从余烬到烈焰：基于元强化学习的快速交互式视频自适应

肖学斗[†‡]、严明轩[†‡]、左莹莹[†]、刘博熙[§]、阮保罗[§]、曹阳[†]、Wei Wang[†*]
[†]华中科技大学电子信息与通信学院，中国[§]腾讯科技有限公司，中国 邮箱：{xuedouxiao, mingxuanyan, yingyingzuo, ycao, weiwangw}@hust.edu.cn, {percyliu, paulruan}@tencent.com

arXiv:2301.05541v1 [cs.MM] 13 Jan 2023

摘要—**最大化交互式视频流的体验质量（QoE）一直是一项长期挑战，因为其对时延敏感的特性使其更容易受到带宽波动的影响。尽管强化学习（RL）已展现出巨大潜力，但现有工作要么受限于固定模型，要么需要大量数据和时间进行在线自适应，难以适应不断变化且多样化的网络状态。基于这些实际考量，我们对企业微信互动视频服务进行了大规模测量，以研究真实世界网络波动。令人惊讶的是，我们的分析表明，与随时间变化的网络指标相比，网络序列表现出显著的短期连续性，足以满足小样本学习的需求。因此，我们提出 Fiammetta，这是首个基于元强化学习的交互式视频流码率自适应算法。利用短期连续性，Fiammetta 通过离线元训练积累学习经验，并通过少量梯度更新，快速在线适应不断变化的网络状态。此外，Fiammetta 创新地引入探测机制，以实时监控网络状态，并提出自适应元测试机制，实现无缝适应。我们在一个测试平台上实现了 Fiammetta，该平台的端到端网络遵循真实世界企业微信轨迹数据。结果表明，Fiammetta 显著优于以往算法，在不增加卡顿率的情况下，将视频码率提升了 3.6%-16.2%。**

索引词—**交互式视频流、码率自适应、元强化学习**

I. 引言

近年来，视频流经历了演进，从传统的视频点播（VoD）、直播电视发展到超低延迟交互式视频应用，例如微信、Skype、Zoom、Facetime 等。尤其是新冠肺炎疫情暴发后，社交距离措施限制了人们的活动，数字课堂 [1], 视频会议 [2], 电子商务 [3], 等需求大幅增长。Polaris Market 报告称，到 2028 [4], 全球交互式视频市场规模预计将达到惊人的 102.3 亿美元。

尽管发展迅速，交互式视频流的体验质量（QoE）仍不尽如人意，常见问题包括令人烦恼的极度模糊画面和频繁卡顿。其内在原因在于，交互式视频流极易受到带宽波动的影响，这是因为*(i)*其时延要求最为严格（例如 200 毫秒），限制了缓冲区大小及应对带宽波动的能力；*(ii)* 其 RTP/UDP 协议难以实现可靠传输；*(iii)* 与视频点播（VoD）这类大批量数据传输相比，实时采集和编码更容易浪费带宽。

为提升交互式视频体验质量，研究人员在码率自适应算法方面投入了大量研究。然而，无论是基于规则的算法 [5]–[7]，还是基于学习的算法 [8]–[16]，都各有局限。*(i)* 基于规则的算法 [5]–[7]通常采用通用的预编程规则。近期大量文献表明，这些固定规则难以适应由异构网络（如 WiFi、4G、5G）、复杂环境（如移动性、室内/室外、密度）等因素导致的多样且不断变化的网络状态。例如，在 WebRTC 中实现的 Google 拥塞控制（GCC） [5], 策略过于保守且缺乏灵活性，导致带宽严重浪费。*(ii)* 现有离线训练的神经网络（NN） [8]–[11]也受限于其固定参数，陷入了与基于规则的算法相同的困境。*(iii)* 然而，现有基于在线学习的码率自适应算法 [12]–[16]（主要是迁移学习）在面对变化的网络状态时，需要大量数据/时间，因此难以实现快速适应。此外，神经网络的实时更新可能会收敛到局部最优，而试错过程还会进一步降低性能。

受这些实际问题的驱动，我们希望回答一个关键问题：互动视频能否快速适应多样且随时间变化的网络状态，将体验质量推向极限？为此，我们首先对企业微信互动视频服务中的真实世界网络波动进行了大规模测量。分析表明，尽管网络指标剧烈波动，但由 {mean μ、标准差 σ、波动 ω、范围 Δ} 定义的网络序列仍具有明显的短期连续性，足以满足小样本学习的要求。

受短期连续性的启发，我们提出了 Fiammetta，这是首个基于元强化学习（RL）的交互式视频系统码率自适应算法，旨在最大化体验质量（QoE）。Fiammetta 利用“学会学习”的优势，保留以往的学习经验（即离线元训练），从而能够通过少量梯度更新（即元测试）快速在线适应不断变化的网络状态。元学习是适用于少样本场景的典型框架，但 Fiammetta 并非只是简单的

*通讯作者为 Wei Wang（weiwangw@hust.edu.cn）。
‡两位作者贡献相同。

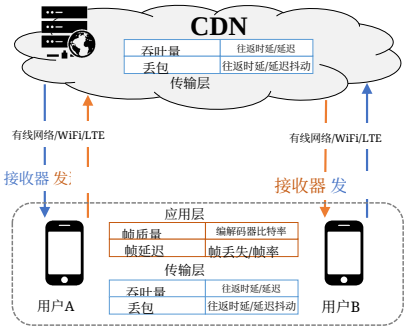


图 1： 企业微信交互式视频系统架构。

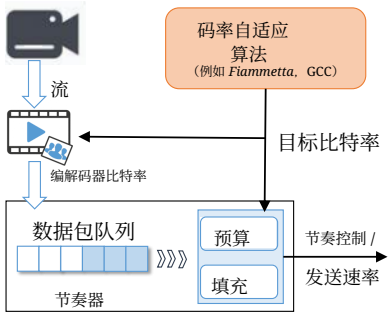


图 2： 交互式视频系统的内置码率控制模块。

功能	描述
时间跨度	2022年1月13日至15日， 2022年5月15日至17日
视频时长	1402382 秒
视频会话	14428
网络类型	有线网络，WiFi，LTE （4G、5G）
国家	200 多个国家
卷	7 Gb

表 I： 数据集描述。

环境迁移。相反，它面临着三项独特挑战。

（*i*）如何定义交互式视频流中的“任务”？元学习利用从先前“任务”中获得的学习经验来加速在线学习。自然，我们将适应不同“网络状态”的过程视为“任务”。问题由此转变为如何定义“网络状态”，以客观地界定“任务”，因为丢包、往返时延、吞吐量等众多网络指标都会受到码率自适应策略的显著影响。一旦使用这些指标，“任务”在策略学习过程中就无法保持恒定。因此，我们专门采用带宽和传播时延来表征环境、交通流密度、运动状态、链路长度等因素，而这些因素受比特率选择的影响较小。在此基础上，我们定义带宽序列的 $\{\mu, \sigma, \omega\}$，并为“任务”设置范围 Δ，以增强其连续性。

（*ii*）如何实时估计“网络状态”，以确定其是否为一个新“任务”？准确估计带宽颇具挑战，因为带宽不像往返时延、吞吐量、丢包等指标那样可以直接测量。基于规则的算法利用稳定的带宽探测机制，通过加法或乘法方式提高比特率。现有的强化学习方法则借助高风险的反复试错来探测带宽。然而，元强化学习在带宽增加时更可能低估带宽，因为子神经网络对当前或上一个“任务”的专用程度更高，导致试错动作更少。为解决这一问题，我们创新性地将探测机制融入元强化学习，提出一种带宽估计与过滤机制。具体而言，利用丢包和往返时延对吞吐量序列进行过滤，从而提供有关网络利用率的线索。在未充分利用阶段，带宽估计值为吞吐量加上一个探测值。这种设计能够快速估计“网络状态”，并利用范围来减轻偏差的影响。

