上海交通大学硕士学位论文

基于服务器无感知架构的深度学习集群资源利用率提升研究

姓　　名：刘俊涵

学　　号：123033910039

导　　师：马汝辉教授

院　　系：计算机学院

学科/专业：计算机科学与技术

申请学位：工学硕士

2026 年 1 月 16 日

A Dissertation Submitted to

Shanghai Jiao Tong University for the Degree of Master

RESEARCH ON IMPROVING RESOURCE UTILIZATION OF DEEP LEARNING CLUSTERS BASED ON SERVERLESS ARCHITECTURES

Author: Liu Junhan

Supervisor: Prof. Ma Ruhui

School of Computer Science

Shanghai Jiao Tong University

Shanghai, P.R. China

January 16th, 2026

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学

学位论文使用授权书

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于：

□**公开论文**

□**内部论文**，保密□1 年/□2 年/□3 年，过保密期后适用本授权书。

□**秘密论文**，保密____年（不超过 10 年），过保密期后适用本授权书。

□**机密论文**，保密____年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“✓”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

随着深度学习在自然语言处理、语音识别、推荐系统、图像识别、视频处理等领域的广泛应用，大规模图形处理单元（Graphics Processing Unit，GPU）集群已成为支撑深度学习训练与推理任务的关键基础设施。为满足复杂模型和海量数据的算力需求，大型科研机构与云服务提供商纷纷建设并扩展多租户深度学习集群，但随之而来的是显著攀升的设备及运维成本。与此同时，真实生产环境中多租户深度学习集群的GPU 利用率普遍偏低，大量 GPU 处于长期闲置或轻载状态，造成严重的资源浪费并制约整体系统吞吐和性价比的提升。因此，在现有硬件条件下，通过系统与调度机制优化挖掘剩余 GPU 能力、提升资源利用率，具有重要的工程意义和应用价值。

在 CPU 与内存资源管理领域，已有研究利用服务器无感知计算这一新兴的轻量级架构范式，实现了对系统空闲资源的细粒度回收与复用。服务器无感知计算具备按需调度、弹性伸缩与细粒度资源分配等特点，尤其适合承载突发性强、执行时间短的函数式任务。然而，当前针对 GPU 资源的回收机制尚不完善，主要瓶颈在于缺乏高效的 GPU 多租户隔离与资源共享方案。当服务器无感知函数实例与常规深度学习任务共享同一 GPU 时，二者在显存与计算资源上易发生竞争，进而导致性能抖动、训练延迟乃至任务超时等问题。通过在自建集群中的观察实验，本文发现：不合理的任务共置会引发显著性能下降，且性能退化随共置任务数量增加而累积；服务器无感知函数通常具有严格的时延约束，性能不确定性易导致其无法按时完成；此外，服务器无感知函数普遍存在冷启动现象，在某些场景下冷启动耗时远超函数实际执行时间，从而显著增加端到端延迟。因此，在 GPU 集群中实现服务器无感知负载与常规服务器式深度学习负载的安全、高效共置，是一项兼具挑战性与实践价值的研究课题。

针对上述问题，本文提出了面向深度学习集群的 GPU 资源优化框架 SMore 。该框架首次系统性地将服务器无感知计算范式引入多租户 GPU 集群，以服务器无感知函数任务作为填充负载，复用常规深度学习任务未占满的 GPU 资源，从而提升整体利用率。与以往仅面向单一负载或将 GPU 视为不可分割资源的方案不同，SMore 旨在保持常规深度学习任务主导地位的前提下，有选择地接纳合适服务器无感知函数任务进行共置，兼顾利用率提升、常规深度学习任务性能稳定与服务器无感知函数任务服务质量保障。本文的主要研究内容包含以下三部分工作：

1. 针对不同负载共置执行可能产生的性能退化问题，本文设计并实现了性能退

化预测器，由双路基准预测器与多路聚合预测器构成，采用离线训练与在线更新相结合。离线阶段通过实验收集不同共置组合下的运行数据构建训练集，比较多种回归模型后选取精度与数据开销综合最优的作为核心预测器；多任务子预测器则基于双路预测结果进行组合推理，以支持更复杂的共置场景。

2. 针对服务器无感知计算函数任务可能无法满足服务质量目标的问题，基于性能退化预测结果，本文设计了一种感知性能退化的调度策略，致力于在最大化资源利用率与控制性能损耗之间寻求平衡。该策略结合实时集群监控状态与性能预测结果，对进入的无服务器函数请求进行准入控制。通过智能评估并选择最合适的 GPU 节点进行任务放置，该算法能够在严格控制性能衰退幅度并满足函数服务质量目标时限要求的前提下，有效利用集群中的碎片化 GPU 资源。

