FlexSpec：冻结草稿邂逅演进目标模型：边云协同大语言模型推测解码

Yuchen Li、Rui Kong、Zhonghao Lyu、Qiyang Li、Xinran Chen、Hengyi Cai、Lingyong Yan、Shuaiqiang Wang、Jiashu Zhao、Guangxu Zhu、Linghe Kong、*IEEE*会士、Guihai Chen、*IEEE* 会士、Haoyi Xiong、Dawei Yin

arXiv:2601.00644v1 [cs.DC] 2 Jan 2026

摘要—**在移动和边缘计算环境中部署大语言模型（LLM），受到设备端资源有限、无线带宽稀缺以及模型频繁演化的制约。尽管采用推测解码（SD）的边云协同推理，可以通过在边缘端运行轻量级草稿模型、并由云侧目标模型对其进行验证来降低端到端延迟，但现有框架在根本上依赖两个模型之间的紧密耦合。因此，反复进行模型同步会带来过高的通信开销，增加端到端延迟，并最终限制 SD 在边缘环境中的可扩展性。为解决这些局限，我们提出了 FlexSpec，这是一种面向演进中的边云系统的高通信效率协同推理框架。FlexSpec 的核心设计是一种共享骨干网络架构，使单个静态边缘侧草稿模型能够与大量不断演进的云侧目标模型保持兼容。通过将边缘部署与云端模型更新解耦，FlexSpec 无需在边缘端重新训练或反复下载模型，从而显著降低通信和维护成本。此外，为适应时变无线条件和异构设备约束，我们开发了一种信道感知自适应推测机制，可根据信道状态信息和设备能量预算实时动态调整推测草稿长度。大量实验表明，与传统 SD 方法相比，FlexSpec 在推理效率方面表现更优。**

索引词—**移动计算、边缘端-云端协作、大语言模型、推测解码、协同推理。**

I. 引言

大ARGE 语言模型（LLMs），以 Chat- GPT [1], LLaMA [2], 和 Gemini [3], 为代表，已成为智能代理的关键引擎，应用范围从移动个人助理到实时翻译设备不等。此类模型的能力在很大程度上决定了用户体验以及复杂推理任务的成功执行 [4]–[7]。尽管许多研究致力于通过量化和剪枝等技术压缩边缘设备上的这些模型，但 [8],单个端侧推理仍受到电池续航有限、热节流以及内存带宽不足等因素的限制 [9]。

边云协同推理通过将计算密集型子任务卸载到高性能云服务器，同时在边缘端保留轻量级处理，克服了这一局限，从而拓展推理能力并提高响应准确性 [10]。这种协同范式面临的一项关键挑战，是在不牺牲准确性的前提下保持实时性能，尤其是在高度动态的移动网络环境中。用户查询与生成响应之间的延迟可能导致输出慢得令人沮丧，进而损害应用的交互性和可靠性。在自回归生成中，这一问题尤为突出，因为逐词元传输会带来显著的通信延迟。此外，边缘端与云端之间交换的信息量通常相当大。因此，必须采用高效的推理加速方法，以降低通信开销和决策时延 [11]。

为缓解自回归生成的延迟，推测解码（SD）已被广泛采用，成为关键的推理加速技术 [12]。在典型的边云推测解码范式中，边缘设备快速生成一系列推测性的“草稿”词元，并将其传输至强大的云服务器进行并行验证。尽管近期基于深度学习的推测解码方法（如 EAGLE [13] 和 Medusa [14],）已展现出颇具前景的延迟降低效果，但其在实际边云系统中的部署仍面临若干根本性挑战。具体而言，现有推测解码框架依赖草稿模型与目标模型之间的紧密耦合，而云端目标模型又经常通过参数高效微调（PEFT） [11], 进行更新，由此导致“更新风暴”，其特点是：在带宽受限的无线链路上，为维持模型一致性而产生过高的通信开销。此外，通过保持边缘草稿模型静态来避免模型同步，会在目标模型演化过程中引发严重的分布偏移，导致令牌接受率急剧下降，这通常被称为“性能崩溃”。最后，推测解码性能对无线信道动态高度敏感，时变的上行时延可能会显著

Y. Li、R. Kong、Q. Li、X. Chen、H. Cai、L. Yan、S. Wang、H. Xiong 和 D. Yin 任职于中国北京的百度公司（电子邮箱： {yuchenli1230、monster119120、qiyangli878、fantastique0910、hengyi1995、lingyongy、shqiang.wang}@gmail.com；haoyi.xiong.fr@ieee.org；yindawei@acm.org）。Y. Li、L. Kong 和 G. Chen 任职于中国上海的上海交通大学计算机科学学院（电子邮箱： {yuchenli、linghe.kong、chen-gh}@sjtu.edu.cn）。Z. Lyu 任职于瑞典斯德哥尔摩的瑞典皇家理工学院信息科学与工程系（电子邮箱: lzhon@kth.se）。（通讯作者：*Z. Lyu*）J. Zhao 任职于威尔弗里德大学物理与计算机科学系（电子邮箱：jzhao@wlu.ca）。G. Zhu 任职于中国深圳的深圳市大数据研究院（电子邮箱：gxzhu@sribd.cn）。

削弱移动环境中固定步长推测执行所带来的延迟收益。综上所述，频繁的模型演化、草稿模型与目标模型之间的分布偏移，以及时变的无线延迟，从根本上限制了现有推测解码框架在实际边云部署中的适用性。

受这些挑战的启发，我们的目标是在边云系统中实现版本无关的推测解码，即利用单个静态边缘模型加速多种不断演化的云端目标模型。一旦构建起这一稳健的草稿框架，就能使用单个草稿模型加速多个微调版本，同时显著降低维护成本。此外，与必须从头开始重新训练的现有系统不同，该解耦框架能够很好地泛化至未见过的目标分布。因此，部署成本和通信成本都大幅降低。

具体而言，受基础模型（FM）在模态无关处理方面取得成功的启发，我们提出了**FlexSpec**，这是一种面向不断演化的大语言模型、通信高效的边云协同推测解码框架。FlexSpec 的核心思想是从根本上将边缘侧草稿模型的生命周期与云端目标模型的生命周期解耦，从而消除带宽受限的无线环境中频繁进行空中模型同步的需求。此外，FlexSpec 通过根据时变网络条件调整推测解码行为，明确考虑了无线信道的随机性。通过在统一框架中联合考虑模型演化和无线动态，FlexSpec 能够在边缘系统中实现可扩展、稳健且低延迟的大语言模型推理。本文的主要贡献总结如下：

- 我们提出了 FlexSpec，这是一种新颖的边缘端-云端协作推测解码框架，可为不断演进的大语言模型实现版本无关推理。通过在边缘草稿模型与云端目标模型之间引入共享冻结锚定骨干网络，FlexSpec 从根本上解耦了二者的生命周期，消除了频繁同步草稿模型的需求，从而有效解决无线边缘网络中的“更新风暴”问题。

- 我们针对边缘环境下的推测解码，设计了一种信道自适应草稿策略。通过对推测步长、令牌接受率和通信速率之间的相互作用进行建模，我们推导出具有理论依据的草稿长度选择方法，使其能够动态适应时变信道条件，并最大限度地提升有效令牌生成吞吐量。

- 我们在大规模 GPU 集群（最多配备 32 个 NVIDIA A800 GPU）上，针对多种推理和编程任务（例如 GSM8K 和 HumanEval）开展了大量实验。结果表明，FlexSpec 始终能够提升端到端生成效率，并在异构无线网络中保持稳健的性能。具体而言，FlexSpec 会根据时变信道条件自适应地调整推测步幅，在通信受限场景中，其性能优于最先进的推测解码方法。此外，FlexSpec 将冻结的边缘草稿模型与频繁更新的云端目标模型解耦，大幅避免了反复进行草稿模型同步，从而显著降低了与更新相关的通信开销。

本文其余部分安排如下。第二节介绍相关工作。第三节介绍边缘端-云端推测解码的系统模型与问题定义。第四节详述所提出 FlexSpec 的网络架构和自适应策略。第五节展示实验结果，以验证 FlexSpec 的性能。第六节总结全文。

II. 相关工作

本节从大语言模型加速的推测解码、边缘端-云端协同推理和无线感知移动人工智能等角度回顾并讨论相关工作。

A. 大语言模型加速的推测解码

推测解码已成为一种重要范式，可缓解大语言模型自回归生成固有的内存受限延迟瓶颈。推测解码的核心思路是采用“先草稿、后验证”策略，以丰富的并行计算能力换取更少的顺序内存访问。

在奠基性工作中，Leviathan 等人 [12] 和 Chen 等人 [15] 对推测执行框架进行了形式化定义，并证明小型草稿模型可以生成候选词元，再由目标模型并行验证，从而保证输出在数学上与目标模型完全一致。在此基础上，近期研究致力于提升草稿质量和接受率。Medusa [14]在冻结的原始模型上引入多个解码头，以同时预测多个未来词元，无需单独的草稿模型。EAGLE [13]和 EAGLE-2 [16] 则进一步利用逐层特征外推，在草稿层处理输入特征，生成准确度更高的上下文草稿。除基于模型的方法外，n 元语法匹配 [17] 和基于检索的增强 [18]等非模型草稿生成技术也利用统计模式或外部数据库提出候选词元，而无需大量计算。

然而，这些现有的 SD 框架都基于静态环境这一假设，即草稿模型与目标模型紧密耦合或在结构上完全相同。在边缘场景中，云端目标模型会通过 PEFT[19]频繁演进。若要维持 EAGLE 或 Medusa 等方法所施加的严格耦合，就需要通过无线链路频繁将边缘侧草稿模型与云端目标模型进行同步，从而产生难以承受的通信开销。FlexSpec 的独特之处在于从结构上解耦草稿模型与目标模型的生命周期，使单一静态边缘模型能够服务于不断演进的云端目标，而无需频繁同步。

B. 边云协同推理

边云协同推理弥合了移动设备有限的资源容量与现代深度神经网络（DNNs）庞大计算需求之间的差距 [20]。该领域通常研究如何在边缘端与云端之间优化划分计算任务，以平衡延迟、能耗和隐私。