(iii) 如何保证无缝适应？元测试所花费的时间可能会导致系统无法及时适应不断变化的“网络状态”，进而造成性能下降。为解决这一问题，我们提出一种自适应元测试机制，将激活阈值设为当前“网络状态”的 $\frac{\Delta}{2}$，而不是等到检测到的属性完全超出范围 Δ。此外，我们根据企业微信交互式视频系统的测量结果配置适当的 Δ，以保证无缝适应。

结果。 我们在一个测试平台上实现了 Fiammetta，该平台的端到端网络遵循真实世界企业微信轨迹。我们还部署了 3 种最先进的解决方案：基于规则的 GCC [5]、基于学习的 OnRL [12]，以及混合式 Loki [13]，使用 389 小时的视频会话进行为期半个月的评估。与基线方案相比，Fiammetta 将视频码率提高了 3.6%-16.2%，并将卡顿率降低了 6.3%-21.9%。

贡献。 *(i)* 通过大规模测量，我们深入研究了真实世界的网络波动以及网络序列的短期连续性。*(ii)* 我们提出了 Fiammetta，据我们所知，这是首个部署元强化学习以快速适应不断变化的网络状态并最大化体验质量的方法。*(iii)*我们在测试平台上实现了 Fiammetta，该平台的端到端网络遵循真实世界企业微信轨迹，并验证了其优于当前最先进方案。

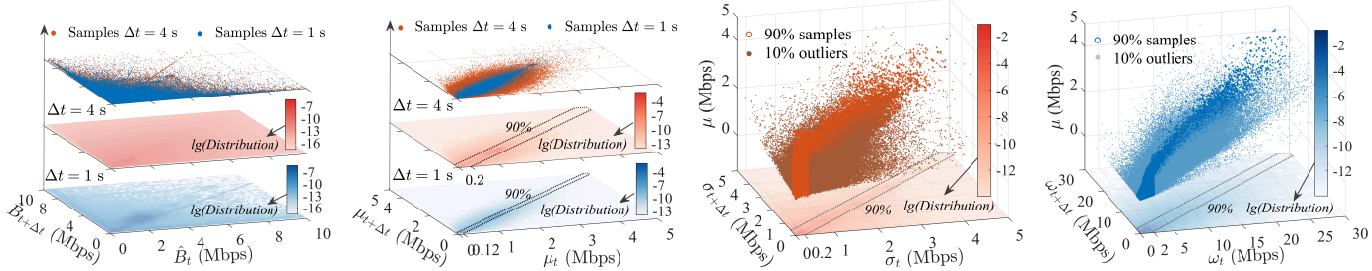
本文其余部分安排如下。§II介绍测量研究以及网络序列的短期连续性。§III详细阐述系统设计。§IV和§V分别介绍实现与评估。§VI回顾相关文献，最后§VII给出结论。

II. 测量与发现

本节对企业微信互动视频平台开展测量研究，以探究真实世界网络状态的波动特征。

A. 测量平台

我们在全球范围内记录企业微信互动视频服务的细粒度端到端网络指标和视频指标。图 1 和图 2 展示了系统架构和内置比特率控制模块。该系统整合传输层和应用层，将比特率控制模块估算出的目标比特率用于发送和编解码器码率自适应。我们在内容分发网络和用户侧设置了日志记录点。其中，只有用户具备视频编码和播放功能，可执行发送和编解码器码率自适应；内容分发网络仅负责转发，并在传输层进行发送码率自适应。因此，我们在用户侧记录传输层指标（即吞吐量、往返时延、往返时延抖动、丢包）和应用层指标（即帧率、量化参数（QP）、卡顿率和编解码器比特率），而内容分发网络侧仅记录传输层指标。



(a) ˆB Δt 前后的值。 (b) μ前后的值 Δt。 (c) σ 前后的值 Δt。 (d) ω 前后的值 Δt。

图 3： 真实世界网络波动。

B. 数据集描述

表 I 总结了测量数据集的详细信息。我们基于企业微信应用，以 1 秒粒度收集了全球视频会话对应的网络指标，时间跨度为 1月13日至15日和5月15日至17日。整个数据集包含 14428 个视频会话，总时长超过 140 万秒，总数据量为 7 Gb。这些视频会话运行于不同的异构网络（如 4G、5G、WiFi、有线网络），使用各种用户设备（如手机、平板电脑、笔记本电脑、台式电脑，甚至智能手表），覆盖全球 200 多个国家的用户。该数据集如实记录了全球真实网络中的网络波动，这些波动受到竞争流量、用户移动、通信环境、网络服务提供商和频率的影响；数据集在时间和空间上均具有广泛的覆盖范围。

C. 网络状态的短期连续性

基于数据集，我们对网络轨迹的波动特征进行了定性分析和定量检验。

如上所述，数据集由吞吐量、丢包和往返时延组成。然而，所有这些指标都受到主观因素（例如码率选择）的显著影响，无法明确表示背景网络波动（例如竞争流量、用户移动）。最理想的是可用带宽，但实时进行细粒度测量较为困难。为解决这一问题，我们提出了一种新颖的带宽过滤与估计机制，利用可测量的指标ˆ来估计带宽 B，详见§III-B，其中我们已经ˆ验证了其有效性。下文将使用 B 来研究网络波动特征。

时变特性。 为了更直观地展示网络波动，我们将所有估计带宽对分组，每对中的估计带宽分别对应同一视频会话中某个时间间隔前后的 B 。图 3(a)以散点图的形式绘制了时间间隔 Δt 为 1 秒和 4 秒时的这些带宽对，并进一步展示了它们的分布。具体而言，我们可以注意到，即使在 $\Delta t = 1$ 秒时，带宽 B 的来回波动也高度多样。带宽 B 值呈现时变特性，而且随着Δt 增加，这种特性甚至会变得更加显著。当 Δt 达到 4 秒时，波动往往更加不规则且难以预测，因为对角线附近的散点变得不那么密集。在这种情况下，元测试（在线自适应）无法跟上时变的网络波动，若仅使用带宽值来定义“网络状态”。

短期连续性。 我们进一步检验带宽序列是否具有可预测且规律的波动特征。类似地，我们在图 3(b)-3(d)中描绘了在时间间隔 $\Delta t = 4$ 秒（与§III-D中详细介绍的元测试时间相关）前后，从滑动窗口 W_r 中获得的 B 序列的均值 μ、ˆ标准差 σ和波动 ω 。其中， ω 是 W_r内相邻值绝对差之和。显然，这些序列属性表现出更强的连续性。即使 Δt 达到4秒，对应于 μ、 σ、 ω 变化的所有散点也都收敛于对角线上。此外，我们发现，通过在 μ、 σ、 ω上额外增加一个小范围，即可覆盖90%的点。也就是说，如果我们将学习任务的“网络状态”ˆi 定义为一组以 $\{\mu_i, \sigma_i \omega_i\}$ 为中心、覆盖范围为 $\mathbf{\Delta}_i$的 B 序列属性聚类，则该“网络状态”呈现出短期连续性特征。

由于即使小样本学习（即元测试过程）也只需很短时间，关键在于如何定义“网络状态”，以增强连续性并实现无缝适应。需要满足一条基本规则：在大多数时间里，网络波动都应处于为上一个“任务”定义的“网络状态”覆盖范围内。为减轻元测试延迟的影响，元测试过程基本上需要在网络变化超出上一个“任务”的范围之前完成。简而言之，元测试必须快于“网络状态”的变化。本节将观察网络序列中的短期连续性，从而使开发基于元强化学习的视频自适应算法成为可能。