3. 针对服务器无感知函数冷启动问题，本文提出了一种基于双周期长短期记忆网络（Long–Short cycle Long Short-Term Memory, LS-LSTM）的动态预热机制，以缓解服务器无感知计算中严重的冷启动现象。该机制利用 LS-LSTM 模型捕捉函数请求到达的长周期与短周期时序模式，实现对未来负载的较为精确预测。基于预测结果，预热器能够在请求到达前提前加载深度学习模型以消除冷启动延迟，或在空闲阶段及时卸载模型以减少资源浪费，从而在响应速度与资源占用之间实现动态权衡与优化。

为了验证该系统的性能，本文实现了 SMore 的原型系统，并在基于真实 Azure 函数跟踪数据的自建 GPU 集群上进行了广泛实验。实验结果表明，SMore 能够有效应对混合部署带来的挑战，在将工作负载性能衰退控制在可接受范围内的同时，使集群的 GPU 资源利用率最高可提升 34%。

关键词：深度学习集群，资源利用率，服务器无感知计算，混合部署，GPU 调度

Abstract

With the widespread application of deep learning in fields such as natural language processing, speech recognition, recommendation systems, image recognition, and video processing, large-scale GPU clusters have become critical infrastructure supporting training and inference tasks. To meet the computational demands of complex models and massive datasets, large research institutions and cloud service providers have extensively constructed and expanded multi-tenant deep learning clusters. However, this expansion is accompanied by significantly escalating equipment and operational costs. Concurrently, GPU utilization in real-world production multi-tenant deep learning clusters remains universally low, with a substantial number of GPUs remaining in long-term idle or underloaded states. This results in severe resource wastage and constrains the improvement of overall system throughput and cost-effectiveness. Therefore, optimizing the exploitation of residual GPU capacity through system and scheduling mechanisms under existing hardware conditions holds significant engineering significance and application value.

In the domains of CPU and memory, existing research has leveraged lightweight paradigms, such as serverless computing, for fine-grained resource reclamation and reuse. Serverless computing, characterized by its on-demand, elastic, and fine-grained nature, is well-suited for handling bursty, short-term tasks. However, existing resource reclamation studies have yet to effectively cover GPUs, primarily due to the lack of mature multi-tenant isolation and sharing mechanisms. When function instances share the same GPU with deep learning tasks, competition for video memory and computing power often leads to performance jitter, training delays, or even task failures. Empirical tests on a self-built cluster indicate that unreasonable co-location triggers significant performance degradation, which accumulates as the number of co-located tasks increases. Furthermore, function tasks typically operate under strict latency constraints, making them vulnerable to performance uncertainty. Additionally, deep learning-related functions universally suffer from cold start issues, where initialization latency often far exceeds effective execution time, drastically increasing end-to-end latency. Consequently, safe and efficient co-location of serverless functions and deep learning workloads within GPU clusters represents a challenging yet valuable research problem.

To address these issues, this thesis proposes SMORE, a GPU resource optimization framework tailored for deep learning clusters. This framework is the first to systematically introduce the serverless computing paradigm into multi-tenant GPU clusters, utilizing function tasks as ”filler workloads” to reuse GPU resources not fully occupied by deep learning tasks, thereby enhancing overall utilization. Unlike previous approaches that focus on single workload types or treat GPUs as indivisible units, SMORE aims to selectively admit appropriate function tasks for co-location while maintaining the dominance of serverful deep learning tasks, balancing utilization improvement with performance stability and service quality assurance. The main research contributions are as follows:

1. A performance degradation predictor is designed and implemented to quantitatively characterize the impact of co-location. The predictor comprises a pair-wise co-location sub-predictor and a multi-way co-location sub-predictor, employing a strategy that combines offline training with online updates. In the offline phase, training data is collected from various co-location combinations to select the regression model with the optimal balance of accuracy and data overhead. The multi-way sub-predictor then performs combined inference based on pair-wise results to support complex co-location scenarios.

2. A degradation-aware dynamic scheduling algorithm is designed based on performance predictions. This scheduler seeks a balance between maximizing resource utilization and controlling performance loss. By integrating real-time cluster monitoring with prediction results, the scheduler performs admission control for incoming serverless function requests. Through intelligent evaluation and selection of the most suitable GPU nodes for task placement, the algorithm effectively utilizes fragmented GPU resources while strictly confining performance degradation and meeting function Service Level Objectives (SLOs).

3. A dynamic prewarming mechanism based on a Long-Short cycle Long Short-Term Memory (LS-LSTM) network is proposed to address function cold starts. This mechanism leverages the LS-LSTM model to capture both long-term and short-term temporal patterns of function request arrivals, achieving precise prediction of future workloads. Based on these predictions, the prewarmer can preload deep learning models in advance to eliminate cold start latency or unload models during idle periods to minimize resource wastage, thereby dynamically optimizing the trade-off between response speed and resource occupancy.