早期工作（如 Neurosurgeon [21] 以及 Matsubara 等人探索的技术 [22], ）侧重于确定 DNN 中的最佳拆分点，以尽量降低端到端延迟。随着大语言模型的兴起，研究人员提出了去中心化推理框架（例如 [23]–[25]），通过互联网将 Transformer 模块分布到异构设备上。研究人员还提出了激进的量化技术 [26], [27] 和激活压缩方法[28] ，以缓解传输大型中间激活张量所造成的通信瓶颈。

尽管取得了这些进展，标准的拆分推理方法应用于生成任务时仍常常面临较高的传输延迟，因为每个令牌都传输高维激活会产生很大的通信开销。此外，这些方法在很大程度上忽视了生产环境中的版本一致性问题，而云端模型每天都会更新。FlexSpec 通过传输轻量级令牌索引而非庞大的激活，并引入一种基于锚点的对齐机制，使边云协同能够稳健应对模型版本不匹配问题，从而推动了该领域的发展。

C. 无线感知移动人工智能

在移动计算环境中部署人工智能应用，必须严格考虑无线网络动态。与稳定的有线连接不同，存在衰落和噪声的动态无线信道会显著影响实时人工智能应用的体验质量（QoE）。为解决此类问题，Park 等人 [29] 和 Zhu 等人 [30] 通过优化计算卸载与通信信道质量之间的权衡，为无线边缘智能奠定了基础。在自然语言处理（NLP）领域，DeeBERT [31] 和 SPINN [32]等动态推理机制可根据置信度阈值提前退出模型，从而节省能耗。近期，DSSD [33]尝试将推测解码应用于无线环境，但依赖固定策略，未能充分利用实时信道状态信息。

FlexSpec 填补了无线网络理论与大语言模型推理系统之间的空白。不同于以往将生成长度 K 视为固定超参数的研究，FlexSpec 将草稿步长建模为一个取决于信道条件的动态变量。通过将信道感知自适应应用于生成过程，FlexSpec 确保系统即使在移动场景特有的波动网络条件下，也能保持较高的吞吐量。

III. 系统模型与问题定义

在本节中，我们首先介绍边缘端-云端协同推理的系统模型。随后，基于使用真实世界数据集和网络轨迹开展的初步实验，我们确定了阻碍现有推测解码框架在移动环境中实际部署的三个关键研究挑战：模型同步所需的高昂带宽成本、分布偏移导致的性能崩溃，以及对无线网络波动性的敏感性。

A. 系统模型：边云协同推理

我们考虑一个由边缘设备 $\mathcal{E}$ 和云服务器 $\mathcal{C}$组成的协同推理系统。云端部署了一个高容量目标模型 $\mathcal{M}_t(\cdot;\theta_t)$（例如 Llama-2-70B），运行于 GPU 集群（例如 NVIDIA A800）上，其中 $\theta_t \in \mathbb{R}^{N_t}$表示其参数。为在不断变化的下游任务中保持竞争力，服务提供商会定期通过参数高效微调或全参数训练，基于不断演进的任务数据集 $\mathcal{D}_{\text{task}}$更新 $\mathcal{M}_t$ 。因此，云端模型会随时间演进，形成序列 $\{\mathcal{M}_t^{(0)}, \mathcal{M}_t^{(1)}, \ldots, \mathcal{M}_t^{(s)}\}$。

在边缘端，客户端部署了一个轻量级草稿模型$\mathcal{M}_d(\cdot;\theta_d)$，其具有 $\theta_d \in \mathbb{R}^{N_d}$ 和 $N_d \ll N_t$，模型规模通常仅为 $\mathcal{M}_t$ 的一小部分（例如，相当于目标模型的一层）。由于存储、电池和计算能力均受到严格限制，边缘设备无法频繁下载大型模型权重。因此，在目标模型更新期间，该模型保持静态（冻结），即 $\mathcal{M}_d^{(s)} = \mathcal{M}_d^{(0)}$ ， $\forall s$。

边缘端与云端通过随机无线链路进行通信。在每个推测解码步骤中，边缘端生成一段词元草稿 x_{draft} 并将其传输至云端，而云端使用 $\mathcal{M}_t$ 对该草稿块进行验证，并返回验证结果 x_{verified}。

给定输入提示 X，系统生成长度为 O的输出序列 $Y = \{y_1, \ldots, y_O\}$。在草稿长度为 K的推测解码中，一次解码步骤 n的延迟建模为

$$T_{\text{step}}(K, R_n) = T_{\text{edge}}(K) + T_{\text{up}}(K, R_n) + T_{\text{cloud}}(K) + T_{\text{down}}, \tag{1}$$

其中， $T_{\text{edge}}(K)$ 表示边缘端生成 K个词元草稿所需的计算时间， $T_{\text{up}}(K, R_n)$ 表示上行传输时延， $T_{\text{cloud}}(K)$ 表示云侧验证时间， T_{down} 表示下行链路传输验证结果/反馈的延迟。此外，$T_{\text{up}}(K, R_n)$ 进一步表示为上行传输开销 $B_{\text{up}}(K)$ 与步骤 n中的可实现上行速率 R_n 之比，即 $T_{\text{up}}(K, R_n) = B_{\text{up}}(K)/R_n$。

B. 挑战一：无线网络中的“更新风暴”

首要障碍是模型新鲜度需求与有限无线带宽之间的冲突。在标准推测解码范式中，草稿模型 $\mathcal{M}_d$ 必须近似目标模型 $\mathcal{M}_t$的分布。因此，每当 $\mathcal{M}_t$ 更新到新版本$\mathcal{M}_t'$（例如，针对医疗应用进行微调）时，为了保持较高的词元接受率 $\hat{}\gamma$，边缘端的 $\mathcal{M}_d$ 理论上都必须重新训练并同步。

我们将这种现象定义为“更新风暴”。为量化其影响，我们初步分析了同步轻量级

表I 无线网络中同步草稿模型的估计延迟。频繁更新使标准推测解码难以实施。

网络类型	带宽	同步时间（单个用户）	可扩展性（1k 个用户）
公共 WiFi	10 Mbps	≈ 48 分钟	**崩溃**
4G LTE	50 Mbps	≈ 9.5 分钟	严重拥塞
5G中频段	300 Mbps	≈ 1.6 分钟	中等负载

草稿模型（约 3.2 GB）在典型移动网络中所需的传输开销。

如表I所示，即使下载经过压缩的草稿模型，在4G 网络上也需要近10分钟。对于每天都要更新模型的移动应用，这会给蜂窝基础设施带来数PB的流量，并迅速耗尽用户的数据套餐和电池电量。这促使我们设计一种无需同步的静态 草稿模型。

C. 挑战 *2*：分布偏移与性能崩溃

如果我们放弃模型同步以节省带宽（即保持 $\mathcal{M}_d$ 固定，同时 $\mathcal{M}_t$ 不断演化），就会遇到第二个挑战：分布偏移。静态草稿模型的输出分布 P_d 与不断演化的目标模型的分布 P_t' 之间的差异，在微调后显著增大。

为验证这一点，我们使用通用的 Llama-2-7B 作为冻结的草稿模型进行了初步实验。我们测量了它相对于 70B 目标模型三个版本的令牌接受率：基础版本、数学调优版本（GSM8K）和代码调优版本（HumanEval）。

表 II 目标模型演化对固定草稿模型性能的影响。缺乏对齐时，接受率急剧下降。

目标模型版本	领域	接受率
Llama-2-70B-Base	通用	**0.72**
Llama-2-70B-Math（LoRA）	数学	0.45（↓ 37.5%）
Llama-2-70B-Code（完整）	编程	**0.18（↓ 75.0%）**

表 II 中的结果揭示了严重的“性能崩塌”。当目标模型演化到编程领域时，通用草稿模型的接受率骤降至 0.18。在如此低的接受率下，推测解码的开销（运行草稿模型并传输词元）超过了其收益，导致其延迟往往高于标准自回归解码。这就需要采用 FlexSpec 的基于锚点的对齐等机制，以弥合静态表示与演化表示之间的差距。

D. 挑战 *3*：无线延迟敏感性

第三个挑战在于推测长度 K 与网络条件之间的耦合。大多数现有工作采用固定的推测步幅 K （例如 $K = 5$）。然而，我们的网络轨迹分析表明，在移动环境中，固定的 K 可能并非最优。在弱信号场景下（例如信噪比（SNR）为 < 5 dB 的电梯或地铁中），上行传输项 $T_{\text{up}}(K,\ R_n)$ 在式（1）中占据端到端延迟的主导地位。例如，传输五个词元可能产生约 200 毫秒的上行延迟，而相应的验证增益仅约为 50 毫秒。相反，在强 5G 信道条件下，较小的固定K 无法充分利用可用带宽。这种不匹配造成了一个根本性困境：静态推测策略无法同时满足异构移动性和信道条件下的延迟要求，因此凸显了本文提出的信道感知自适应推测机制的必要性。

E. 问题定义

基于 (1)，我们的目标是设计一种在演进中的边云系统中仍能有效运行的边云推测解码框架。具体而言，给定一系列随时间更新的云侧目标模型 $\{\mathcal{M}_t^{(s)}\}$，我们寻求一种模型耦合机制，使边缘草稿模型 $\mathcal{M}_d$ 保持静态 （即无需边缘端重新训练或反复下载模型），同时在与每个 $\mathcal{M}_t^{(s)}$ 协作时仍能保持较高的令牌接受率。

同时，在时变无线条件下，我们旨在为 K 寻找一种策略，并为 $\mathcal{M}_d$ 与 $\mathcal{M}_t$ 设计一种模型耦合机制，以最大化有效令牌生成速率（ETGR），即单位时间内预期接受的词元数量。设 τ 表示在草稿长度为 K的一轮草稿-验证中接受的词元数量，则 ETGR 可表示为：长度为K的推测性草稿所产生的预期接受词元数 τ （即 $\mathbb{E}[\tau \mid K]$）与式 (1) 中每轮延迟之比，即

$$\mathrm{ETGR}(K) \triangleq \frac{\mathbb{E}[\tau \mid K]}{\mathbb{E}[T_{\text{step}}(K, R_n)]}, \tag{2}$$