III. 系统设计

A. 概述

图4展示了 Fiammetta 设计的高层概览，其中包含三个阶段：预处理（§III-B）、离线元训练（§III-C）和在线元测试（§III-D）。

(i) 预处理是后续元训练和元测试的基础，因为它包含轨迹收集、带宽估计与过滤等关键步骤，以及“网络状态”的设计与定义。

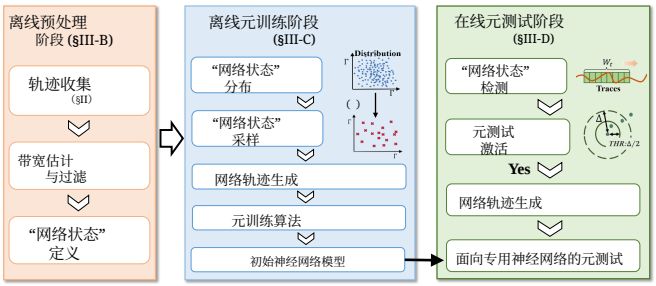


图4： 系统概述。

(ii) 元训练在线下进行。其最终目标是获得一个能够快速适应并在元测试期间最大化体验质量的初始神经网络模型。*(iii)* 元测试在线进行，通过实时检测和元测试激活机制，无缝适应不断变化的"网络状态"。

B. 预处理阶段

如图4所示，预处理阶段为后续的元训练和元测试做准备。本小节重点介绍带宽估计与过滤机制的设计，以及"网络状态"的定义。

带宽估计与过滤。 元学习中"任务"的定义往往是客观的，从而避免在特定任务的训练过程中因主观的比特率选择和策略更新而任意改变"任务"。最客观的网络指标是可用带宽和传播时延，但它们无法直接测量。因此，我们提出了一种基于可测量网络指标的带宽估计与过滤机制。该机制可能不够精确，但能够快速估计"网络状态"的变化趋势，并利用所定义的范围来减轻偏差的影响。

带宽估计的基本原理是：当存在由拥塞导致的丢包或排队时延时，网络处于满管道阶段，吞吐量 η_t 可视为带宽 [6]。满管道状态通过以下条件进行识别（参见[6]）：

$$F_t = (l_t > 5\%) \;||\; (d_t > d_{prop,t} + \sigma(d_{prop})), \tag{1}$$

$$d_{prop,t} = \min(d_{t'}, \max(d_{prop}))), \quad t' \in [\max(t - W_d, 0), t], \tag{2}$$

其中，l_t 表示时间 t的丢包率，d_t 表示分组时延，$d_{prop,\,t}$ 表示传播时延，而 $F_t = 1$ 表示满管道。其中，5%的阈值（参见 [5], [17]）考虑了非拥塞丢包的存在，例如有损无线链路、路由器端口抖动等，这些并非由传输层比特率自适应导致。分组时延 d 是排队时延、传播时延 d_{prop}及其他较小值之和。由于路径变化的时间尺度远大于d_{prop}，因此$d_{prop,\,t}$ 可以估算为较长时间窗口 W_d [6]内的滑动最小值。一旦 d_t 超过$d_{prop,\,t}$，我们通常就认为存在排队时延，即处于满管道状态。此处，d_{prop} 是通过数据集（§II-C）中的所有 W_d 个窗口收集到的统计量，而 $\sigma(d_{prop})$用于容忍一些并非由拥塞导致的不规则时延抖动。此外，配置max(d_{prop})是为了避免视频会话开始时$d_{prop,\,t}$ 被高估。然后，我们提出将带宽 $B_t(Mbps)$估算如下

$$\hat{B}_t = \begin{cases} \eta_t, & F_t = 1, \\ \max(\eta_t + pb_t^2, pb_t^1), & F_t = 0. \end{cases} \tag{3}$$

在未填满条件下，我们基于η_t，通过一种同时涉及加性增长和乘性增长的探测机制进行估计。其中，pb_t^2 设置为 Δ_μ，而 pb_t^1 通过以下公式计算：

$$pb_t^1 = \begin{cases} \eta_t \times (e^{-\eta_t - 1.3} + 1), & F_{t-1} = 1, F_t = 0, \\ pb_{t-1}^1 \times (e^{-pb_{t-1}^1 - 1.3} + 1), & F_{t-1} = F_t = 0. \end{cases} \tag{4}$$

这里，pb_t^2 是一个恒定探测值，不会随时间增加，用于补偿 $\eta < B$这一一般现象。相比之下，pb_t^1 通过持续的乘法增长进行探测，以适应带宽增量。此外，当离线使用此带宽估计与滤波算法时，例如在一个ˆ数据集中，我们可以提前获得满管道ˆ中的后续 B_{later}，并利用它在未填满条件下调整 B_t，这

$$t' = \arg\max pb_{t'}^1,\ pb_{t'}^1 < \hat{B}_{later}. \tag{5}$$

$$\hat{B}_t = \begin{cases} \max(pb_{t'}^1 - \eta_{t'}, pb_t^2) + \eta_t & t > t' \\ \hat{B}_t, & else. \end{cases} \tag{6}$$

图5和图6展示了我们估计方法的有效性，无论比特率由Fiammetta还是GCC [5]控制。借助我们的测量测试平台（详见§IV），我们可以控制带宽轨迹。与原始的吞吐量与带宽差距相比，我们的估计算法能够显著降低估计误差。此外，通过范围 **Δ**，还可以进一步减轻这些误差的影响，将"网络状态"估计的偏差率从0.5降低至约0.1。

"网络状态"的定义。 我们将对"网络状态"的适应定义为"任务"，二者均表示为 $\Gamma_i, i \in \mathbb{N}^+$。每个 Γ_i 表示具有相似特征的网络序列，即位于 $\{\mu_i,\ \sigma_i\omega_i, d_{prop,i}, \mathbf{\Delta}_i\}$的特定属性簇内，其中 μ_i, σ_i, ω_i, $d_{prop,i}$分别表示中心和$\mathbf{\Delta}_i$ 范围。所有这些属性均在窗口 W_r 内评估（设为 8 s），单位时间为 1 s。其中，**Δ** 的设计至关重要，因为 **Δ**越小，任务特定更新的改进幅度越大，但也越容易受到元测试滞后的影响。因此，我们依据 §II-C 设置最小阈值 $\mathbf{\Delta}' = \{\Delta'_\mu = 0.2, \Delta'_\sigma = 0.2, \Delta'_\omega = 2\}$（Mbps），并以相同方式设置 $\Delta'_{d_{prop}} = 3$ ms。这些 **Δ**′一方面覆盖了90% 的样本（$\Delta t = 4$ s），如图7所示，从而确保元测试期间能够平滑适应；另一方面，简化了 Γ空间的复杂度，降低了元学习的难度。