To validate system performance, a prototype of SMORE was implemented and extensively

tested on a self-built GPU cluster using real-world Azure function traces. Experimental results demonstrate that SMore effectively addresses the challenges of hybrid deployment, improving cluster GPU resource utilization by up to 34% while maintaining workload performance degradation within an acceptable range.

Key words: Deep Learning Clusters, Resource Utilization, Serverless Computing, Co-location, GPU Scheduling

目 录

插 图

表 格

算 法

第 1 章 绪论

1.1 研究背景及意义

近年来,在硬件加速技术飞跃与大数据浪潮的共同推动下,深度学习(Deep Learning, DL)已成为推动人工智能前沿发展的核心引擎[1]。随着神经网络结构的日益复杂与海量数据集的广泛应用,深度学习算法在自然语言处理[2]、语音识别[3]、推荐系统[4]、图像识别[5]、视频处理[6]等关键领域取得了突破性的性能提升。为了满足日益增长的计算需求,大型研究机构与科技企业纷纷斥巨资建设大规模 GPU 集群[7-8],以此作为支撑复杂模型训练与高效在线推理的基础设施。GPU 集群凭借其强大的并行计算能力,极大地促进了大规模数据的处理效率。可以预见,随着人工智能应用的深化,GPU 集群的规模化部署将是未来发展的必然趋势,而如何在高昂的部署成本下优化其资源使用效率,已成为学术界和工业界共同关注的焦点。

然而,尽管硬件设施不断升级,现有的多租户深度学习集群普遍面临 GPU 资源利用率低下的严峻挑战,这不仅限制了系统的整体吞吐量,也造成了巨大的算力浪费[9]。基于对现有工作[7,10]及开源生产环境跟踪日志(Trace)[11]的分析,本文将这一问题归因于 GPU 资源分配机制的僵化与分布式训练任务的固有特性两大方面。从资源分配角度看,在传统的集群管理中,GPU 通常被视为不可分割的独占单元,缺乏原生的细粒度共享机制[8,12]。这种粗粒度的分配方式导致大多数深度学习任务无法充分饱和 GPU 的流式多处理器(Streaming Multiprocessor, SM),据统计约 90% 的 GPU 在运行时的利用率低于 80%[10]。从任务特性角度看,分布式训练中频繁的参数同步与通信开销极易成为性能瓶颈,导致 GPU 在等待网络传输时处于空闲状态。研究表明,在云端训练场景中,高达 90% 的时间可能被耗费在网络等待上[13]。此外,诸如 Gang Scheduling 等传统调度策略的刚性需求也进一步加剧了资源的碎片化与闲置[7]。因此,现有的集群管理模式已难以适应高效能计算的需求。

在此背景下,服务器无感知计算(Serverless Computing)作为云计算领域的一次范式转移,为解决上述资源利用难题提供了新的契机。服务器无感知计算,或称功能即服务(Functions as a Service, FaaS),允许开发者专注于业务逻辑代码(即“函数”),而将底层的服务器配置、维护与弹性伸缩交由云服务商全权接管[14-16]。其核心优势在于极细粒度的资源按需分配与毫秒级的弹性伸缩能力。若能将服务器无感知计算引入利用率低下的深度学习集群,利用其灵活的调度特性来填充常规服务器式任务

（Serverful Workloads）留下的资源空隙，理论上可以打破 GPU 独占的限制，实现资源的细粒度复用，从而显著提升集群的整体效能。

尽管前景广阔，但将服务器无感知计算应用于 GPU 集群并非易事，现有的服务器无感知研究主要集中在 CPU 与内存资源的回收利用上[17-18]，在 GPU 领域仍面临诸多技术鸿沟。首先，资源隔离与干扰问题是主要障碍。GPU 缺乏成熟的硬件级隔离机制，当服务器无感知函数与常规深度学习任务共置（Co-location）时，极易产生严重的资源争抢，导致常规任务性能下降（实验观测可达 10% 以上），同时也难以保障服务器无感知函数的服务等级目标（Service Level Objectives, SLO）。其次，冷启动（Cold Start）延迟在 GPU 场景下尤为凸显。由于深度学习模型通常体积庞大，加载模型至 GPU 显存的开销巨大，某些任务的冷启动时间甚至高达热启动的 100 倍，严重削弱了服务器无感知计算即时响应的优势。因此，如何在保障现有任务性能的前提下，克服冷启动延迟并高效调度 GPU 资源，是实现服务器无感知化 GPU 集群的关键挑战。

本工作旨在探索并设计一种新型的集群资源优化系统，即 SMore，以解决上述挑战。通过构建高精度的性能干扰预测模型、设计感知干扰的动态调度算法以及基于长短时记忆网络的预热机制，本研究致力于在多租户深度学习集群中实现服务器无感知负载与常规负载的高效共存。这不仅能够显著提升昂贵的 GPU 集群资源利用率，降低运营成本，同时也拓展了服务器无感知计算在高性能计算领域的应用边界，具有重要的研究意义与实用价值。