这等价于最小化每个已接受令牌的预期延迟，即 $\mathbb{E}[T_{\text{step}}(K)]/\mathbb{E}[\tau \mid K]$。总之，我们旨在学习一种关于 K 的自适应策略，以及一种连接 $\mathcal{M}_d$ 和 $\mathcal{M}_t$ 的模型耦合机制，以最大化 ETGR。

IV. 方法：FLEXSPEC

本节详细介绍 FlexSpec 框架，以应对上述挑战。我们的方法由三个相互集成的组件构成：(1) 一种结构解耦架构，使静态边缘草稿模型能够服务于不断演进的云端目标模型；(2) 一种信道感知推测策略，根据实时信道条件优化草稿长度 K ；以及 (3) 一种语义同步协议，用于最大限度地降低传输开销。

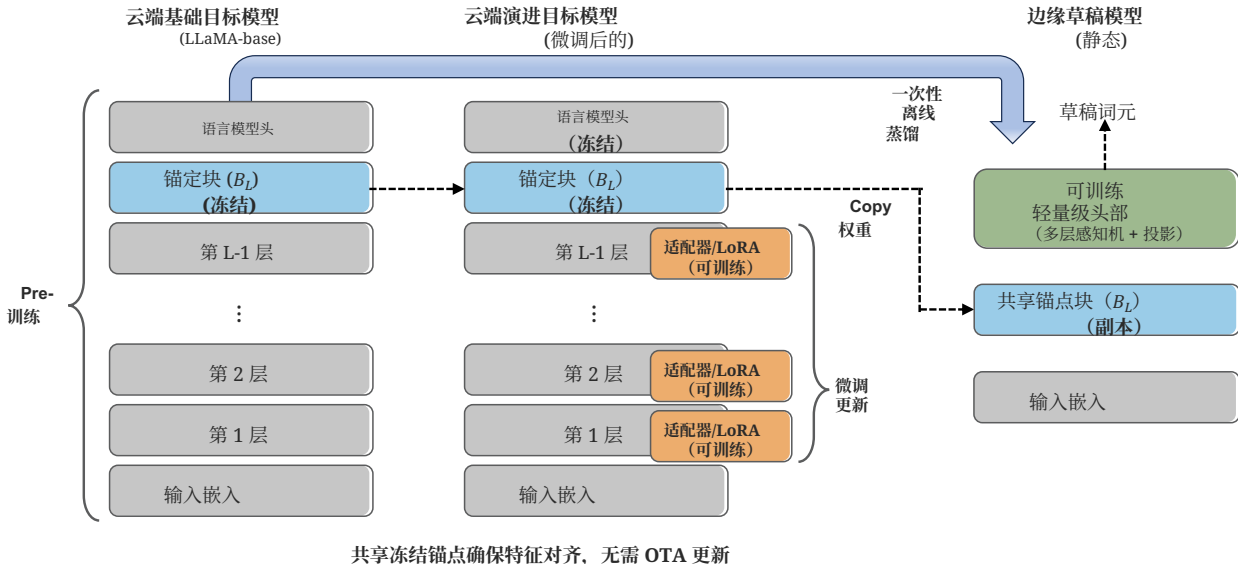


图 1. FlexSpec 训练架构：共享冻结锚点块(B_L)确保静态边缘端草稿模型与不断演进的云端目标模型之间的特征空间对齐。轻量级头部通过一次性离线蒸馏进行训练。

A. 架构：基于锚点的特征对齐 *ent*

为满足 θ_d 保持静态而 θ_t不断演进这一约束，我们提出了基于锚点的对齐架构。由于云端微调导致分布$P_d($ $y|x)$与 $P_t(y|x)$出现偏差，标准推测解码会失效。我们假设，尽管微调过程中高层语义指令会发生变化，但骨干网络 [34]中的基础词元表示逻辑仍保持稳定。

1) 结构定义：我们将 $\mathcal{M}_{\text{base}} \triangleq \mathcal{M}_t^{(0)}$ 记作针对特定任务进行微调之前的云端目标模型。按照图 1 所示的架构，我们将 $\mathcal{M}_{\text{base}}$ 分解为一个特征提取器和一个头部：

$$\mathcal{M}_{\text{base}}(x) = \Psi_{\text{base}}\big(\Phi_{\text{base}}(x)\big), \tag{3}$$

其中， Φ_{base} 由底部 Transformer 层堆栈（第 1 层至第 $L-1$ 层）以及输入嵌入模块构成，而 Ψ_{base} 表示头部，其中包含最后一个 Transformer 块（第 L 层）和语言模型头。我们将 Ψ_{base} 内部的最后一个 Transformer 块称为锚定块，记作 $\mathcal{B}_L^{\text{base}}$。

基于这一分解，边缘端草稿模型 $\mathcal{M}_d$ 通过以下方式构建：(i) 从基础头部复制锚定块并将其保持冻结；(ii) 添加一个用于词元预测的轻量级可训练头部。具体而言，

$$\mathcal{M}_d(x) = \mathcal{H}_{\text{small}}\big(\mathcal{B}_{\text{shared}}(x)\big), \tag{4}$$

其中， $\mathcal{B}_{\text{shared}} \triangleq \mathcal{B}_L^{\text{base}}$是锚定块的冻结副本， $\mathcal{H}_{\text{small}}$ 是一个轻量级可训练投影头，由一个两层多层感知机（MLP）和一个词表投影矩阵组成。通过这种方式， $\mathcal{M}_d$ 复用了基础头所诱导的表示空间，同时保持足够紧凑，适合边缘部署。

在云端微调期间，我们采用 PEFT 风格的参数化方式，将不断演进的目标模型 $\mathcal{M}_t(\cdot;\theta_t)$ 的参数分解为 $\theta_t = \{\theta_t^{\text{backbone}}, \theta_t^{\text{adapters}}\}$。其中，$\theta_t^{\text{backbone}}$表示 $\mathcal{M}_{\text{base}}$ 的原始预训练权重（即，Φ_{base} 和 Ψ_{base}的参数，包括锚定块和语言模型头），而 $\theta_t^{\text{adapters}}$ 表示注入 Transformer 层的额外参数高效微调参数（例如 LoRA/适配器模块）。我们通过冻结 $\theta_t^{\text{backbone}}$并仅在微调期间更新 $\theta_t^{\text{adapters}}$ 来施加骨干冻结约束。因此，头部 Ψ_{base} （层 L和语言模型头）保持不变，从而稳定锚定块所见的特征流形，并保留静态边缘草稿 $\mathcal{M}_d$ 准确起草所依赖的特征兼容性。

2）通用模型训练目标：为了训练静态边缘组件（$\mathcal{H}_{small}$），我们采用结合特征回归和知识蒸馏（KD）的多目标损失函数 [35]。训练在通用语料库（例如 RedPajama [36]）上使用基础模型 $\mathcal{M}_{base}$ 作为教师模型，**仅进行一次**。具体而言，损失函数如下所示。

- **特征回归损失（$\mathcal{L}_{feat}$）：** $\mathcal{L}_{feat}$ 确保草稿隐藏状态 h_d 与目标隐藏状态h_t对齐，即：

$$\mathcal{L}_{feat} = \frac{1}{\omega \cdot \pi} \sum_{i=1}^{\omega} \sum_{j=1}^{\pi} \| W_p \cdot h_d^{(i,j)} - h_t^{(i,j)} \|_2^2, \tag{5}$$

其中， W_p 是用于匹配维度的可学习投影矩阵 [37], ， ω 是批次大小， π 是序列长度。

- **软目标知识蒸馏损失 ($\mathcal{L}_{KD}$)：** 它最小化词元分布之间的 KL 散度 [38], ，即，

$$\mathcal{L}_{KD} = \mathcal{T}^2 D_{KL}\left(\sigma\left(\frac{z_t}{\mathcal{T}}\right) \middle\| \sigma\left(\frac{z_d}{\mathcal{T}}\right)\right), \tag{6}$$

其中 z_t， z_d 表示对数几率， σ 表示 Softmax 激活函数， $\mathcal{T}$表示温度系数。

因此，总目标是式（5）和式（6）的组合，即 $\mathcal{L} = \lambda_1 \mathcal{L}_{feat} + \lambda_2 \mathcal{L}_{KD}$。算法 1 总结了这一离线训练过程。

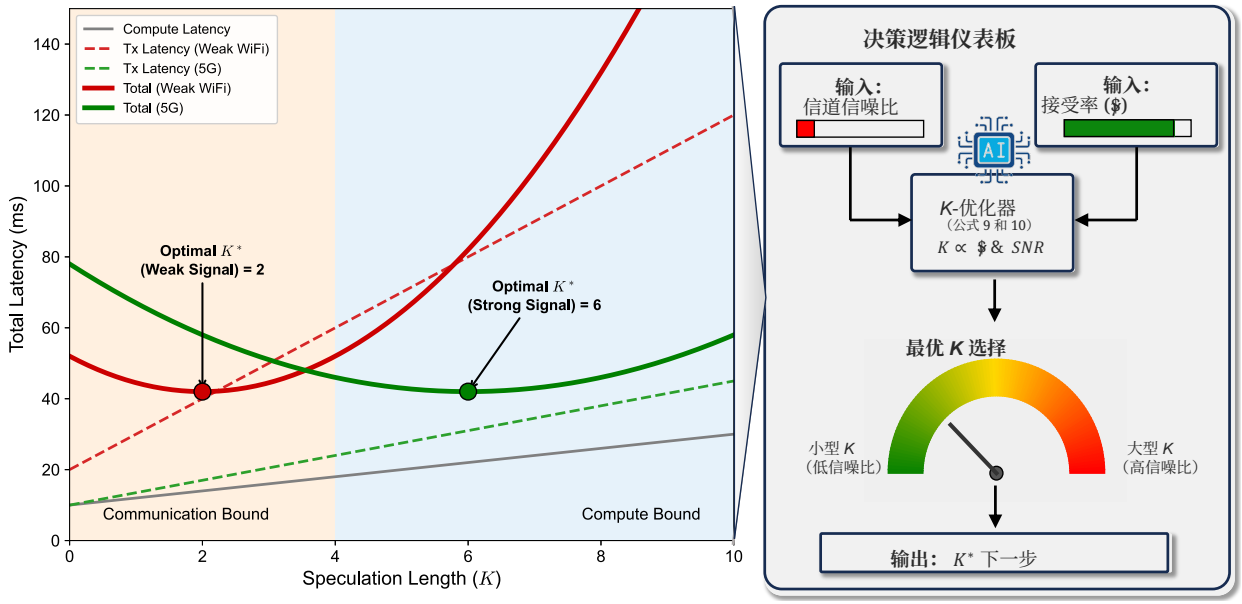


图 2. 通道感知的自适应推测 机制：左图展示了通信与计算延迟之间的权衡 -，以及不同信号强度下的情况。最优草稿 nt 信号强度。最佳草稿长度 K^* 从 2（弱信号）变化到 6（强信号）。