C. 离线元训练

元训练在线下实现。关键步骤和架构分别如图4和图8所示。

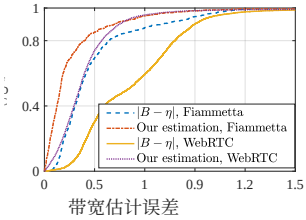


图5： 带宽估计。

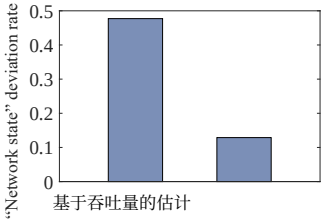


图6： “网络状态”估计。

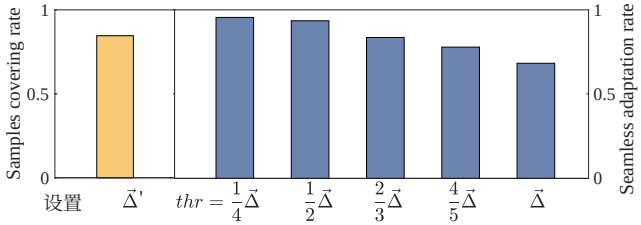


图7： $\boldsymbol{\Delta}'$ 和元测试 thr的影响。

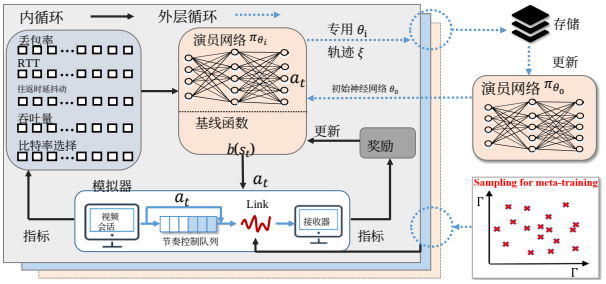


图8： Fiammetta 元训练架构

“网络状态”分布。在一系列元学习算法中，我们选择模型无关元学习（MAML） [18]。MAML本质上是针对给定的“任务”分布，训练具有高敏感性的初始神经网络参数，从而只需少量梯度步数即可极高效地适应“任务”。因此，我们需要获得“网络状态”分布。具体而言，我们首先对§II-C中收集的数据集应用带宽估计算法（§III-B）。然后，我们利用滑动窗口 W_r收集所有网络序列的 $\{\mu_t, \sigma_t, \omega_t, d_{prop,t}, \boldsymbol{\Delta}_t\}$，其中 $\boldsymbol{\Delta}_t = \{|\mu_t - \mu_{t-\Delta t}|, \cdots, |d_{prop,t} - d_{prop,t-\Delta t}|\}$。我们将每个网络序列视为 Γ_i 的“网络状态”：$\{\mu_i = \mu_t, \sigma_i = \sigma_t, \cdots, \boldsymbol{\Delta}_i = \max(\boldsymbol{\Delta}_t, \boldsymbol{\Delta}')\}$，无需额外分类。值得注意的是，为了保持“网络状态”的短时连续性，$\Delta t = 4$ s与§III-B和§II-C中的取值一致。最后，基于多维“网络状态”，我们通过细粒度划分计算其联合概率密度，再结合插值，得到概率密度函数 $p(\Gamma) = p(\mu, \sigma, \omega, \boldsymbol{\Delta}) \times p(d_{prop})$。

“网络状态”采样。基于 $p(\Gamma)$，我们对“网络状态”进行采样 Γ_i，$i \in \mathbb{N}^+$ 为元训练过程提供任务，如图8所示。从理论上讲，元训练过程中采样的任务越多，采样任务的分布与 $p(\Gamma)$ 之间的差异就越小，在 $p(\Gamma)$上的性能也就越稳健。

网络轨迹生成。对于每个任务，都需要足够的网络轨迹作为训练样本。然而，在元训练期间，某些概率较低的“网络状态” Γ 常常会遇到收集到的网络轨迹不足的问题。此外，在元测试期间，也没有足够的时间获取新“网络状态”的网络轨迹。因此，除了收集真实轨迹外，我们还探索了第二种方法，即生成合成网络轨迹。

给定一个 Γ_i，我们首先按照高斯分布，在各自的范围内对 μ、σ、d_{prop} 进行采样，以确保中心处的采样概率略高。同样，我们也以相同方式对 d_{prop} 进行采样。这样做的目的是为每个 Γ_i提供优化偏好。然后，我们通过两种分布生成带宽样本。一种是高斯分布，常用于合成轨迹生成研究 [8], [19]；另一种是在范围 $[0, max]$ 上构建的贝塔分布，可以轻松实现非对称采样。这里，max 是数据集中的最大 B。对于任意给定的 μ 和 σ，如果 $[\mu - 3\sigma, \mu + 3\sigma] \subseteq [0, max]$ （3σ 规则 [20]），我们假设带宽样本服从高斯分布；否则，则采用贝塔分布作为替代。最后，我们重复前两个步骤以获得足够的带宽样本，然后按不同顺序将它们排列成轨迹，并滤除不满足Δ_{μ_i}、Δ_{σ_i}、Δ_{ω_i} 的轨迹。

初始神经网络模型的元训练。图8展示了 Fiammetta 的元训练架构。下面介绍其中采用的定制化设计和关键算法。

（*i*）状态与动作。在任意时刻 t，强化学习智能体将状态 s_t 作为输入，以神经网络作为函数 π_{θ_t}，并输出目标比特率 a_t，与交互式视频系统进行交互。最终目标是找到最优的比特率自适应策略，即 $\pi_\theta^*: s_t \to a_t$，以最大化体验质量。具体而言，$s_t$ 记为 $\{\boldsymbol{\eta}_t, \boldsymbol{b}_{t-\Delta t'}, \boldsymbol{l}_t, \boldsymbol{d}_t, \boldsymbol{j}_t\}$，表示过去 3 s 内吞吐量、目标比特率、丢包率、时延和时延抖动的序列，其中 $\Delta t' = 0.1$ s 为时间单位。所有这些指标都可由发送端通过周期性 RTCP反馈获取。在观测到 s_t 后，强化学习智能体从包含 21 个离散动作的集合 $\{-2, -1.8, \cdots, 1.8, 2\}$ 中选取并输出 a_t。这里，a_t表示连续两次目标比特率选择之间的缩放因子，即 $b_t = b_{t-\Delta t'} \times e^{a_t}$，类似于 Libra [21]。

（*ii*）奖励。确定 a_t 后，Fiammetta 通过调整编解码器比特率和发送速率与交互式视频系统进行交互，随后获得奖励 r_t，以更新 π_θ。我们采用体验质量作为设计 r_t 的标准，具体如下：

$$r_t = w_1 \times \eta_t - w_2 \times l_t - w_3 \times d_t - w_4 \times |b_t - b_{t-\Delta t'}|. \quad (7)$$

其中，所有这些指标均在一个单位时间内取平均值，即从 $t - \Delta t'$ 到 t。$|b_t - b_{t-\Delta t'}|$ 强制保证编解码器比特率的平滑性，以防止出现较大的帧延迟抖动和质量抖动。参考近期研究 [12], [13], ，我们通过经验将这四个权重分别设为 50、50、200 和 20。

（*iii*）神经网络结构。首先，将状态序列展平后输入网络。演员网络由

三个全连接层组成，分别包含 128、64 和 32 个神经元，之后接一个激活函数。基线函数 $b^{\pi_\theta}(s)$ 只需通过线性拟合，实现为 s 在 π_θ 下的平均期望奖励。

（*iv*）训练算法。整个元训练由两部分组成：内循环和外层循环。内循环负责任务特定优化，外层循环则用于获得高效的初始神经网络模型，从而能够在内循环中快速适应。

在内循环中，每个 Γ_i 都包含一个初始状态分布 $p_i(s_t)$和一个转移分布 $p_i(s_{t+\Delta t'}|s_t,\ a_t)$。因此，$\Gamma_i$ 是一个马尔可夫决策过程（MDP）。每次梯度更新期间，强化学习智能体可以查询 K轨迹 $\xi_k = \{s_{\Delta t'}, a_{\Delta t'}, r_{\Delta t'}, \cdots, s_N, a_N, r_N\}(k = 1, \cdots, K)$；这些轨迹由 π_{θ_i}（初始设为 θ_0）在网络轨迹（详见上文）和模拟器（§IV）上通过滚动采样得到。随后，累积奖励 $R^{\pi_{\theta_i}}(s_t\ a_t = \sum_{t'=\frac{t}{\Delta t'}}^{\frac{t+\Delta t}{\Delta t'}} \gamma^{(t'-\frac{t}{\Delta t'})} r_{t'\times\Delta t'}$ θ_i ,)用于按如下方式更新

$$\mathcal{L}_{\Gamma_i}(\theta_i) = \mathbb{E}_{\xi_k\sim(\pi_{\theta_i},p_i)}\left[\sum_t^{\xi_k} \hat{A}^{\pi_{\theta_i}}(s_t,a_t)\right], \tag{8}$$

$$\theta_i \leftarrow \theta_i + \alpha\Delta_{\theta_i}\mathcal{L}_{\Gamma_i}(\theta_i), \tag{9}$$