1.2　国内外研究现状

本节对利用服务器无感知计算优化深度学习集群 GPU 资源利用率相关的国内外研究现状进行介绍。首先，第 1.2.1 节对深度学习集群中的工作负载特性进行分析，旨在阐明资源利用率低下的根本原因。其次，鉴于资源调度与共享策略直接决定了集群的运行效率，第 1.2.2 节对现有的深度学习工作负载调度及 GPU 资源共享相关研究进行介绍。再次，利用服务器无感知计算在细粒度资源管理上的独特优势，第 1.2.3 节重点阐述基于服务器无感知架构的集群资源优化方法。最后，针对服务器无感知计算引入的启动延迟挑战，第 1.2.4 节对冷启动优化机制相关的工作进行介绍。

1.2.1 深度学习工作负载特征与执行瓶颈分析

随着人工智能技术的飞速演进，特别是近年来在计算机视觉、自然语言处理等领域取得的突破性进展，深度学习模型的参数规模与计算复杂度呈现指数级增长趋势。面对海量参数模型训练对算力的巨大需求，分布式训练已成为当前深度学习领域的主流计算范式。然而，在实际生产环境中，GPU 集群的资源管理与调度面临着计算与通信解耦难、作业间干扰严重、以及异构资源适配复杂等多重挑战[19]。针对由于工作负载特性导致的资源利用率低下问题，国内外的研究主要可以归纳为两个类别：一类是基于大规模生产集群日志的宏观特性分析与建模，旨在揭示任务资源需求的统计规律；另一类是针对分布式训练微观执行机制的优化，致力于缓解通信同步与并行策略带来的性能瓶颈。

第一类研究侧重于通过大规模集群跟踪日志分析，刻画深度学习工作负载的资源需求模式与时空分布特征。深入理解作业的运行特征是优化调度策略的前提。Hu 等人[20]对商汤科技的大规模生产集群日志进行了深入剖析，揭示了深度学习作业在执行时长、GPU 使用量以及用户提交行为上存在显著的长尾分布特征，并基于此提出了能够预测作业特征的调度框架以最小化平均完成时间。Gu 等人[21]通过分析微软的生产集群发现，深度学习作业具有全有或全无的执行特性且运行时长难以预测，现有的调度器往往导致较长的排队延迟，进而提出了 Tiresias 调度器，利用离散化的 Gittins 指数算法在信息未知的条件下优化作业完成时间。在架构适配层面，Wang 等人[22]基于阿里巴巴 PAI 平台的实践经验指出，随着模型规模的扩大，传统的参数服务器（Parameter Server, PS）架构逐渐成为通信瓶颈，而 Ring AllReduce 架构[23]能更高效地利用 GPU 间的高速互连技术，从而显著提升带宽利用率。

第二类研究聚焦于分布式训练过程中的系统瓶颈，特别是针对通信开销、同步机制及并行策略的优化。分布式训练的效率往往受限于木桶效应，即系统整体性能取决于最慢的节点。Gu 等人[24]的研究指出，分布式训练的吞吐量随 GPU 数量增加呈现非线性扩展特性，且训练效率极易受计算节点物理位置与网络拓扑的影响。在广泛采用组调度策略的集群中，网络抖动或负载不均导致的滞后节点会迫使整个同步组处于空闲等待状态，显著降低了 GPU 的有效计算吞吐量，为解决这一问题，Narayanan 等人[25]提出了 PipeDream 系统，引入了流水线并行技术，通过将跨批次的流水线与批次内的并行相结合，实现了计算与通信的有效重叠，从而在隐藏通信延迟的同时提升了硬件效率。此外，针对数据密集型任务，Murray 等人[26]的研究表明，I/O 开销是制约扩展性的关键因素。在网络调度层面，Peng 等人[27]与 Sapio 等人[28]则提出了网

络拓扑感知的调度算法，通过优化任务放置策略以减少跨交换机的流量竞争。

值得注意的是，上述现有研究在揭示工作负载特征与优化分布式训练效率方面已取得丰硕成果，但仍存在局限性。上述工作大多聚焦于优化特定类型的长周期训练任务或依赖于静态的资源预留机制，缺乏对细粒度资源动态变化的适应性分析。与 PERFLOW[29]侧重于提供性能调试与瓶颈检测工具不同，本文的研究视角拓展至更广泛的深度学习通用工作负载（包括在线推理与离线训练）。本文致力于全景式地刻画异构负载在资源需求模式及延迟敏感度上的差异，论证服务器无感知计算在解决资源碎片化与启动延迟挑战方面的必要性，从而为设计更加高效、弹性的集群资源管理策略提供数据支撑与理论依据。