算法 1： FlexSpec 离线草稿模型训练

输入： 基础目标模型 $\mathcal{M}_{\text{base}}$，数据集 $\mathcal{D}_{\text{pretrain}}$
输出： 静态草稿模型 $\mathcal{M}_d = \{\mathcal{B}_{\text{shared}}, \mathcal{H}_{\text{small}}\}$

// 步骤 1：初始化和结构冻结
随机初始化轻量级头部 $\mathcal{H}_{\text{small}}$ ；
// 复制锚定块
$\mathcal{B}_{\text{shared}} \leftarrow \mathcal{M}_{\text{base}}$.块$[-1]$;
冻结 $\mathcal{B}_{\text{shared}}$ 和 $\mathcal{M}_{\text{base}}$的参数;
// 步骤 2：对齐训练循环
对每个批次(x, y) 于 $\mathcal{D}_{pretrain}$ **执行**
  // 教师前向传播（基础模型）
  $h_t,\ z_t \leftarrow \mathcal{M}_{\text{base}}(x)$;
  // 学生前向传播（草稿模型）
  $h_d,\ z_d \leftarrow \mathcal{H}_{\text{small}}(\mathcal{B}_{\text{shared}}(x))$;
  // 多目标损失计算
  ...;
  $\mathcal{L}_{\text{total}} \leftarrow \lambda_1 \mathcal{L}_{\text{feat}} + \lambda_2 \mathcal{L}_{\text{KD}}$;
  通过优化器$(\nabla \mathcal{L}_{\text{total}})$更新 $\mathcal{H}_{\text{small}}$ ；
返回 $\mathcal{M}_d$

B. 信道感知自适应推测

在移动计算环境中，端到端延迟与信道动态紧密耦合。静态草稿长度K 无法适应传输开销与推测增益之间的权衡。为此，我们将最优 K^* 选择建模为吞吐量最大化问题，并明确考虑传播时延和可变的云端验证成本。

1) 精细化时延建模：在式（1）中，步幅为 K 的单个推测解码步骤的总耗时记为 $T_{\text{step}}(K,\ R_n)$，由四部分组成：边缘侧计算、上行传输、云侧验证和下行反馈。不同于以往假设云端验证成本恒定的研究，我们将云侧时延建模为 K 的仿射函数，以反映加载 K 个新生成词元及其相关键值（KV）缓存所产生的内存带宽开销。因此，精细化时延模型如下：

$$\hat{T}_{\text{step}}(K, R_n) = \hat{T}_{\text{edge}}(K) + \hat{T}_{\text{up}}(K, R_n) + \hat{T}_{\text{cloud}}(K) + T_{\text{down}}. \tag{7}$$

上行传输时延明确计入传播时延 T_{prop}（等于往返时间的一半），以及由通信开销 $B_{\text{up}}(K) \triangleq K \cdot b + O_{\text{header}}$ 和瞬时通信速率 R_n决定的数据传输时延，即：

$$\hat{T}_{\text{up}}(K, R_n) = T_{\text{prop}} + \frac{K \cdot b + O_{\text{header}}}{R_n}, \tag{8}$$

其中，每个词元都编码为 b 位的整数索引，O_{header} 表示协议相关传输开销（例如数据包头）。此外，云端验证时延建模为基础处理成本加上每个词元的边际计算成本：

$$\hat{T}_{\text{cloud}}(K) = T_{\text{base}} + K \cdot \delta_{\text{cloud}}, \tag{9}$$

其中，δ_{cloud} 表示验证一个额外 token 所需的额外计算时间。

为便于在资源受限的边缘设备上进行高效的在线优化，我们进一步将延迟项分为固定开销和边际成本。此外，我们采用线性近似来估算边缘计算时延，

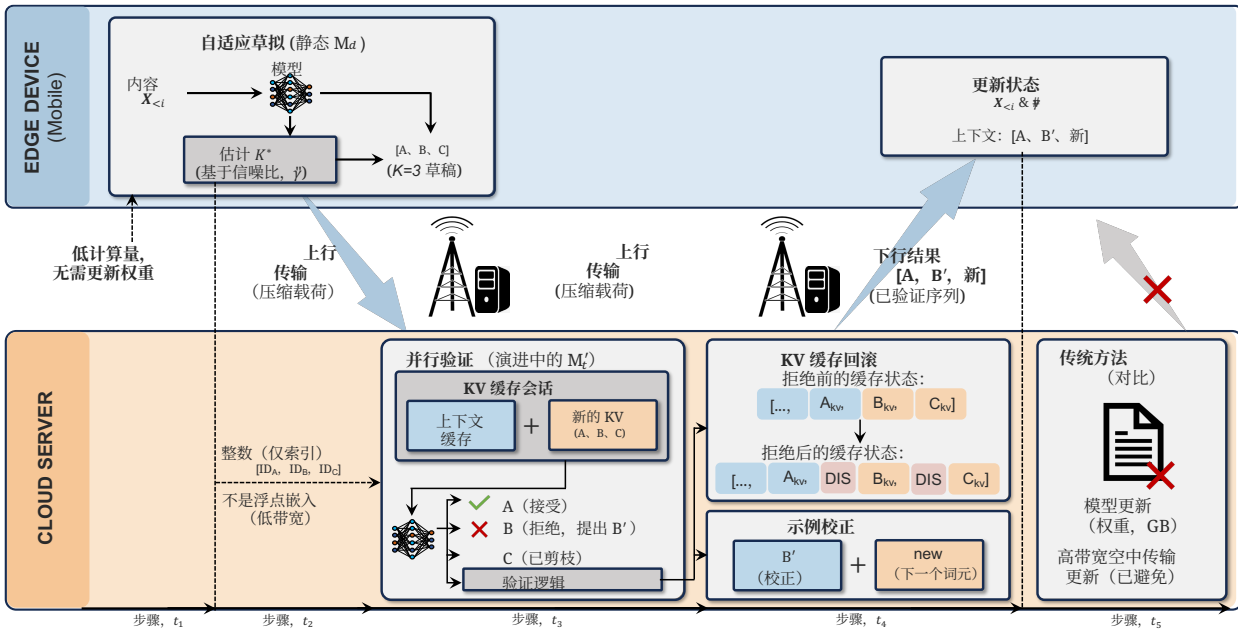


图 3. FlexSpec 无线协同推理流程：该流程包括边缘端的自适应草拟（上方）、压缩后的上行传输、云端的并行验证与 KVCacherollback（下方），以及已验证结果的下行传输。

$\hat{T}_{\text{edge}}(K) \approx \alpha_{\text{edge}}K + \beta$，这种方法在实践中被广泛采用。将上述表达式代入可得

$$\hat{T}_{\text{step}}(K, R_n) = T_{\text{fixed}} + K \cdot \underbrace{\left(\alpha_{\text{edge}} + \frac{b}{R_n} + \delta_{\text{cloud}}\right)}_{T_{\text{marginal}}(n)}, \quad (10)$$

其中，$T_{\text{fixed}} = T_{\text{prop}} + T_{\text{base}} + T_{\text{down}} + \frac{O_{\text{header}}}{R_n} + \beta$，而 $T_{\text{marginal}}(n)$ 则表示在信道条件随时间变化时，每增加一个推测词元所产生的总边际延迟。

2) 吞吐量最优策略： FlexSpec 的目标是最大化式 (2) 中的 ETGR。基于式 (10) 中精细化的时延模型，我们根据瞬时信道条件对式 (2) 进行可处理的近似，并通过求解来计算推测步长

$$K_n^* = \underset{K \in [1, K_{\max}]}{\arg\max} \frac{\mathbb{E}[\tau \mid K]}{T_{\text{fixed}} + K \cdot T_{\text{marginal}}(n)}, \quad (11)$$

其中， $K_{\max}$ 表示最大草稿长度。式 (11) 刻画了令牌接受增益与端到端时延成本之间的基本权衡。

图 2 展示了由此得到的优化景观。如图所示，总延迟随无线链路质量而显著变化。在弱信号区域，通信成本占主导地位，式 (11) 中的分母会随 K 增大而迅速增长，从而使最优解 K_n^* 转向较小的值。相反，在信道条件良好时，较大的 K 更有利，因为传输开销可以分摊到更多推测词元上。

为实现实用的在线自适应，我们采用几何衰减模型或指数移动平均（EMA）来近似预期接受率 $\mathbb{E}[\tau \mid K]$，从而得到 $\mathbb{E}[\tau \mid K] \approx \hat{\gamma} \cdot K$，适用于 K 取中等值的情况。在这一近似下，目标函数 (11) 具有明确的物理含义。具体而言，较大的传播时延 T_{prop} （即较大的 T_{fixed}）会促使采用更大的推测步幅，以摊销往返开销；而通信速率 R_n 降低会增大 T_{marginal}（n），因此需要采用更小的 K_n^* 以避免拥塞。

C. FlexSpec 协议与推理流程

边缘端与云端之间的运行时交互在草稿模型版本方面采用无状态设计，同时在键值（KV）缓存方面保持有状态。完整的推理过程总结于算法 2，并在图 3 中展示。为避免云端在整个前缀历史上重复计算注意力，FlexSpec 为每位用户维护一个持久化的 KV 缓存会话。如图 3 的步骤 t_3 和 t_4 所示，当边缘端传输 K 推测性草稿词元时，云端仅针对这些新接收的词元计算相应的键值对。