其中，$\mathcal{L}$ 为丢包函数，α 为学习率，$b^{\pi_{\theta_i}}(s_t)$表示在 s_t 时、依据 π_{θ_i}得到的平均期望奖励，$\hat{A}^{\pi_{\theta_i}}(s_t,a_t) = R^{\pi_{\theta_i}}(s_t,a_t) - b^{\pi_{\theta_i}}(s_t)$为优势函数，表示某个 a_t带来的额外收益。此外，“3 s”是根据经验设定的时间，表示 Fiammetta 对未来的考虑范围。在适应 Γ_i时，神经网络参数根据公式 (9)，通过仅 3 次梯度更新从θ_0 演变为 θ_i，这足以实现收敛。

外层循环通过更新初始 θ_0，进一步提升 π_{θ_i}的性能，具体采用近端策略优化算法 [22]。在每轮外层循环中，Fiammetta 从$p(\Gamma)$中采样 M 个任务，在内循环中更新至相应的 $\{\theta_i\}_{i=1}^M$，然后使用 π_{θ_i}，通过对每个任务重新采样 K 条轨迹 $\{\xi_{k,i}\}_{k=1}^K$ 来评估。接着，Fiammetta 得到丢包函数 $\mathcal{L}$，并通过以下方式更新 θ_0

$$\mathcal{L}^{\theta_0'}(\theta_0) = \mathbb{E}^{i\sim p(\Gamma)}_{\xi_{k,i}\sim(\pi_{\theta_i},p_i)}\left[\sum_t^{\xi_{k,i}} \min\left(\delta^{\theta_0'}(\theta_0)\hat{A}^{\pi_{\theta_i}}(s_t,a_t),\right.\right.$$

$$\left.\left. clip(\delta^{\theta_0'}(\theta_0), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_{\theta_i}}(s_t,a_t)\right)\right], \tag{10}$$

$$\theta_0 \leftarrow \theta_0 + \beta\Delta_{\theta_0}\mathcal{L}^{\theta_0'}(\theta_0), \tag{11}$$

其中，θ_0' 是每轮外循环更新前的旧初始参数，而 $\delta^{\theta_0'}(\theta_0) = {}^{\pi_{\theta_0}(s_t,a_t)}\pi_{\theta_{old,0}(s_t,a_t)}$ 表示新策略与旧策略的比值。基本思路是通过裁剪超出 $[1-\epsilon, 1+\epsilon]$ 范围的 $\delta^{\theta_0'}(\theta_0)$ 值，使 θ_0 的更新更加平滑，并避免梯度振荡。

D. 在线元测试

基于初始神经网络模型，Fiammetta 实现了无缝—通过在线元测试，更少地适应时变的“网络状态”；其步骤如图4和图9所示。

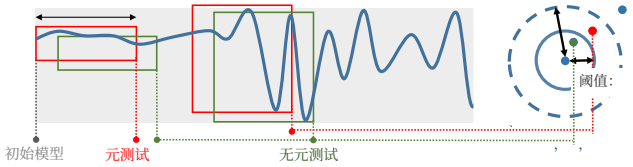


图9： Fiammetta元测试设计。

“网络状态”监测。Fiammetta通过计算过去 8 s 内滑动窗口 W_r 中的网络序列的 μ_t、σ_t、ω_t 和 $d_{prop,\ t}$，对实时“网络状态”进行监测。ˆ值得注意的是，带宽 B 和 d_{prop} 按照 §III-B 中的步骤进行估算，并以 1 s 的粒度移动滑动窗口，以实现实时保障。

元测试激活。元测试激活的目标是在“网络状态”发生变化时实现无缝适应。因此，不能等到检测到的属性（例如 μ_t、σ_t、ω_t、$d_{prop,\ t}$）超出上一“任务”的范围 $\mathbf{\Delta}_{last}$后才进行激活。作为替代方案，我们根据图中的情况将激活阈值设为 $\frac{\mathbf{\Delta}_{last}}{2}$ 7, $|\mu_t - \mu_{last}| > \Delta_\mu$ 即每个 $\frac{last}{2}$、$|\gamma_t - \gamma_{last}| > \frac{\Delta_\gamma\ last}{2}$、$\cdots$都会激活元测试。这样，通过减轻图9所示的元测试延迟影响，即可确保无缝适应。需要注意的是，$\mathbf{\Delta}$ 也必须与该延迟相匹配。根据我们的统计，元测试（即包含3次梯度更新的内循环）大约需要2秒，因此，$\mathbf{\Delta}$ 需要至少覆盖4秒的网络变化，这也是根据此配置 $\Delta t = 4$ 秒（§II-C）并获得 $\mathbf{\Delta}'$（§III-C）的原因。

元测试。与内循环一致，元测试负责适应实时检测到的不同“网络状态”。激活后，会生成新的“网络状态”Γ_i：$\{\mu_i = \mu_t, \sigma_i = \sigma_t, \omega_i = \omega_t, \mathbf{\Delta}_i = \max(\mathbf{\Delta}_t, \mathbf{\Delta}')\}$，其中 $\mathbf{\Delta}_t = \{|\mu_t - \mu_{last}|, |\sigma_t - \sigma_{last}|, |\omega_t - \omega_{last}|\}$。随后，我们遵循内循环的工作流程，首先采样 Γ_i的K 条轨迹，这些轨迹同样通过在合成网络轨迹（§III-C）和模拟器（§IV）上进行 K 滚动采样获得。然后，根据式（8）-（9）进行 3 次梯度更新，即可得到适用于 Γ_i 的专用神经网络参数 θ_i。

IV. 实现

测试床实现。我们搭建了一个端到端测量测试平台，并利用企业微信应用提供的真实世界网络轨迹（详见§II），测试 Fiammetta 和基线算法的性能。如图10所示，测试平台主要由两台运行 WebRTC、组成视频流量收发器对的个人计算机，以及一台通过 TC（流量控制）工具控制网络链路的个人计算机组成 [23]。此外，我们在远程强化学习服务器上实现了 Fiammetta 和基于学习的基线算法，视频收发器对通过额外的路由器连接到强化学习服务器，以查询目标比特率。强化学习服务器是一台配备英特尔酷睿 i7-9700K 处理器、GeForce RTX 1080 Ti 显卡、32 GB 内存并运行 Ubuntu 18.04 的台式机。我们使用 PyTorch 1.10.2 版实现 Fiammetta。图11进一步展示了 Fiammetta 的详细系统实现：它通过替换交互式视频系统中的码率控制模块来运行



图10： 测试平台设置。

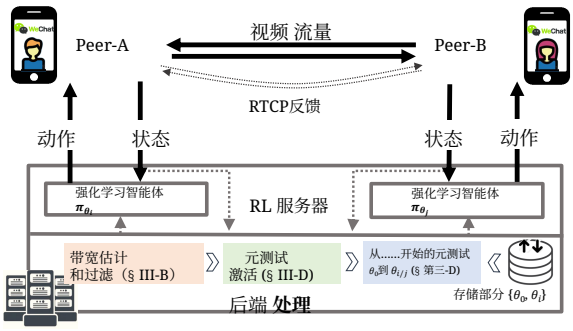


图 11： Fiammetta 的系统实现。

。在运行时，发送端会实时将网络指标（通过 RTCP反馈获取）发送到强化学习服务器，并采用强化学习模型 π_{θ_i} 选定的码率决策 a_t。此外，我们将元测试与前端查询服务分离，以确保响应及时。具体来说，我们设置了一个后端进程，用于接收来自前端的转移元组，并在不干扰前端的情况下进行元测试。元测试完成后，前端的旧神经网络参数 θ_{old}将被新参数 θ_{new}替换。为进一步降低计算开销，除了初始神经网络参数外，我们还根据访问频率，在服务器上动态存储部分专用神经网络参数 θ_i。