1.2.2　深度学习工作负载调度与资源共享研究

在当前的多租户生产集群环境中，深度学习训练与推理工作负载的资源管理主要依赖于 Kubernetes 或 YARN 等成熟的通用基础设施平台。然而，传统的资源管理模式普遍沿用静态且独占式的分配策略，即一旦 GPU 计算资源被分配给某个特定的任务，无论该任务当前处于高负载的矩阵运算阶段，还是处于低负载的数据加载、预处理或通信同步阶段，这些资源均被长期锁定。这种粗粒度的管理机制导致了严重的资源碎片化现象，使得集群内普遍存在 GPU 分配率高但实际利用率低下的矛盾问题[8]。特别是在大模型训练日益普及、高端 GPU 资源（如 NVIDIA A100/H100）极度稀缺与昂贵的背景下，如何打破独占式分配的壁垒，实现细粒度的资源共享，已成为提升集群效能亟待解决的关键挑战。针对这一问题，国内外学者进行了广泛的研究，相关工作主要可以归纳为基于集群全局视角的调度策略与基于单节点粒度的资源共享机制两个类别。

第一类研究侧重于集群全局视角的调度策略，旨在通过优化作业排队、放置与弹性伸缩策略来提升整体吞吐量与公平性。早期的研究主要关注如何通过时间切片技术实现资源的逻辑复用。Gandiva[30]作为该领域的开创性工作，引入了挂起与恢复机制，通过时间分片技术在不同深度学习任务间快速切换 GPU 执行上下文，并结合任务迁移与资源打包策略，实现了 GPU 资源的细粒度时分复用。在此基础上，针对异构集群的调度策略成为研究热点。Narayanan 等人提出的 Gavel[31]将调度策略形式化为异构性感知的优化问题，通过将不同优先级的策略表示为矩阵形式，实现了多种调度目标（如公平性、最小化完成时间）的统一表达。为了进一步应对动态负载，Qiao 等人设计了 Pollux[32]架构，该系统打破了资源分配与作业参数配置之间的隔阂，通过协同优化作业的批量大小与资源分配量，在提升集群有效吞吐量的同时提高了

资源利用效率。针对多租户公平性问题，Gandiva$_{fair}$[33]与 Themis[34]分别基于最大最小公平性和帕累托效率设计了复杂的拍卖或仲裁机制。此外，最新的研究如 Sia[35]进一步引入了自适应的弹性伸缩算法，能够根据集群的实时负载压力动态调整任务的并行度与放置位置，从而在异构且动态的环境中保持高效运行。

第二类研究聚焦于单节点粒度的资源共享与隔离机制，致力于解决多任务共存时的显存竞争与性能干扰问题。在同一块物理 GPU 上并发执行多个任务是提升利用率的直接手段，但这也带来了严峻的隔离挑战。Yu 等人提出的 Salus[36]是一种细粒度的 GPU 共享原语，它通过在软件层面拦截 CUDA 调用，实现了高效的显存管理和计算流调度，从而允许在无须修改用户代码的情况下安全地进行多任务空间共享。为了在保障高优先级任务服务质量（Quality of Service, QoS）的同时利用空闲资源，AntMan[10]引入了“机会主义工作负载”概念，系统动态监控 GPU 算力使用情况，利用低优先级任务填补微小的资源空隙，并在发生冲突时快速降级。在硬件辅助隔离方面，Nvidia 推出的 MIG（Multi-Instance GPU）技术虽然提供了物理级的故障隔离，但其固定的划分规格限制了灵活性。针对这一局限，Zhu 等人提出的 MGM[37]构建了一种软件定义的细粒度共享框架，通过在 MIG 实例之上进一步实施基于显存带宽调节的虚拟化管理，显著提升了大型语言模型服务场景下的资源密度。此外，Orion[38]采取了更为底层的优化手段，通过设计算子级（Kernel-level）的调度算法，在亚毫秒级粒度上管理 GPU 指令流的并发执行，进一步压榨了硬件性能。

然而，无论是基于预留的调度策略（如 Pollux, Gavel）还是基于细粒度切分的共享机制（如 Salus, MGM），其本质上仍依赖于相对静态的资源视图或复杂的启发式规则，难以完美契合负载具有极高突发性的场景。大多数现有方案在应对从零启动或极短任务时，往往面临调度开销过大或资源回收不及时的问题。与上述工作不同，本文提出的 SMore 基于服务器无感知计算范式，利用其天然的事件驱动特性与极速弹性能力，将资源管理的粒度从“作业级”进一步细化至“函数级”。本文致力于探索在服务器无感知架构下，如何通过轻量级的状态感知与冷启动优化，在保证深度学习任务性能的前提下，实现比传统调度方案更优的资源效能。