如果在索引 $j < K$处发生拒绝，云端会执行*KV* 缓存回滚，在处理下一个请求前丢弃从索引j 开始的所有无效键值对。该机制确保云端计算成本随验证长度而非完整上下文长度增长，从而保留推测解码带来的效率提升。

随着云侧目标模型 $\mathcal{M}_t'$ 不断演进，分布差异 $D_{\text{KL}}(P_d \| P_t')$ 可能会随时间增大。FlexSpec 能够通过轻量级回退机制妥善应对这类分布偏移，其理念与 SpecInfer [39], 采用的方法类似，从而在无需同步草稿模型的情况下，保持稳健的推理性能。

算法 2： FlexSpec 协同推理

输入： 静态边缘端模型 $\mathcal{M}_d$，演进中的云端模型 $\mathcal{M}_t'$，上下文$\mathbf{x}_{<i}$，衰减率 μ
输出： 生成的词元序列

初始化指数移动平均接受率 $\hat{\gamma} \leftarrow 0.8$;
当尚未生成序列结束标记 **时执行**
  // 步骤 1：边缘侧自适应草拟草拟
  测量信道条件;
  // 计算边际延迟和最优推测步幅
  $T_{\text{marginal}}(n) \leftarrow \alpha_{\text{edge}} + b/R_n + \delta_{\text{cloud}}$;
  $K_n^* \leftarrow \arg\max_K \frac{1+\gamma \cdot K \quad \hat{}(n)}{T_{\text{fixed}}+K \cdot T_{\text{marginal}}}$;
  $\mathbf{x}_{\text{draft}} \leftarrow ()$;
  对于 $k = 1, \ldots, K_n^*$ **do**
    $\hat{x} \sim \mathcal{M}_d(\mathbf{x}_{<i} \cdot \mathbf{x}_{\text{draft}})$;
    $\mathbf{x}_{\text{draft}} \leftarrow \mathbf{x}_{\text{draft}} \cdot \hat{x}$;
  通过上行链路向云端传输压缩后的$(\mathbf{x}_{\text{draft}})$;
  // 步骤 2：云端并行验证
  接收$\mathbf{x}_{\text{draft}}$ 并恢复 KV 缓存;
  $\mathbf{q} \leftarrow \mathcal{M}_t'(\mathbf{x}_{\text{draft}}$,上下文 $= \mathbf{x}_{<i})$;
  $\tau \leftarrow 0$;
  对于 $k = 1, \ldots, K_n^*$ **do**
    如果 $x^{(k)}$ 草稿 $== \arg\max \mathbf{q}^{(k)}$ **则**
      $\tau \leftarrow \tau + 1$;
    else
      中断;
  采样校正令牌 x_{new} 来自 $\mathcal{M}_t'$;
  传输$\mathbf{x}_{\text{verified}} \leftarrow \mathbf{x}_{\text{draft}}^{1:\tau} \cdot x_{\text{new}}$ 至边缘端;
  // 步骤 3：状态更新
  $\mathbf{x}_{<i} \leftarrow \mathbf{x}_{<i} \cdot \mathbf{x}_{\text{verified}}$;
  $\hat{\gamma} \leftarrow (1-\mu)\hat{\gamma} + \mu(\tau/K_n^*)$;

V. 评估

为全面评估 FlexSpec，我们进行了大量实验，以回答以下五个关键研究问题：

- **RQ1（性能与鲁棒性）：** 在不同网络条件和不同采样温度下，FlexSpec 与最先进的基线方法相比表现如何?
- **研究问题 2（消融研究）：** 信道感知自适应推测机制是否必不可少？在不同信道质量下，它与固定步长策略相比表现如何?
- **RQ3（硬件通用性）：** FlexSpec 能否适应不同移动设备异构的计算能力，并应对不同任务复杂度?
- **RQ4（模型可扩展性）：** 该架构能否扩展至更新的大语言模型系列（Llama-3、Mistral）和稀疏架构（混合专家）？
- **RQ5（效率）：** 在内存占用、能耗细分和热效率方面有哪些切实优势?

A. 实验设置

扩展后的硬件测试平台： 我们大幅扩展了硬件设备的多样性，以模拟真实的跨设备部署场景。云服务器： 8× NVIDIA H800（80GB），通过 NVLink 连接； 8× NVIDIA A800（80GB）（主流配置）； 8×NVIDIA V100（32GB）。边缘设备（移动客户端）：NVIDIA Jetson AGX Orin（64GB 内存）、Snapdragon 8 Gen 3 参考设备（16GB 内存）、Apple iPhone 15 Pro Max 模拟器（A17 Pro）、树莓派 5（8GB 内存）。

模型与数据集： 我们使用涵盖多种目标模型和下游任务的丰富套件，在分布偏移情况下评估 FlexSpec。对于目标模型，我们以 LLaMA-2 70B 作为代表性的稠密基线进行详细分析，并进一步评估其在较新架构上的可扩展性，包括 LLaMA-3 70B 和混合专家模型 Mixtral 8×7B。为了全面考察任务层面的泛化能力，我们选取了六项核心下游任务，涵盖不同的推理模式和数据分布，包括用于数学推理的 GSM8K 数据集、用于问答的 Natural Questions、用于检索增强生成的 Natural Questions、用于多轮对话的 MT-Bench、用于机器翻译的 WMT14 德英翻译，以及用于文档摘要的 CNN/DailyMail。

基线方法： 我们将 FlexSpec 与一组扩展的基线方法进行了比较，这些方法涵盖多种解码范式，从标准自回归方法到最先进的推测解码框架。

- **仅云端：** 完全在云服务器上执行的标准自回归解码。它作为吞吐量和延迟的基线：每次生成词元都需要完整的网络往返时间，并产生云端计算成本。
- **标准推测解码（朴素）：** 一种传统的推测解码设置，其中通用的预训练小模型（例如 Llama-2-7B）充当目标模型（例如 Llama-2-70B）的草稿模型。重要的是，该基线不采用我们的基于锚点的对齐，因此体现了草稿模型未能与不断演进的目标模型保持同步时，分布偏移造成的性能下降。
- **PLD（n 元语法）：** 提示查找解码是一种基于检索的方法，它利用字符串匹配，在当前上下文窗口中识别常见的 n 元语法模式，以草拟后续词元。这代表了一种轻量、无需训练且内存高效的基线。
- **Lookahead [40]:** 一种基于雅可比迭代的并行解码算法，无需单独的草稿模型即可同时生成多个词元。它依靠目标模型自身的能力，通过多步验证并行细化多个候选序列。
- **EAGLE-2（同步） [16]:** 这是一种采用逐层特征外推的最先进基于模型的方法。我们在“理想同步”设置下对其进行评估，假设边缘侧扩展层已完美更新，与云端目标版本保持一致，并忽略更新所产生的通信开销。
- **Medusa-1（同步） [14]:** 一种并行解码框架

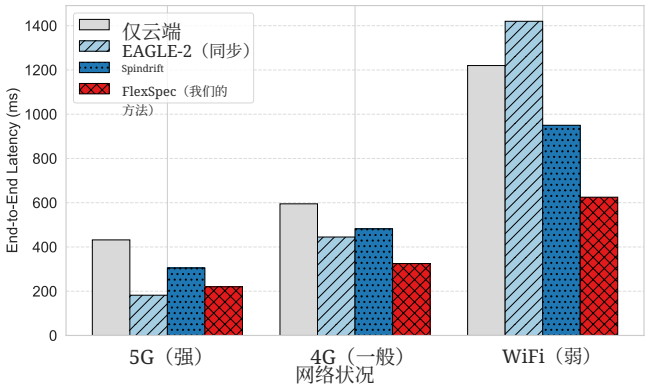


图 4. GSM8K 上的端到端延迟对比。与仅云端和最先进水平基线方法（EAGLE- 2、DSSD）相比，FlexSpec 实现了显著的加速，尤其是在带宽受限环境（WiFi）中，因为它能有效降低通信开销。

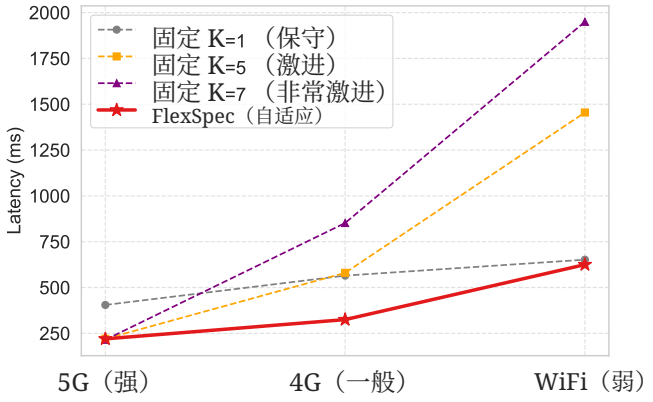


图 5. 推测步幅（K）对延迟的影响。固定步幅无法适应变化的信道条件（例如，$K = 5$ 会导致 WiFi 超时）。FlexSpec 的自适应机制会动态选择合适的推测步幅，从而获得更优性能。

该框架通过为目标模型添加多个解码头来预测未来的词元。与 EAGLE-2 类似，我们假设 Medusa 头与目标模型完美同步，以表示紧耦合架构的理论上限。

- **DSSD [33]：** 一种专为无线边缘云系统设计的协同推理框架。与 FlexSpec 不同，它采用固定或基于启发式的推测长度调度策略，无法根据不断变化的信道条件进行实时调整。

B. RQ1：端到端延迟与网络鲁棒性

为进行细致分析，我们将评估划分为两种不同模式，即确定性生成（温度系数 = 0）和随机采样（温度系数 = 1）。我们展示了 Llama-2 70B 模型在全部六个数据集上的详细结果，以证明其在不同任务类型上的稳健性。

1) 模式 *A*：确定性生成（温度系数 = *0*）：图 4 展示了 GSM8K 上的性能差距，表 III 则列出了采用贪心解码时的详细数值结果。此设置有利于依赖严格分布匹配的方法。

