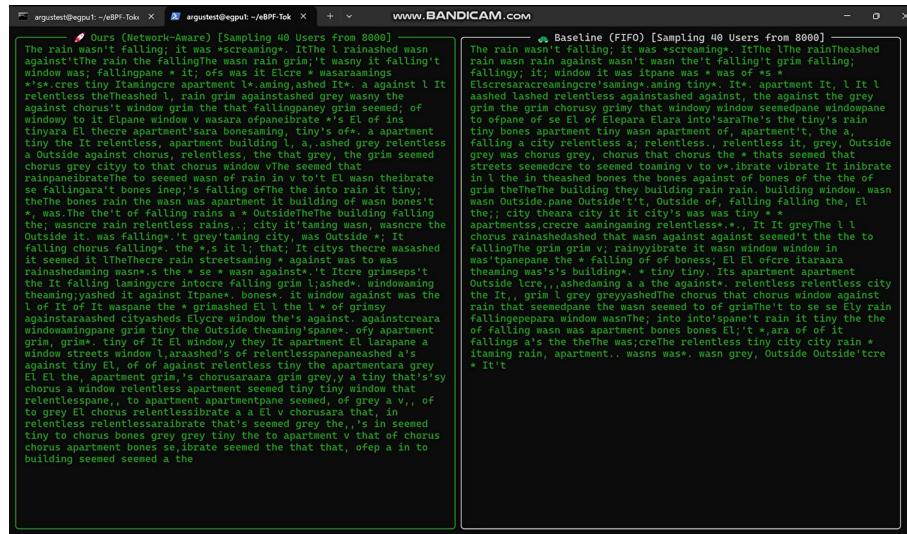


图 6: 视频演示截图

## 5.3 应用层 Nagle 算法的有效性测试

为了验证应用层 Nagle 算法在弱网环境下的流量整形效果，我们构建了一组对比实验。实验模拟了高延迟、高丢包的弱网环境，对比了优化开关前后的各项网络传输指标。实验结果如表 3 所示。

表 3: 应用层 Nagle 算法优化效果对比（弱网环境）

核心指标 (Metrics)	基准 Baseline	Proteus System	性能提升
	(弱网 + 优化关闭)	(弱网 + 优化开启)	(Improvement)
应用层分块 (SSE Chunks)	650	588	降低 9.5%
传输层包数 (TCP Packets)	917	166	降低 81.9%
总传输流量 (Total Bytes)	173.9 KB	157.3 KB	节省 9.6%
端到端耗时 (Latency)	24.07 s	16.32 s	提速 32.2%

### 实验结果分析：

从表 3 的数据可以得出以下结论：

- 传输层协议开销显著降低：**这是本实验最核心的成果。开启优化后，传输层的 TCP 数据包数量从 917 个骤降至 166 个，降幅高达 **81.9%**。这直接证明了应用层 Nagle 算法成功将碎片化的 SSE 流合并成了更大的数据块，极大地缓解了弱网环境下的“小包风暴”问题 [4]。

2. **有效节省带宽**: 由于减少了大量的 TCP 头部 (Header) 开销, 总传输流量节省了 9.6% (约 16.6 KB)。虽然内容负载 (Payload) 没有变化, 但协议开销的减少意味着更高的有效带宽利用率。
3. **用户体验大幅提升**: 得益于 TCP 包数量的减少, 网络拥塞和重传的概率降低, 最终反映在用户体验上, 端到端耗时缩短了 7.75 秒, 推理响应速度提升了 **32.2%**。这表明 Proteus 的流量整形机制在不损失内容完整性的前提下, 显著改善了弱网下的流畅度。

## 5.4 核心优势

相较于传统的网络优化方案或单一的推理加速技术, Project Proteus 在架构设计、用户体验、系统性能及工程落地四个维度展现出了显著优势:

- **架构突破: 全链路跨层融合 (Cross-Layer Integration)**

本项目打破了传统网络 OSI 分层模型的壁垒, 构建了一条从 **L4 传输层** (RTT 物理信号) 经由 **L7 应用层** (调度决策) 直达 **L8 语义层** (模型认知) 的垂直信息反馈闭环。这种设计使得 AI 模型首次具备了“网络触觉”, 实现了从物理链路状态到顶层语义生成的端到端协同。