模拟器。我们的模拟器由两个模块组成：（*i*）视频压缩模块，可将真实视频压缩为不同的比特率（$\{0.1, 0.2, \cdots, 2.5\}$ 兆比特每秒），并记录每帧大小，以忠实模拟帧大小抖动。运行时，该模块会持续输出与目标比特率最接近的视频帧大小。（*ii*）传输模块，将随机数据封装成与帧大小相同的 RTP 数据包，并忠实模拟发送节奏控制机制，将数据包发送到模拟的网络路径中。带宽和传播时延根据用于 Fiammetta 训练的企业微信轨迹数据进行配置。

V. 评估

A. 方法

轨迹驱动的测试平台实验。我们使用同一数据集（详见 §II）训练和测试 Fiammetta 与基线算法，其中 75% 用作训练集，25% 用作测试集。由于难以精确测量细粒度带宽，我们在实验中使用吞吐量作为带宽。这无疑通过获得精确的“网络状态”分布 $p(\Gamma)$简化了实验。不过，我们希望明确指出，我们的设计无需显式带宽信息即可适用于真实环境部署。

基线算法如下所示

(i) GCC [5]，作为一种官方拥塞控制算法，GCC 广泛应用于主流交互式视频系统，例如微信、Google Hangouts 等。其核心思想是结合基于丢包和基于时延梯度的方法，以避免拥塞并调整比特率。

(ii) OnRL [12] 是首个基于在线强化学习的交互式视频系统适应算法。我们直接在测试平台上使用同一训练集对其进行训练，耗时约 12 天、共 291 小时；并在为期 4 天、共 100 小时的测试期间实施在线自适应。

(iii) Loki [13]提出在特征层面融合基于规则的 GCC 与基于强化学习的算法，以提升长尾性能。借鉴 Loki，我们将 GCC“黑盒化”，并将其与 OnRL 集成，然后在我们的测试平台上使用相同的训练集重新训练新的神经网络。测试过程中也会进行在线自适应。

B. 与基线算法的比较

比较结果总结于表 II。

应用层指标。 Fiammetta 的表现显著优于所有三种竞争算法：*(i)* 与 GCC 相比，Fiammetta 的峰值信噪比提高了 1.34 dB，卡顿率降低了 9.6%。更重要的是，在帧率、帧延迟和抖动方面，Fiammetta 也可与保守的 GCC 相媲美。*(ii)* 与基于学习的 OnRL 和 Loki 相比，Fiammetta 分别将卡顿率降低了 21.9% 和 6.3%，并且帧质量更佳，峰值信噪比和帧率均略有提升。此外，Fiammetta 的帧延迟和时延抖动表现也优于 Loki 和 OnRL。

传输层指标。 同样，Fiammetta 在大多数传输层指标上都取得了显著提升：*(i)* 与应用层指标的结果一致，Fiammetta 的吞吐量相较于 OnRL、Loki 和 GCC 分别提高了 3.6%、10.3% 和 16.2%。*(ii)* 与此同时，与 Loki 和 OnRL 相比，Fiammetta 的往返时延分别显著降低了 1.1% 和 7.4%，丢包率也分别降低了 12.3% 和 17.5%。这些指标与保守的 GCC 处于同一水平。*(iii)*虽然 Fiammetta 的码率抖动略逊于 OnRL，但与 GCC 相当，且优于 Loki。

体验质量/奖励。 表II进一步展示了体验质量/奖励，以提供整体评估。为确保比较公平，所有基于强化学习的算法均使用§III-C中定义的相同体验质量/奖励设置进行训练。可以看到，Fiammetta的体验质量比OnRL高11.1%；与Loki和GCC相比，差距则逐渐扩大至17.0%和26.7%。这些结果验证了Fiammetta的整体优势：它能更好地动态平衡时延、丢包、平滑度和吞吐量等相互冲突的指标，从而最大化交互式视频体验质量。

表 II： 应用层和传输层指标的整体性能 ($Mean \pm Std\ D \qquad ev$)

算法	应用层指标					传输层指标				QoE/奖励
	FPS	PSNR (dB)	卡顿率 （帧率 <12，%）	帧延迟 (ms)	帧延迟 抖动（毫秒）	RTT (ms)	丢包率 (%)	吞吐量 （兆比特每秒）	码率抖动 （兆比特每秒/10分钟）	
GCC [5]	29.41 ± 4.41	33.17 ± 7.98	0.83 ± 5.27	**160.23 ± 45.42**	**5.25 ± 10.35**	68.93 ± 110.27	1.68 ± 1.01	0.74 ± 0.23	1.09 ± 2.04	22.85
OnRL [12]	29.02 ± 4.29	34.12 ± 7.32	0.96 ± 3.55	170.68 ± 36.32	5.53 ± 10.82	73.95 ± 119.01	2.00 ± 1.33	0.83 ± 0.25	**0.92 ± 5.20**	26.05
Loki [13]	29.04 ± 4.36	33.82 ± 6.55	0.80 ± 2.32	165.21 ± 38.34	5.343 ± 10.92	69.29 ± 97.86	1.88 ± 1.25	0.78 ± 0.22	1.20 ± 1.81	24.74
Fiammetta	**29.51 ± 3.41**	**34.51 ± 7.27**	**0.75 ± 2.31**	162.53 ± 30.24	5.27 ± 9.63	**68.51± 63.86**	**1.65 ± 1.25**	**0.86 ± 0.24**	1.09 ± 2.16	**28.94**

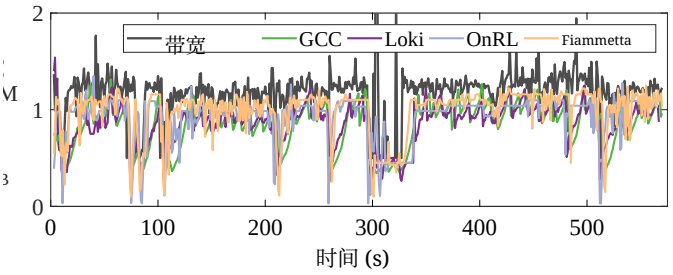


图 12： 600秒会话的微观展示。

微观案例展示。 我们发现，Fiammetta的优势源于它能够快速生成并切换到适合当前“网络状态”的策略。图12展示了一个具有代表性的600秒会话。*(i)* 当带宽相对稳定时，Fiammetta能够紧密跟随带宽变化，以最小幅度进行探测，从而最大限度地利用带宽。这是因为Fiammetta始终能够对“网络状态”形成粗略的先验假设。在运行时粗略检测到“网络状态”后，Fiammetta便能快速生成适应该“网络状态”范围的策略，从而避免大量反复试错。相比之下，GCC受基于AIMD的探测机制影响，比特率会周期性下降，而OnRL则频繁出现过冲。由于与GCC集成，Loki也会遭遇不必要的比特率下降：当GCC的比特率降低并成为影响更大的因素时，带宽利用率便会受损。*(ii)* 当带宽剧烈波动时（280–370秒），Fiammetta也能利用先验假设生成相较于OnRL更保守的策略，使往返时延与GCC相当，同时获得更高的吞吐量。

C. 不同“网络状态”下的比较

我们进一步评估了 Fiammetta 在大幅/小幅、稳定/动态带宽下的表现，结果如图 13 所示。

大幅稳定带宽。 我们首先在带宽大幅且稳定的最高质量“网络状态”下评估 Fiammetta（$\mu \in [1,3]$ 和 $\sigma \in [0,0.1]$）。与三种基线算法相比，Fiammetta 显著提升了视频清晰度，使吞吐量提高了 9.5%- 21.4%，峰值信噪比提高了 0.2 dB-2.6 dB。同时，Fiammetta 在往返时延和卡顿率方面也表现更佳。这些结果进一步验证了关于“网络状态”的先验粗略假设有助于 Fiammetta 实现更好的性能。