如表 III 所示，FlexSpec 在重推理任务（GSM8K）和抽取式任务（CNN/DM）上均实现了稳定的加速。值得注意的是，在标准推测解码表现不佳的 GSM8K 上（由于草稿质量较差，在 WiFi 下仅为 0.66×），FlexSpec 仍实现了 1.95× 的加速比，验证了我们基于锚点的对齐方法的有效性。虽然 EAGLE-2 等同步方法在 5G 下表现最佳（最高达 2.47×），但由于传输开销，它们在 WiFi 环境下会崩溃（$< 0.9\times$）。FlexSpec 则通过根据网络状态调整草稿长度 K，成为弱网络下唯一可行的加速方案。

2）情形*B*：随机采样（温度系数 = *1*）：表 IV 使用 Top-p 采样（$p = 0.9$，T =1）评估性能。这对静态草稿模型适应不断变化的云端目标的对齐能力提出了考验，因为草稿分布必须覆盖目标分布的概率质量。

即使在随机采样下，FlexSpec 在全部六项任务中仍保持较高的加速比（1.65×-1.90×）。相比之下，EAGLE-2 等方法会严重退化，因为其草稿生成与贪心路径紧密耦合；在 Temperature= 1 下，其性能会大幅下降（5G 场景中，从 Temperature = 0 下的 2.4× 降至 Temperature = 1 下的 1.6×）。

如表 IV 所示，标准 *SD*呈现了我们在研究动机中预测的“性能崩溃”现象。尤其是在弱网络下的 GSM8K 等专门领域中，标准 *SD* 会导致显著减速（0.65×），因为未对齐的草稿模型无法匹配目标模型的随机分布，从而频繁遭到拒绝。同样，在这种情况下，*Lookahead*带来的收益微乎其微（$< 1.06\times$），因为其 n 元语法匹配能力受到随机词元选择的严重限制。

对于 MT-Bench 和 CNN/DM 这类开放式任务，FlexSpec 的基于锚点的对齐可确保静态边缘草稿模型生成的词元在统计上与目标模型的概率质量对齐，从而避免标准推测解码中出现的性能下降，且无需进行模型再训练。

C. 研究问题 *2*：信道感知自适应的消融研究

为验证信道感知自适应推测机制的必要性，我们进行了消融研究：在保留基于锚点的对齐的同时，将动态 K 调整替换为固定推测步长（$K \in \{1, 3, 5, 7\}$）。实验在 GSM8K 数据集上，针对三种具有代表性的网络环境开展。

图 5 中的结果展示了所提出的自适应策略在异构网络条件下的有效性。在高带宽 5G 条件下，较大的固定推测步幅（例如 $K = 5$）能够实现较低延迟（220.0 ms），但在弱 WiFi 场景下表现不佳（1455.0 ms），由于传输延迟过高，相较于基线产生了 2.3× 的减速。相反，较为保守的步幅（例如 $K = 1$）在弱

表 III工作 A（温度 = 0）： LLAMA-2 70B 在全部 6 个数据集上的端到端延迟和加速比。FlexSpec 在多种任务的带宽受限环境中均展现出卓越性能。

数据集	网络	仅云端	Lookahead	标准 SD	Medusa-1	EAGLE-2	DSSD	FlexSpec
是否需要同步?		No	No	No	Yes	Yes	No	**No**
GSM8K (数学)	5G（强）	432.0毫秒（1.0×）	415.0毫秒（1.04×）	392.0毫秒（1.10×）	205.0毫秒（2.10×）	**182.0毫秒（2.37×）**	305.0毫秒（1.41×）	220.0ms (1.96×)
	4G（平均）	595.0ms (1.0×)	580.0ms (1.02×)	805.0ms (0.74×)	480.0ms (1.24×)	445.0ms (1.33×)	482.0ms (1.23×)	**325.0ms (1.83×)**
	WiFi（弱）	1220.0毫秒（1.0×）	1205.0 毫秒 (1.01×)	1850.0毫秒 (0.66×)	1455.0毫秒 (0.84×)	1420.0毫秒 (0.86×)	950.0毫秒 (1.28×)	**625.0毫秒 (1.95×)**
Natural Questions（问答）	5G（强）	415.0毫秒（1.0×）	390.0毫秒（1.06×）	345.0毫秒（1.20×）	192.0毫秒（2.16×）	**168.0毫秒（2.47×）**	275.0毫秒 (1.51×)	202.0毫秒 (2.05×)
	4G（平均）	570.0毫秒 (1.0×)	552.0毫秒 (1.03×)	685.0毫秒 (0.83×)	435.0毫秒 (1.31×)	405.0毫秒 (1.41×)	452.0毫秒 (1.26×)	**295.0毫秒 (1.93×)**
	WiFi（弱）	1185.0毫秒 (1.0×)	1165.0毫秒 (1.02×)	1680.0毫秒 (0.71×)	1380.0毫秒 (0.86×)	1352.0毫秒 (0.88×)	905.0毫秒 (1.31×)	**582.0毫秒 (2.04×)**
Natural Questions（检索增强生成）	5G（强）	428.0毫秒（1.0×）	402.0毫秒（1.06×）	360.0毫秒（1.19×）	198.0毫秒（2.16×）	**175.0毫秒 (2.44×)**	288.0毫秒 (1.49×)	212.0毫秒 (2.02×)
	4G（平均）	582.0毫秒 (1.0×)	568.0毫秒 (1.02×)	735.0毫秒 (0.79×)	455.0毫秒 (1.28×)	422.0毫秒 (1.38×)	468.0毫秒 (1.24×)	**308.0毫秒 (1.89×)**
	WiFi（弱）	1208.0 毫秒 (1.0×)	1190.0 毫秒 (1.02×)	1765.0 毫秒 (0.68×)	1425.0 毫秒 (0.85×)	1390.0 毫秒 (0.87×)	932.0 毫秒 (1.30×)	**600.0ms (2.01×)**
MT-Bench (聊天)	5G (强)	420.0ms (1.0×)	398.0ms (1.05×)	358.0ms (1.17×)	202.0毫秒（2.08×）	**178.0毫秒（2.36×）**	295.0毫秒（1.42×）	215.0毫秒（1.95×）
	4G（平均）	578.0毫秒（1.0×）	562.0毫秒 (1.03×)	742.0毫秒 (0.78×)	462.0毫秒 (1.25×)	430.0毫秒 (1.34×)	475.0毫秒 (1.22×)	**312.0毫秒 (1.85×)**
	WiFi（弱）	1192.0毫秒（1.0×）	1175.0毫秒（1.01×）	1780.0毫秒（0.67×）	1440.0毫秒（0.83×）	1405.0毫秒（0.85×）	945.0毫秒 (1.26×)	**615.0毫秒 (1.94×)**
WMT14 (翻译)	5G (强)	418.0毫秒 (1.0×)	395.0毫秒 (1.06×)	350.0 毫秒（1.19×）	195.0 毫秒（2.14×）	**172.0 毫秒（2.43×）**	282.0 毫秒（1.48×）	208.0 毫秒（2.01×）
	4G（平均）	575.0毫秒 (1.0×)	560.0毫秒 (1.03×)	720.0毫秒 (0.80×)	445.0毫秒 (1.29×)	415.0毫秒 (1.38×)	460.0毫秒 (1.25×)	**302.0毫秒 (1.90×)**
	WiFi（弱）	1188.0毫秒 (1.0×)	1168.0毫秒 (1.02×)	1725.0毫秒 (0.69×)	1405.0毫秒 (0.85×)	1372.0毫秒 (0.87×)	918.0毫秒 (1.29×)	**592.0毫秒 (2.00×)**
CNN/DM (摘要)	5G (强)	425.0毫秒 (1.0×)	400.0毫秒 (1.06×)	355.0毫秒 (1.20×)	198.0毫秒 (2.15×)	**175.0毫秒 (2.43×)**	285.0毫秒 (1.49×)	210.0毫秒 (2.02×)
	4G（平均）	582.0ms (1.0×)	565.0ms (1.03×)	728.0ms (0.80×)	452.0ms (1.29×)	420.0ms (1.38×)	465.0毫秒（1.25×）	**306.0毫秒 (1.90×)**
	WiFi（弱）	1205.0毫秒（1.0×）	1180.0毫秒（1.02×）	1750.0毫秒（0.69×）	1420.0 毫秒 (0.85×)	1382.0毫秒（0.87×）	924.0ms (1.30×)	**598.0ms (2.01×)**

表 IV B 模式 (T=1)：LLaMA-2 70B 在全部 6 个数据集上的端到端延迟和加速比。与其他方法相比，FlexSpec 在随机采样下仍能保持稳健性，而其他方法的表现则会下降。