1.2.3 基于服务器无感知计算的集群资源优化研究

服务器无感知计算作为云计算领域的一种革命性范式，通过屏蔽底层基础设施的运维细节，赋予了系统极细粒度的资源分配能力与毫秒级的弹性伸缩特性。然而，这种高度的抽象化也带来了资源管理的复杂性：如何在保障函数极速启动与执行性能的同时，最大化集群资源的利用率并降低运营成本，成为当前研究的核心议题。针

对这一挑战，学术界与工业界进行了大量探索，现有研究主要可以归纳为面向成本效益的闲置资源回收策略与面向服务质量的精细化调度架构两个类别。

第一类研究聚焦于空闲资源的动态回收与计算成本优化，旨在通过资源利用与混合供给机制解决资源碎片化问题。在传统静态集群中，预留但未被使用的资源形成了巨大的浪费。为了挖掘这部分潜能，Libra[18]提出了一种动态资源剖析架构，该系统实时监控集群各节点的 CPU 与内存水位，一旦检测到空闲气泡，便立即将其回收并分配给排队中的服务器无感知函数，从而在不增加硬件投入的情况下显著提升了吞吐量。为了进一步降低对昂贵公有云的依赖，UnFaaSener[17]设计了混合云环境下的计算卸载决策算法，优先利用用户本地已有的非服务器无感知空闲资源执行任务，仅在负载溢出时请求云服务。在成本控制方面，Zhang 等人提出的 MArk[39]构建了一个基于预测的资源供给框架，通过分析历史调用数据与云服务商的定价模型，动态调整实例规格与并发度，在满足服务水平目标（Service Level Objective, SLO）的前提下实现了计算成本的最小化。此外，针对有状态任务的资源效率，Shillaker 等人提出的 Faasm[40]通过引入轻量级的隔离机制与共享内存架构，大幅降低了状态同步带来的 I/O 开销，从而提升了单位资源下的任务处理密度。

第二类研究侧重于混合部署环境下的调度策略与 QoS 保障，致力于解决高并发场景下的资源争抢与长尾延迟问题。如何在保障长运行应用性能的同时，高效处理突发的短任务是调度优化的关键。ServerMore[41]提出了一种机会主义的混合调度机制，精准识别服务器的剩余算力来并发执行短周期的服务器无感知函数，实现了长短任务的高效混部。针对微服务链条中复杂的 SLA 约束，Kannan 等人提出的 GrandSLAM[42]设计了一种 SLA 感知的调度架构，通过对请求进行全局重排序与批处理，优先保障紧迫度高的请求，有效避免了队头阻塞问题。为了应对服务器无感知负载的高度波动性，Jia 等人提出的 Nightcore[43]构建了一个具有微秒级调度能力的函数计算引擎，通过细粒度的 I/O 与计算资源解耦，实现了高负载下的低延迟响应。此外，Aquatope[44]引入了贝叶斯优化方法，针对多阶段服务器无感知工作流自动搜索最优的资源配置参数，从而在复杂的异构环境中实现了 QoS 与成本的帕累托最优。INFless[45]则结合了非均匀扩展机制，通过动态合并细粒度请求来摊薄系统开销。

尽管上述工作在通用计算资源的调度优化方面取得了显著进展，但其在深度学习场景下的适用性仍存在局限。现有的服务器无感知优化方案（如 GrandSLAM, Nightcore）大多针对 CPU 密集型或轻量级 Web 服务设计，假设任务是无状态且启动迅速的。然而，深度学习工作负载具有显著的异构性与重负载特征：GPU 资源不仅昂贵且

不可抢占，模型加载带来的冷启动开销巨大，且推理（在线、低延迟）与训练（离线、高吞吐）任务对资源的需求截然不同。现有的 CPU 导向型调度器难以感知 GPU 显存的碎片化状态，也无法有效协调训练与推理任务的混合编排。与这些工作不同，本文提出的 SMore 专为 GPU 集群设计，利用服务器无感知范式实现了深度学习训练与推理任务的混合部署。通过引入 GPU 感知的状态管理与协同调度机制， SMore 旨在填补现有服务器无感知研究在高性能计算领域的空白，从根本上突破深度学习生产集群的资源利用率瓶颈。

1.2.4 服务器无感知计算环境下的冷启动优化研究

冷启动延迟是服务器无感知计算框架实现缩容至零特性的主要代价，也是制约其在时延敏感型应用中普及的关键性能瓶颈。当函数实例因长时间空闲被回收，或新版本函数首次被触发时，系统需经历资源调度、容器创建、运行时环境初始化以及代码加载等一系列复杂操作。对于深度学习任务而言，冷启动挑战尤为严峻：除了常规的容器启动开销外，还涉及昂贵的 GPU 驱动加载、CUDA 上下文初始化以及 GB 级别的模型参数传输[46]。这些操作的累积耗时往往高达数秒甚至数十秒，远超推理任务本身的执行时间。针对这一问题，国内外研究工作主要可以归纳为基于概率预测的预热策略与基于运行时机制的启动加速两个类别。