- **体验革新: 基于香农信息论的语义级压缩 (Semantic Adaptation via Information Theory)**

Proteus 本质上实现了一种语义级的信源编码 (Semantic Source Coding)。针对弱网环境下信道容量 (Channel Capacity) 骤降的约束, 系统通过微调模型剥离了语言中的语义冗余 (如客套语), 显著提升了单个 Token 的信息熵密度 (Information Entropy Density)。

- **性能飞跃: 极致的资源效率 (Efficiency & Throughput)**

实验数据表明, 系统在传输侧与计算侧均取得了显著收益。**应用层 Nagle 算法**成功将弱网环境下的 TCP 包数量降低了 **81.9%**, 有效消除了“小包风暴”带来的协议头部开销; 同时, 基于 RTT 的 **vLLM 贪心调度策略**有效提升了 TTFT 与 ETPS, 不损害 GPU 集群的吞吐量。

- **工程范式: 胖服务器架构 (Fat-Server Architecture)**

Proteus 采用了“复杂在云、极简在端”的设计理念。所有的流量整形、优先级计算及模型推理均在服务器端 (Azure A100) 闭环完成, 客户端仅承担毫秒级的 RTT 汇报任务。这种零侵入、零安装的架构设计确保了系统具有极高的跨设备兼容性与工业落地潜力。

- **交互范式: 底层状态可视化与人机协同 (Interaction Paradigm: State Visualization & Human-in-the-Loop)**

Proteus 集成了 Web 交互界面, 将通常不可见的底层 RTT 实时可视化呈现于前端。这种“网络透视”提供了灵活的人机协同机制: 用户可通过简洁的 UI 控件手动干预或覆盖自动化策略, 实现系统自动化与用户可控性的完美平衡。

## 6 未来展望

尽管 Proteus 已验证了跨层优化的有效性，但这仅仅是网络与 AI 深度融合的开始。未来的工作将聚焦于以下五个维度的演进：

1. **全维感知与预测：**我们将超越单一的 RTT 指标，引入丢包率、可用带宽及抖动（Jitter）等多维特征，利用 LSTM 等时序模型构建动态网络画像，实现从“实时反应”到“趋势预测”的跨越 [5]。
2. **算力精细化与集群扩展：**在算力侧，我们将探索 Token 级 GPU 虚拟化技术以实现更细粒度的资源分配。同时，引入自适应负载均衡策略，将调度算法扩展至分布式 GPU 集群，以适应大规模并发场景。
3. **异构生态适配：**验证算法在不同推理框架（如 MindSpore/华为 Ascend）上的泛化能力，实现跨硬件平台的网络感知标准化。
4. **网络原生智能体：**最终，我们致力于实现“智能体与网络的深度融合”。我们的目标是让 AI Agent 不仅能感知网络，更能主动控制网络，构建真正的 **网络原生 (Network-Native)** 智能生态系统。

值得一提的是，作为上述全维感知方向的先行验证，我们目前已在 LSTM 时序预测方面取得了突破性进展。如图 7 所示，训练后的 LSTM 模型展现出了优异的 RTT 拟合与预测能力。