数据集	网络	仅云端	Lookahead	标准 SD	Medusa-1	EAGLE-2	DSSD	FlexSpec
是否需要同步?		No	No	No	Yes	Yes	No	**No**
GSM8K （数学）	5G（强）	435.0毫秒（1.0×）	420.0毫秒（1.04×）	405.0毫秒（1.07×）	272.0毫秒（1.60×）	245.0毫秒（1.77×）	345.0毫秒（1.26×）	**232.0毫秒（1.87×）**
	4G（平均）	598.0毫秒 (1.0×)	585.0毫秒 (1.02×)	825.0毫秒 (0.72×)	665.0毫秒 (0.90×)	592.0毫秒 (1.01×)	540.0毫秒 (1.11×)	**360.0毫秒（1.66×）**
	WiFi（弱）	1225.0毫秒（1.0×）	1210.0毫秒（1.01×）	1880.0毫秒（0.65×）	1852.0毫秒（0.66×）	1680.0毫秒（0.73×）	1150.0毫秒（1.06×）	**705.0ms (1.74×)**
Natural Questions(问答)	5G (强)	418.0ms (1.0×)	395.0毫秒（1.06×）	355.0毫秒（1.18×）	290.0毫秒（1.44×）	258.0毫秒（1.62×）	332.0毫秒（1.26×）	**222.0毫秒（1.88×）**
	4G（平均）	575.0ms (1.0×)	560.0ms (1.03×)	700.0ms (0.82×)	655.0ms (0.88×)	590.0ms (0.97×)	528.0ms (1.09×)	**348.0ms (1.65×)**
	WiFi（弱）	1190.0 毫秒（1.0×）	1172.0毫秒 (1.02×)	1700.0毫秒 (0.70×)	1820.0ms（0.65×）	1665.0ms（0.71×）	1110.0ms（1.07×）	**690.0ms（1.72×）**
Natural Questions（检索增强生成）	5G（强）	430.0毫秒 (1.0×)	408.0毫秒 (1.05×)	370.0毫秒 (1.16×)	285.0毫秒 (1.51×)	255.0毫秒 (1.69×)	338.0毫秒 (1.27×)	**228.0ms (1.89×)**
	4G（平均）	588.0ms (1.0×)	575.0ms (1.02×)	750.0ms (0.78×)	660.0ms (0.89×)	595.0ms (0.99×)	532.0ms (1.10×)	**355.0ms (1.66×)**
	WiFi（弱）	1215.0 毫秒（1.0×）	1198.0 毫秒（1.01×）	1785.0毫秒 (0.68×)	1840.0毫秒 (0.66×)	1675.0毫秒 (0.72×)	1130.0毫秒 (1.07×)	**700.0毫秒 (1.73×)**
MT-Bench (聊天)	5G（强）	428.0毫秒（1.0×）	405.0毫秒（1.06×）	368.0毫秒（1.16×）	285.0毫秒（1.50×）	252.0毫秒（1.70×）	329.0 毫秒 (1.30×)	**225.0 毫秒 (1.90×)**
	4G (平均)	585.0 毫秒 (1.0×)	570.0 毫秒 (1.03×)	760.0 毫秒 (0.77×)	650.0ms (0.90×)	585.0ms (1.00×)	522.0ms (1.12×)	**344.0ms (1.70×)**
	WiFi（弱）	1210.0 毫秒（1.0×）	1195.0毫秒 (1.01×)	1795.0毫秒 (0.67×)	1805.0毫秒 (0.67×)	1652.0毫秒 (0.73×)	1100.0毫秒 (1.10×)	**685.0毫秒 (1.76×)**
WMT14 (翻译)	5G (强)	420.0毫秒 (1.0×)	400.0毫秒 (1.05×)	360.0毫秒 (1.17×)	295.0毫秒 (1.42×)	260.0毫秒（1.61×）	335.0毫秒（1.25×）	**225.0毫秒（1.87×）**
	4G（平均）	578.0毫秒（1.0×）	565.0毫秒（1.02×）	735.0 毫秒 (0.79×)	675.0 毫秒 (0.85×)	605.0 毫秒 (0.95×)	530.0 毫秒 (1.09×)	**350.0 毫秒 (1.65×)**
	WiFi（弱）	1195.0毫秒 (1.0×)	1180.0毫秒 (1.01×)	1740.0毫秒 (0.69×)	1900.0毫秒 (0.63×)	1690.0毫秒 (0.71×)	1120.0毫秒 (1.07×)	**715.0毫秒 (1.67×)**
CNN/DM (摘要)	5G (强)	425.0毫秒 (1.0×)	402.0毫秒 (1.06×)	365.0毫秒 (1.16×)	300.0毫秒 (1.42×)	265.0毫秒 (1.60×)	338.0毫秒 (1.26×)	**228.0毫秒 (1.86×)**
	4G（平均）	582.0毫秒 (1.0×)	568.0毫秒（1.02×）	740.0毫秒（0.79×）	680.0毫秒（0.85×）	610.0毫秒（0.95×）	535.0毫秒（1.09×）	**352.0毫秒（1.65×）**
	WiFi（弱）	1200.0毫秒 (1.0×)	1185.0毫秒 (1.01×)	1765.0毫秒 (0.68×)	1915.0毫秒 (0.63×)	1700.0毫秒 (0.71×)	1125.0毫秒 (1.07×)	**720.0毫秒 (1.67×)**

网络条件下仍具有较强的稳健性，但在 5G 环境中却显著未能充分利用可用带宽（405.0 ms，而非 220.0 ms）。相比之下，FlexSpec 会根据瞬时信道条件动态调整推测步幅。在有利的 5G 条件下，它能够达到 $K = 5$ 的低延迟（220.0 ms）；而在 WiFi 场景下，它会自动减小有效步幅，使其接近 $K = 1$ 的表现（625.0 ms），从而显著降低与静态推测策略相关的尾延迟风险。

D. RQ3：异构边缘硬件适应性

我们在设置中定义的特定边缘设备上评估了 FlexSpec。为更好地了解硬件限制如何与任务复杂度相互作用，我们测量了三个具有代表性的数据集上的加速比：GSM8K（复杂逻辑）、MT-Bench（聊天）和 HumanEval（代码）。

如表 V 所示，FlexSpec 是否可行取决于本地草稿生成速度与网络传输节省量之间的比率。树莓派 5 仅依靠 CPU，草稿生成速度为 6.9 词元/秒，低于计入网络延迟后的云端有效生成速度。这会导致系统变慢（0.72× - 0.85×），由此确立了硬件下限：FlexSpec 需要加速器支持（GPU/NPU）。

消费级移动设备（iPhone 15 Pro Max 和 Snapdragon 8 Gen 3）展现出令人瞩目的加速效果，接近工作站级别的 Jetson AGX Orin。通过将计算量大的 70B 目标模型卸载到云端，并仅在 NPU/Metal 后端上运行轻量级的对齐草稿，FlexSpec 让标准智能手机也能实现交互式大语言模型体验。

在 HumanEval 等更困难的任务上，由于接受率较低，需要更多验证轮次，因此所有设备的加速比都有所下降（例如，Jet- son 上从 2.10× 降至 1.88× ）。不过，支持 NPU 的设备仍保持了超过 1.75× 的稳健加速比，证实了即使在边缘硬件上进行复杂的代码生成，基于锚点的对齐依然有效。

表 V：4G 网络条件下 FlexSpec 在异构边缘设备上的性能（相较于仅云端的加速比）。

设备	处理器	草稿延迟	草稿吞吐量	GSM8K（困难）	MT-Bench（中等）	HumanEval（困难）
树莓派 5	Cortex-A76（CPU）	145 毫秒/令牌	6.9 个令牌/秒	0.76×（减速）	0.85×（减速）	0.72×（减速）
Jetson AGX Orin	Ampere GPU	8.5 毫秒/令牌	117.6 令牌/秒	**1.96**×	**2.10**×	**1.88**×
iPhone 15 Pro Max	A17 Pro（NPU）	12.0 毫秒/令牌	83.3 令牌/秒	1.82×	1.92×	1.75×
Snapdragon 8 Gen 3	Hexagon 神经网络处理器	10.5 毫秒/令牌	95.2 令牌/秒	1.93×	2.05×	1.85×

表六：FlexSpec 在较新模型架构上的可扩展性（数据集：MT-Bench，网络：5G/4G）。

目标模型	架构	基线延迟	FlexSpec（5G）	FlexSpec（4G）
Llama-2-70B	稠密	420.0毫秒 / 578.0毫秒	**1.95**×	**1.85**×
Llama-3-70B	稠密	395.0ms / 550.0ms	**2.30**×	**1.92**×
Mixtral 8x7B	MoE	320.0毫秒 / 485.0毫秒	**1.75**×	**1.68**×

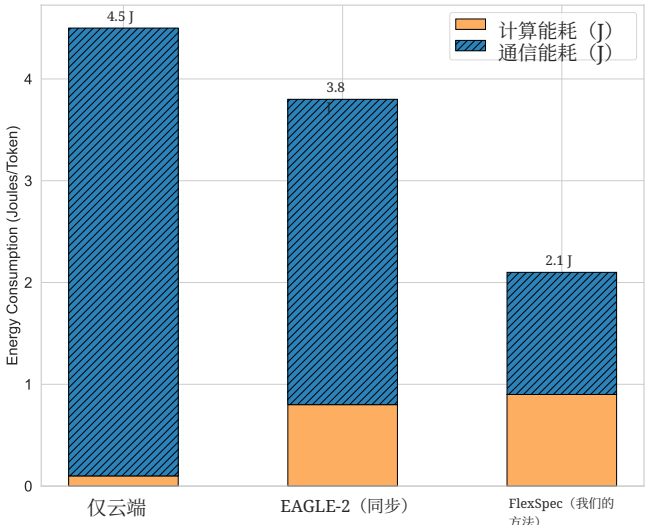


图 6. 移动设备上的能耗细分。与纯云端推理相比，FlexSpec 通过压缩令牌传输显著降低了通信能耗（蓝色斜线区域），从而使总能耗降低了 53%。

E. 研究问题*4*：模型可扩展性

为验证我们的基于锚点的对齐并非 Llama-2 架构专用，我们将评估扩展到了更新的 Llama-3 70B 和稀疏混合专家（MoE）模型 Mixtral 8x7B。

尽管 Llama-3 的词汇表更大，训练数据也各不相同，FlexSpec 在 5G 条件下仍实现了 2.30× 的峰值加速比（表六）。这表明，语义锚点这一概念可迁移至不同的稠密模型。对于 Mixtral 8x7B，由于采用条件计算（激活参数 $\approx$ 13B），其基线云端推理速度更快。虽然这缩小了推测增益的潜在空间，FlexSpec 在 4G 下仍实现了 1.68× 的加速比。信道感知策略会自动向下调整 K，以适应更快的云端验证，从而避免过度推测。

F. 研究问题*5*：内存与能效

最后，我们对 Jetson AGX Orin（工作站）和 Snapdragon 8 Gen 3（移动设备）上的资源消耗进行了详细分析，以量化效率提升。

完全端侧推理运行 70B 模型需要约 42.5 GB 显存（4 位量化），这在 Jetson 上可行，但在手机上则无法实现。FlexSpec 通常仅需 $\sim$3.5 GB 来存放草稿模型组件，现代智能手机的 12-16GB 内存足以轻松满足这一需求。图 6 揭示了 FlexSpec 能效高的原因。仅云端方案的能耗较高（4.5 J/词元），主要是无线电尾态造成的（通信能耗）。FlexSpec 在本地生成 K 词元的草稿，并将其压缩后一次性发送，从而显著缩短无线电活跃时间（通信能耗降至 1.2 J），与标准流式传输相比，总能耗降低了 53 %。完全在端侧运行大型模型会产生大量热量（Jetson 上温度高于 80°C），导致热节流。FlexSpec 将繁重计算转移到云端，使热特性保持在适合手持设备使用的低至中等水平。