第一类研究侧重于基于概率预测的预热策略，旨在通过精准预测流量峰值，提前启动容器以掩盖初始化延迟。预热的核心挑战在于如何在“减少延迟”与“避免资源浪费”之间取得平衡。早期的研究如 SSC[47]和 faaShark[48]主要采用基于梯度的轻量级预测算法，通过监控调用频率的变化率来决定扩缩容时机。然而，简单的线性外推难以应对具有复杂波动特征的生产负载。针对这一问题，Shahrad 等人提出的 HHP（Hybrid Histogram Policy）[49]引入了统计学方法，通过构建历史空闲时间的概率直方图，动态推导最优的“保活窗口”（Keep-alive Window）和“预热窗口”(Pre-warming Window)，从而在统计意义上最小化资源浪费。在此基础上，INFless[45]提出了长短期直方图（LSTH）策略，同时维护长周期与短周期的访问分布，既能捕捉宏观的昼夜节律，又能敏锐感知局部的突发流量。此外，IceBreaker[50]则从异构性的角度出发，通过分析不同函数间的调用依赖关系，设计了能够感知异构硬件特性的预热策略，进一步提升了复杂微服务链路的启动速度。

第二类研究聚焦于基于运行时机制的启动加速，致力于通过系统层面的优化来削减初始化过程本身的开销。快照与检查点技术是该领域的优化热点。Catalyzer[51]提出了一种极速恢复机制，通过在特定检查点对应用程序的内存状态进行快照，并利用

OverlayFS 重用底层文件系统，实现了亚毫秒级的函数恢复。SEUSS[52]则采用单核思想，缓存常用函数的快照状态，从而跳过了冗余的库加载与环境配置路径。针对容器运行时的优化，SOCK[53]设计了更加精简的容器 Zygote 池，通过预先初始化一组通用依赖的父进程，在请求到达时仅需执行轻量级的克隆操作即可完成实例创建。在深度学习特定的 I/O 优化方面，FaaSwap[46]针对模型加载这一主要瓶颈，利用 NVLink 设计了显存与主机内存之间的高速交换机制。PipeSwitch[54]则进一步引入了流水线并行技术，通过将模型传输与 GPU 计算初始化重叠执行，实现了多任务间的快速上下文切换。

现有冷启动研究已在预测算法与系统加速方面积累了丰富成果，但在 GPU 集群场景下仍存在不足。首先，HHP 与 LSTH 等预测策略大多面向 CPU 环境设计，而在 GPU 资源极度昂贵且稀缺的场景下，过度预热导致的资源闲置成本是不可接受的，且简单的历史统计难以感知 GPU 显存的碎片化状态。其次，FaaSwap 与 PipeSwitch 等 I/O 加速技术虽然削减了加载延迟，但并未解决“何时预热”的决策问题。与上述工作不同，本文提出的 SMore 融合了上述两类方法的优势。不同于仅关注单一函数历史行为的传统预测器， SMore 结合了集群整体的实时资源水位进行协同调度。通过在资源充裕时激进预热、在资源紧张时保守回收，本文旨在建立一种成本感知的冷启动治理机制，在有效降低 GPU 任务启动延迟的同时，最大限度地减少昂贵算力的闲置浪费。

1.3　主要研究内容与研究思路

围绕提升多租户深度学习集群中 GPU 利用率、抑制混部性能干扰并降低服务器无感知推理冷启动时延这三个核心目标，本文的主要研究内容包括：(1) 构建面向不同类型负载混合部署场景的 GPU 性能退化预测模型，用于量化不同共置组合下的干扰程度；(2) 设计感知预测退化的服务器无感知调度策略，实现对推理请求的准入控制与资源分配决策，在保障多任务 SLO 的前提下提升资源利用率；(3) 提出基于 LS-LSTM 的请求流量预测与模型预热机制，通过提前加载与按需卸载深度学习模型，降低冷启动开销并减少无效资源占用。在此基础上，本文进一步给出整体研究思路与系统设计，如图 1.1 所示。

基于上文的调研，现有的大规模多租户深度学习集群普遍面临 GPU 利用率低下的问题，其根源在于粗粒度的 GPU 资源分配方式以及分布式训练任务固有的通信瓶颈。尽管服务器无感知计算具备细粒度资源管理和按需伸缩的优势，但现有的服务器