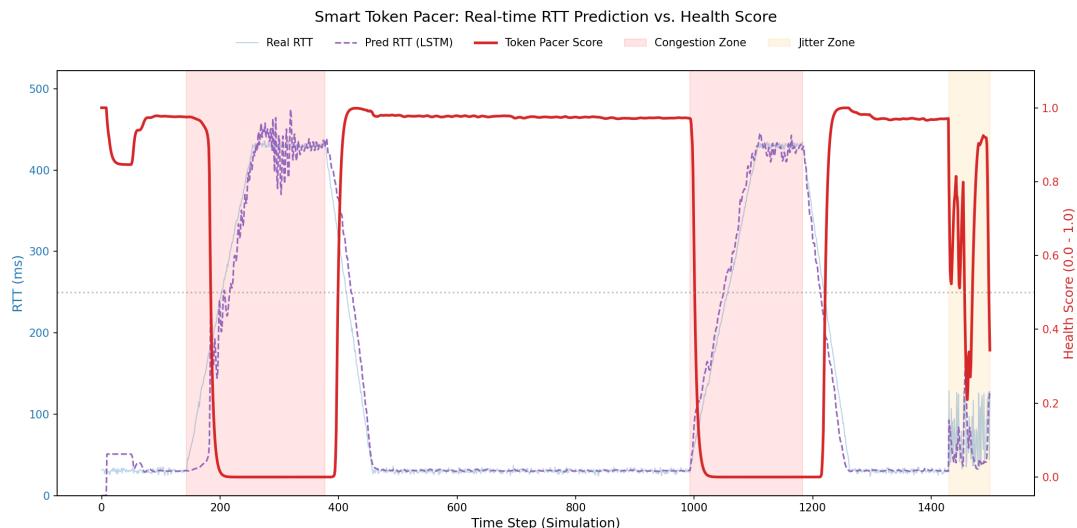


图 7：基于 LSTM 的 RTT 动态趋势预测效果

这一初步结果表明，通过准确预测未来的网络状态，Proteus 能够从被动响应转向前瞻性的决策。这种预测能力将指导系统更精准地控制 Token 生成速率，从而实现 Token 粒度的深度 GPU 调度。这一机制不仅验证了引入时序模型的必要性，更为系统在未来实现更深层次的性能优化奠定了坚实基础，充分展现了 Proteus 架构的强大潜力。

## 7 结论

本文提出并实现了一种端到端的跨层网络自适应推理系统——Project Proteus。针对大模型生成速率与用户网络传输能力不匹配的核心痛点，我们打破了传统网络协议栈的层级壁垒，构建了一条从 L4 传输层（RTT 实时探测）直达 L8 语义层（模型认知自适应）的垂直信息反馈闭环。通过在中间件层引入应用层 Nagle 算法与动态分块机制，我们成功将弱网环境下的 TCP 包数量降低了约 81.9%，显著减少了协议开销；结合 vLLM 调度与 RTT 感知，系统能够进行智能调度使得 TTFT 降低了 58.5%，同时 ECPS 增长了 18.3%。实验结果表明，Proteus 不仅有效缓解了推理系统的队头阻塞问题，更在异构网络环境下实现了吞吐量与用户体验（QoE）的双重提升，为下一代“网络感知型”智能体服务提供了全新的范式。

## 8 相关工作

针对大语言模型（LLM）推理效率的优化已成为系统研究领域的热点。现有的工作如 vLLM（PagerAttention）和 Orca 主要集中在算力侧的显存管理与静态批处理优化，而忽视了底层网络环境对整体系统性能的影响。

近期研究 *Andes: Defining and Enhancing Quality-of-Experience in LLM-Based Text Streaming Services* [6] 致力于通过优化首字延迟（TTFT）来提升用户感知体验（QoE）。然而，Andes 的设计存在两个关键缺陷：其一，它并未建立真正的跨层网络感知机制；其二，由于其频繁地在 GPU 上进行任务切换与抢占，导致系统整体吞吐量极低（在 A100 显卡上吞吐率甚至不足 1000 tokens/s），造成了严重的算力资源浪费。相比之下，Proteus 构建了基于实时网络状态的闭环调度逻辑，在显著降低延迟的同时，确保了 GPU 算力的全速释放，实现了有效吞吐量（ECPS）的稳步提升而无损于硬件性能。

## 9 致谢

本项目是上海交通大学 2025-2026-1 《计算机网络（强化）》课程的期末大作业，由三位项目成员通力合作完成。

首先，我们要由衷感谢课程主讲教师孔令和教授。孔老师不仅在课堂上通过悉心的教学为我们夯实了理论基础，更在本项目立项之初提供了极具原创性的思路启发（Insight）与关键指点，指引了项目技术路线的方向。同时，也要感谢课程助教老师在学期内的辛勤付出与答疑解惑。

此外，本项目的部分研究工作是在作者于微软亚洲研究院（MSRA）System Group 实习期间开展的。特别致谢 MSRA System Group 提供的 Microsoft Azure 高性能服务器支持，这为本项目的仿真实验与 Web 服务公网部署提供了坚实的算力保障。同时，感谢实习 Mentor 在 Proteus 系统架构设计与跨层优化实现上给予的宝贵工业界视角与专业指导。

## 参考文献

- [1] Woosuk Kwon, Zhuohan Li, Siyuan Zhang, Xiguang Zhuang, Ying Sheng, Cody H. Zheng, Lianmin and Iyengar, Yunsheng Ruan, Eric Xiang, Ion Stoica, et al. Efficient memory man-

- agement for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. ACM, 2023.
- [2] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538. USENIX Association, 2022.
  - [3] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
  - [4] John Nagle. Congestion control in IP/TCP internetworks. RFC 896, January 1984.
  - [5] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 197–210. ACM, 2017.
  - [6] Jiachen Liu, Jae-Won Chung, Zhiyu Wu, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services, 2024.