VI. 结论

本文提出了 FlexSpec，这是一种基于推测解码、面向不断演进的边云系统的低通信开销协同推理框架。FlexSpec 旨在解决现有基于推测解码的方法在云端模型频繁更新和无线网络条件动态变化时所面临的可扩展性限制。为此，我们首先引入了一种共享骨干架构，使单个静态的边缘端草稿模型能够与一系列不断演进的云侧目标模型保持兼容，从而避免在边缘端反复重新训练和模型同步。随后，我们进一步提出了一种信道感知自适应推测机制，可根据时变网络条件动态调整推测草稿长度，以平衡端到端延迟、通信开销和设备能耗。最后，我们通过大量实验评估了 FlexSpec，结果表明，FlexSpec 的性能始终优于传统基线方法，并为在动态边云环境中部署大语言模型提供了有效策略。

参考文献

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat 等，《Gpt-4 技术报告》，*arXiv* 预印本 *arXiv:2303.08774*，2023 年。[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale 等，《Llama 2：开放基础模型与微调聊天模型》，*arXiv* 预印本 *arXiv:2307.09288*，2023 年。

[3] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. S oricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican 等，《Gemini：一系列高能力多模态模型》，arXiv 预印本 arXiv:2312 11805，2023 年。.[4] Y. Li, H. Cai, R. Kong, X. Chen, J. Chen, J. Yang, H. Zhang, J. Li, J. Wu, Y. Chen 等，《迈向人工智能搜索范式》，arXiv 预印本 arXiv:2506.17188，2025 年。[5] Y. Li, H. Zhang, Y. Zhang, X. Ma, W. Ye, N. Song, S. Wang, H. Xiong, D. Yin, and L. Chen，《Moerank：面向满意度的网页搜索多目标混合专家增强排序》，载于 2025 年 IEEE 第 41 届数据工程国际会议（ICDE）。IEEE，2025 年，第 4441–4454 页。[6] H. Xiong, J. Bian, Y. Li, X. Li, M. Du, S. Wang, D. Yin, and S. Helal，《当搜索引擎服务遇上大语言模型：愿景与挑战》，IEEE 服务计算汇刊，2024 年。[7] Y. Li, Z. Lyu, Y. Zhang, H. Zhang, T. Peng, H. Xiong, S. Wang, L. Kong, G. Chen, and D. Yin，《Sprank：通过半监督预训练实现面向满意度的学习排序》，IEEE 知识与数据工程汇刊，2025 年。[8] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi 等，《Mobilellm：针对端侧应用场景优化十亿参数以下的语言模型》，载于第四十一届国际机器学习会议，2024 年。[9] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui，《边缘智能：为网络边缘赋能智能》，IEEE 会刊，第 109 卷，第 11 期，第 1778–1837 页，2021 年。[10] J. Liu, Y. Du, K. Yang, J. Wu, Y. Wang, X. Hu, Z. Wang, Y. Liu, P. Sun, A. Boukerche 等，《面向分布式智能与模型优化的边缘云协同计算：综述》，arXiv 预印本 arXiv:2505.01821，2025 年。[11] Z. Lyu, Y. Li, G. Zhu, J. Xu, H. Vincent Poor, and S. Cui，《重新思考边缘学习中的资源管理：联合预训练与微调设计范式》，第 1584–1601 页，2025 年。[12] Y. Leviathan, M. Kalman, and Y. Matias，《通过推测解码实现 Transformer 的快速推理》，载于国际机器学习会议，2023 年。[13] Y. Li, F. Wei, C. Zhang, and H. Zhang，《EAGLE：推测采样需要重新思考特征不确定性》，载于国际机器学习会议，2024 年。[14] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao，《Medusa：采用多个解码头的简易大语言模型推理加速框架》，arXiv 预印本 arXiv: 2401.10774，2024 年。[15] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper，《通过推测采样加速大语言模型解码》，arXiv 预印本 arXiv:2302.01318，2023 年。[16] Y. Li, F. Wei, C. Zhang, and H. Zhang，《EAGLE-2：借助动态草稿树加速语言模型推理》，载于自然语言处理实证方法会议，2024 年。[17] N. Yang, T. Ge, L. Wang, B. Jiao, D. Jiang, L. Yang, R. Majumder, and F. Wei，《基于参考的推理：大语言模型的无损加速》，arXiv 预印本 arXiv:2304.04487，2023 年。[18] Z. He, Z. Zhong, T. Cai, J. Lee, and D. He，《REST：基于检索的推测解码》，载于 2024 年计算语言学协会北美分会会议论文集，2024 年。[19] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen，《LoRA：大语言模型的低秩适配》，载于国际学习表征会议，2022 年。[20] Z. Lyu, G. Zhu, J. Xu, B. Ai, and S. Cui，《通过深度联合信源信道编码实现图像恢复与分类的语义通信》，I EEE 无线通信汇刊，第 23 卷，第 8 期，第 8388–8404 页，2024 年。[21] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang，《Neurosurgeon：云端与移动边缘之间的协同智能》，载于第二十二届编程语言与操作系统架构支持国际会议论文集，2017 年。[22] Y. Matsubara, M. Levorato, and F. Restuccia，《深度学习应用的分割计算与提前退出：综述与研究挑战》，ACM 计算调查，2024 年。[23] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel，《Petals：大模型的协同推理与微调》，载于 2022 年拓展研究合作研讨会，2022 年。[24] M. Ryabinin, T. Dettmers, M. Diskin, and A. Borzunov，《蜂群并行：训练大模型的通信效率可能出奇地高》，载于国际机器学习会议，2023 年。

[25] Z. Lyu, M. Xiao, J. Xu, M. Skoglund 和 M. D. Renzo，“越大越好？无线边缘网络中的高效大型人工智能模型推理”，*IEEE*通信领域精选期刊，2025。[26] E. Frantar、S. Ashkboos、T. Hoefler 和 D. Alistarh，“GPTQ：生成式预训练变换器的精准训练后压缩”，载于第十一届国际学习表征会议，2023。[27] J. Lin、J. Tang、H. Tang、S. Yang、W.-M. Chen、W.-C. Wang、G. Xiao、X. Dang、C. Gan 和 S. Han，“AWQ：面向端侧大语言模型压缩与加速的激活感知权重量化”，载于《机器学习与系统论文集》，2024。[28] M. Li、T. Cai、J. Cao、Q. Zhang、H. Cai、J. Bai、Y. Jia、M.-Y. Liu、K. Li 和 S. Han，“Distrifusion：高分辨率扩散模型的分布式并行推理”，载于*IEEE/CVF*计算机视觉与模式识别会议论文集，2024。[29] J. Park、S. Samarakoon、M. Bennis 和 M. Debbah，“边缘端的无线网络智能”，*IEEE* 会刊，第107卷，第2204–2239页，2019。[30] G. Zhu、D. Liu、Y. Du、C. You、J. Zhang 和 K. Huang，“迈向智能边缘：无线通信与机器学习的交汇”，*IEEE*通信杂志，第58卷，第19–25页，2020。[31] J. Xin、R. Tang、J. Lee、Y. Yu 和 J. Lin，“DeeBERT：通过动态提前退出加速BERT推理”，载于计算语言学协会第*58*届年会论文集，2020。[32] S. Laskaridis、S. I. Venieris、M. Almeida、I. Leontiadis 和 N. D. Lane，“SPINN：设备与云端神经网络的协同渐进式推理”，载于第*26*届移动计算与网络国际年会论文集，2020。[33] J. Ning、C. Zheng 和 T. Yang，“DSSD：通过分布式拆分式推测解码实现高效的边缘设备部署与协同推理”，载于第四十二届国际机器学习会议，2025。[34] S. Malladi、T. Gao、E. Nichani、A. Damian、J. D. Lee、D. Chen 和 S. Arora，“仅通过前向传播微调语言模型”，载于第三十七届神经信息处理系统会议，2023。[35] G. Hinton、O. Vinyals 和 J. Dean，“蒸馏神经网络中的知识”，载于第二十九届神经信息处理系统会议，2015。[36] M. Weber、D. Y. Fu、Q. G. Anthony、Y. Oren、S. Adams、A. Alexandrov、X. Lyu、H. Nguyen、X. Yao、V. Adams、B. Athiwaratkun、R. Chalamala、K. Chen、M. Ryabinin、T. Dao、P. Liang、C. Re、I. Rish 和 C. Zhang，“RedPajama：用于训练大语言模型的开放数据集”，载于第三十八届神经信息处理系统会议，2024。[37] X. Jiao、Y. Yin、L. Shang、X. Jiang、X. Chen、L. Li、F. Wang 和 Q. Liu，“TinyBERT：通过蒸馏BERT实现自然语言理解”，载于*2020*年自然语言处理实证方法会议论文集，T. Cohn、Y. He 和 Y. Liu 编，2020。[38] Y. Gu、L. Dong、F. Wei 和 M. Huang，“MiniLLM：大语言模型的知识蒸馏”，载于第十二届国际学习表征会议，2024。[39] X. Miao、G. Oliaro、Z. Zhang、X. Cheng、Z. Wang、Z. Zhang、R. Y. Y. Wong、A. Zhu、L. Yang、X. Shi、C. Shi、Z. Chen、D. Arfeen、R. Ab-hyankar 和 Z. Jia，“SpecInfer：利用基于树的推测推理与验证加速大语言模型服务”，载于第*29*届*ACM*编程语言与操作系统架构支持国际会议论文集，2024。[40] Y. Zhao、Z. Xie、C. Liang、C. Zhuang 和 J. Gu，“Lookahead：一种保证生成准确率无损的大语言模型推理加速框架”，载于第*30*届*ACM*知识发现与数据挖掘会议论文集，2024。