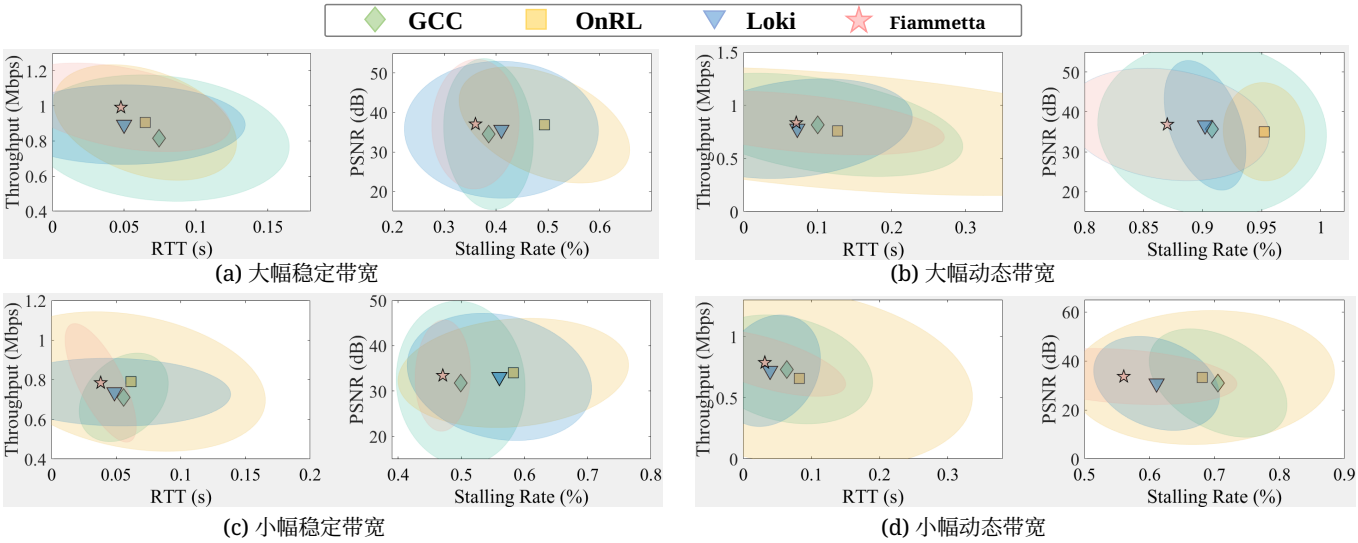
大幅动态带宽。 对于带宽较大但动态变化的“网络状态”（$\mu \in [1,3]$ 和 $\sigma \in [0.3,0.5]$），Fiammetta 仍实现了 2.2%-10.3% 的吞吐量提升，以及 0.2 dB-1.9 dB 的峰值信噪比提升。同时，与基线算法相比，其往返时延和卡顿率均明显降低。这表明，当带宽波动时，Fiammetta 能够迅速切换到相对保守的策略，以避免时延大幅增加。

小幅稳定带宽。 在适应小幅稳定带宽的“网络状态”（$\mu \in [0.5,1]$ 和 $\sigma \in [0,0.1]$）时，Fiammetta 的卡顿率与 GCC 相当，且显著优于 Loki 和 OnRL。此外，Fiammetta 的峰值信噪比和吞吐量介于 OnRL 与 Loki 之间，并在体验质量方面达到全局最优。这些结果表明，Fiammetta 已针对小幅稳定带宽形成了相应策略，而非盲目探测并导致频繁过冲。

小幅动态带宽。 我们认为，小幅动态带宽（$\mu \in [0.5,1]$ 和 $\sigma \in [0.3,0.5]$）代表最差的“网络状态”，但在这种情况下，Fiammetta 仍取得了令人瞩目的结果。Fiammetta 并未盲目追随带宽波动曲线，而是采用了相对保守的策略。与三种基线算法相比，其吞吐量提升了 7.5%–19.8%，峰值信噪比提升了 0.4 dB–2.8 dB，卡顿率则降低了 8.3%–20.6%。

D. Fiammetta 有无元测试的比较

在本小节中，我们将进一步探究 Fiammetta 的性能提升究竟源于在线元测试能够快速适应不断变化的“网络状态”，还是源于初始神经网络本身更优。我们开展实验，对比 Fiammetta 使用和不使用元测试时的性能。从图14的结果可以看出，与不使用元测试相比，使用元测试的 Fiammetta 在所有应用层和传输层指标上均取得了性能提升：*(i)* 在应用层指标方面，最显著的提升是峰值信噪比提高了 7.5%，卡顿率降低了 19.5%。*(ii)*在传输层指标方面，与不使用元测试的 Fiammetta 相比，吞吐量提高了 9.7%，往返时延和码率抖动分别降低了 7.8% 和 60.8%。



(a) 大幅稳定带宽 (b) 大幅动态带宽

(c) 小幅稳定带宽 (d) 小幅动态带宽

图13： Fiammetta 对不同“网络状态”的适应能力，与基线算法相比。

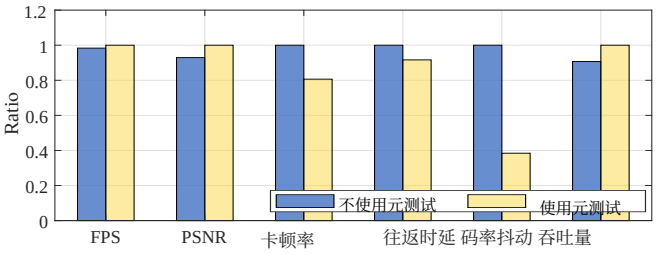


图14： Fiammetta 使用和不使用元测试的对比。

VI. 相关工作

交互式视频自适应。 人们投入了大量精力，为交互式视频流开发码率自适应算法，主要分为两类：*(i)* 基于规则的算法 [5]–[7]，和*(ii)* 基于学习的算法 [11]–[15]。基于规则的算法 [5]–[7] 通常采用类似加性增乘性减的方法，根据丢包、时延和时延抖动等因素进行码率自适应。然而，这类预先编程的通用规则很难适用于多样且异构的现代网络。基于学习的算法包括离线 [11]和在线学习方法 [12]–[15]。例如，Concerto [11] 采用离线模仿学习。OnRL [12] 是首个通过联邦学习在交互式视频应用中实现完全在线学习的方法。Loki [13]提出了强化学习与 GCC 的紧密结合。尽管如此，离线训练的模型仍受固定参数的限制。然而，这些在线学习算法 [12]–[15]，依赖大量数据和时间，因此难以快速适应不断变化的“网络状态”。此外，Salsify [24] 设计了定制编解码器，以解决实际编解码器比特率与传输比特率/目标比特率之间的不匹配问题。另外，许多相关研究 [8]–[10]，[16]，[19] 着眼于视频点播流媒体中的码率自适应，也为 Fiammetta 提供了诸多启发。

元强化学习。 元学习 [25]是解决具有挑战性的小样本学习设置的一种典型框架，它通过提出合理的假设来实现这一目标：*(i)* 存在一个任务分布-布 $p(\Gamma)$，*(ii)* 目标新任务遵循 $p(\Gamma)$；*(iii)*可以获得从 $p(\Gamma)$ 中采样的元训练集，用于训练模型；*(iv)* 还可以使用从$p(\Gamma)$ 中采样的另一个元测试集来评估预期性能 [25]。Fiammetta 基本遵循这些假设。元强化学习的研究涵盖了对初始神经网络参数 [18]、[26]，损失函数 [27]，和适应过程 [28]，的元学习，而它们都是强化学习的关键组成部分。其中，MAML [18] 是一项里程碑式的工作，它仅对初始神经网络参数进行简单调整，样本效率高、轻量，并且极其适合小样本学习设置。此外，Fiammetta 与 MAML 的约束高度兼容，例如固定的神经网络结构（如输入/输出尺寸）以及任务之间的强相关性。

VII. 结论

Fiammetta 是首个基于元强化学习的交互式视频系统比特率适应算法。它开创了一种新的优化方式：通过小样本学习，实现对不断变化的网络状态的快速适应。该系统的设计灵感来自一项对企业微信交互式视频系统开展的真实世界测量研究。研究发现，网络序列表现出明显的短期连续性，足以支持小样本学习。我们在本地测试平台上评估了 Fiammetta，其中网络链路采用企业微信轨迹数据。实验结果证实，Fiammetta 优于最先进的算法。

致谢

本研究部分经费来自国家重点研发计划项目（资助编号：2020YFB1806600）、国家自然科学基金（资助编号：62071194）、湖北省重点研发计划（资助编号：2021EHB002）、武汉市知识创新曙光计划、腾讯犀牛鸟基础平台技术重点研究项目2021，以及欧盟地平线2020研究与创新计划玛丽·斯克沃多夫斯卡-居里资助协议（编号：101022280）。

参考文献

[1] Matthew Lynch。交互式视频对课堂的影响。https://www.thetechedvocate.org/the-impact-of-interactive-videos-in-the-classroom/。在线；访问日期：2020年9月23日。[2] Justin Pot。2022年适合团队使用的最佳视频会议软件。https://zapier.com/blog/best-video-conferencing-apps/。在线；访问日期：2022年5月11日。[3] Zanna Zhang。中国的交互式电子商务如何推动电子商务的下一阶段发展。https://pandaily.com/。在线；访问日期：2020年8月13日。[4] Allan Parker。最新研究：交互式视频墙市场份额预计到2028年将达到10%的复合年增长率。https://whatech.com/og/markets-research/semiconductor-and-electronics/。在线；访问日期：2022年1月9日。[5] Gaetano Carlucci、Luca De Cicco、Stefan Holmer 和 Saverio Mascolo。Web实时通信（WebRTC）的Google拥塞控制分析与设计。载于第七届国际多媒体系统会议论文集，第1–12页，2016年。[6] Neal Cardwell、Yuchung Cheng、C Stephen Gunn、Soheil Hassas Yeganeh 和 Van Jacobson。BBR：基于拥塞的拥塞控制。《*ACM*通讯》，60(2):58–66，2017年。[7] Subir Varma。互联网拥塞控制。Morgan Kaufmann，2015年。[8] Hongzi Mao、Ravi Netravali 和 Mohammad Alizadeh。基于Pensieve的神经网络自适应视频流。载于*ACM*数据通信特别兴趣小组会议论文集，第197–210页，2017年。[9] Xutong Zuo、Jiayu Yang、Mowei Wang 和 Yong Cui。面向视频流、具有用户级QoE偏好的自适应码率。载于*IEEE INFOCOM 2022—IEEE*计算机通信会议，第1279–1288页，2022年。[10] Francis Y Yan、Hudson Ayers、Chenzhi Zhu、Sadjad Fouladi、James Hong、Keyi Zhang、Philip Levis 和 Keith Winstein。实地学习：视频流中的一项随机实验。载于第*17*届*USENIX*网络系统设计与实现研讨会（*NSDI 20*）论文集，第495–511页，2020年。[11] Anfu Zhou、Huanhuan Zhang、Guangyuan Su、Leilei Wu、Ruoxuan Ma、Zhen Meng、Xinyu Zhang、Xiufeng Xie、Huadong Ma 和 Xiaojiang Chen。学习如何协调移动视频通话中的视频编解码器与传输协议。载于第*25*届移动计算与网络年度国际会议，第1–16页，2019年。[12] Huanhuan Zhang、Anfu Zhou、Jiamin Lu、Ruoxuan Ma、Yuhan Hu、Cong Li、Xinyu Zhang、Huadong Ma 和 Xiaojiang Chen。OnRL：通过在线强化学习改进移动视频通话。载于第*26*届移动计算与网络年度国际会议论文集，第1–14页，2020年。[13] Huanhuan Zhang、Anfu Zhou、Yuhan Hu、Chaoyue Li、Guangping Wang、Xinyu Zhang、Huadong Ma、Leilei Wu、Aiyun Chen 和 Changhui Wu。Loki：融合基于规则的模型，提升基于学习的实时视频适应的长尾性能。载于第*27*届移动计算与网络年度国际会议论文集，第775–788页，2021年。[14] Lei Zhang、Yong Cui、Mowei Wang、Kewei Zhu、Yibo Zhu 和 Yong Jiang。DeepCC：通过多目标优化弥合拥塞控制与应用之间的差距。《*IEEE/ACM*网络汇刊》，2022年。[15] Yiqing Ma、Han Tian、Xudong Liao、Junxue Zhang、Weiyan Wang、Kai Chen 和 Xin Jin。多目标拥塞控制。载于第十七届欧洲计算机系统会议论文集，第218–235页，2022年。[16] Tianchi Huang、Chao Zhou、Xin Yao、Rui-Xiao Zhang、Chenglei Wu、Bing Yu 和 Lifeng Sun。结合终身模仿学习的质量感知神经网络自适应视频流。《*IEEE*通信选定领域期刊》，38(10):2324–2342，2020年。[17] Mo Dong、Tong Meng、Doron Zarchy、Engin Arslan、Yossi Gilad、Brighten Godfrey 和 Michael Schapira。PCC Vivace：在线学习拥塞控制。载于第*15*届*USENIX*网络系统设计与实现研讨会（*NSDI 18*）论文集，第343–356页，2018年。[18] Chelsea Finn、Pieter Abbeel 和 Sergey Levine。用于深度网络快速适应的模型无关元学习。载于国际机器学习会议，第1126–1135页，2017年。[19] Zahaib Akhtar、Yun Seong Nam、Ramesh Govindan、Sanjay Rao、Jessica Chen、Ethan Katz-Bassett、Bruno Ribeiro、Jibin Zhan 和 Hui Zhang。Oboe：针对网络条件自动调优视频自适应码率算法。载于*2018*年 *ACM*数据通信特别兴趣小组会议论文集，第 44–58 页，2018年。[20] 维基百科。68–95–99.7法则。https://en.wikipedia.org/wiki/68%E2%80% 9395%E2%80%9399.7_rule。在线。[21] Zhuoxuan Du、Jiaqi Zheng、Hebin Yu、Lingtao Kong 和 Guihai Chen。面向多样化应用偏好和网络条件的统一拥塞控制框架。载于第 *17* 届新兴网络实验与技术国际会议论文集，第 282–296 页，2021年。[22] John Schulman、Filip Wolski、Prafulla Dhariwal、Alec Radford 和 Oleg Klimov。近端策略优化算法。*arXiv* 预印本 *arXiv:1707.06347*，2017年。[23] Tc（Linux）。https://en.wikipedia.org/wiki/Tc_（Linux），2019年。[24] Sadjad Fouladi、John Emmons、Emre Orbay、Catherine Wu、Riad S Wahby 和 Keith Winstein。Salsify：通过更紧密地集成视频编解码器与传输协议，实现低延迟网络视频。载于第 *15* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 18*），第 267–282 页，2018年。[25] Chelsea B Finn。学习如何通过梯度进行学习。加州大学伯克利分校，2018年。[26] Alex Nichol 和 John Schulman。Reptile：一种可扩展的元学习算法。*arXiv* 预印本 *arXiv:1803.02999*，2(3):4，2018年。[27] Sungyong Baik、Janghoon Choi、Heewon Kim、Dohee Cho、Jaesik Min 和 Kyoung Mu Lee。用于小样本学习的任务自适应损失函数元学习。载于*IEEE/CVF*国际计算机视觉会议论文集，第 9465–9474 页，2021年。[28] Zhongwen Xu、Hado P van Hasselt 和 David Silver。元梯度强化学习。神经信息处理系统进展，31，2018年。