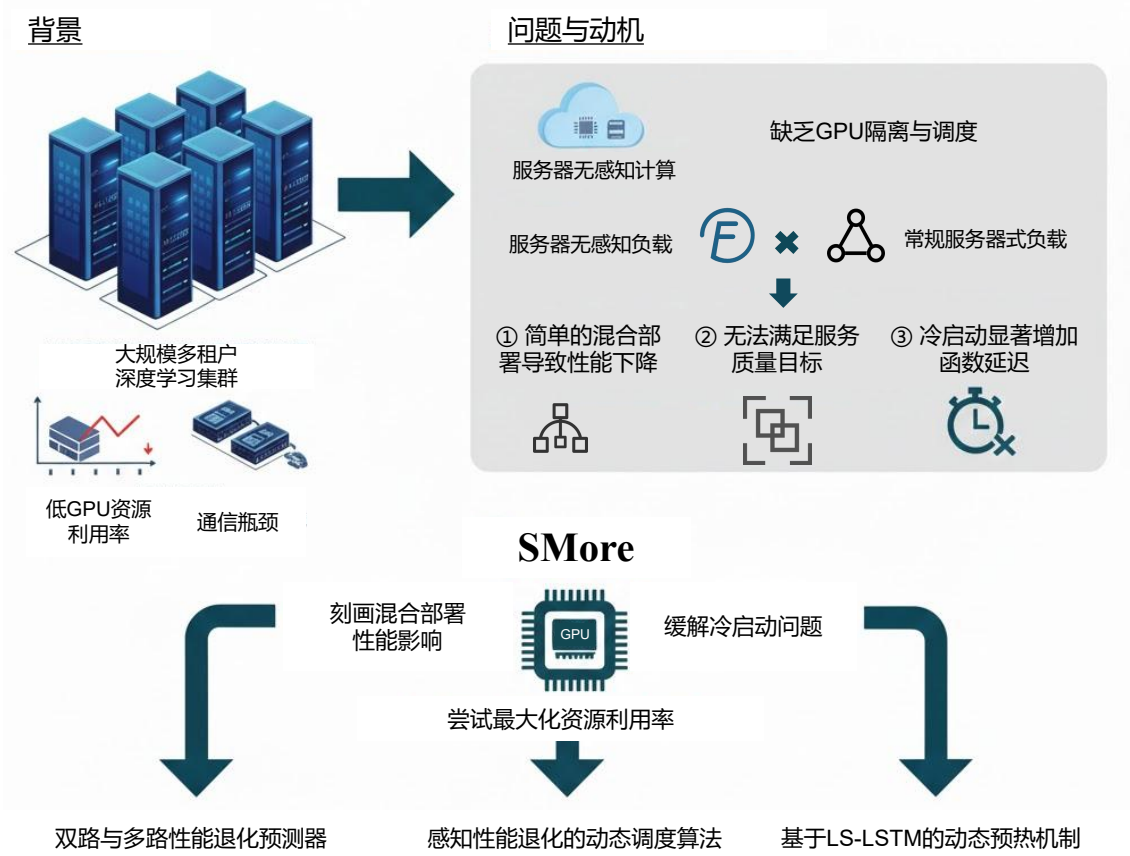

图 1.1 本文研究思路图

Figure 1.1 Research Ideas of This Paper

无感知方案主要集中在 CPU 和内存资源的利用上，缺乏对 GPU 资源的有效隔离与调度机制。为此，本文设计并实现了一个基于服务器无感知架构的 GPU 集群资源优化系统 SMore，旨在通过将服务器无感知与常规服务器式深度学习负载混合部署，以挖掘闲置的 GPU 算力。

由于 GPU 并不原生支持细粒度的资源共享，且不同类型的负载在混合部署时会产生激烈的资源竞争，导致常规任务的性能可能下降超过 10%，同时服务器无感知函数也面临无法满足服务等级目标的风险。为此，本文首先致力于解决混合部署带来的性能干扰问题。本文构建了一个性能下降预测器，用于量化共置负载之间的相互影响。具体的，该预测器包含双路（Pair-wise）和多路（Multi-way）子预测模型，采用离线训练与在线更新相结合的机制，以高精度预测潜在的性能损耗。基于预测结果，本文进一步设计了一种感知性能下降的调度器（Degradation-Aware Scheduler）。该调度器结合集群的实时监控状态，在接纳服务器无感知请求时动态评估其对现有任务的影响，并为其匹配最合适的 GPU 资源，从而在提升资源利用率的同时，将性能干

扰控制在可接受范围内并满足 SLO 要求。

此外，服务器无感知计算虽然灵活，但其特有的冷启动延迟严重影响了函数的响应速度，实验显示部分任务的冷启动开销高达热启动的 100 倍，这大大降低了系统处理瞬时请求的能力。为此，本文设计了一个预热模块（Prewarmer），利用双周期长短时记忆网络（Long Short LSTM, LS-LSTM）模型来应对这一挑战。该模型能够有效捕捉函数请求在长期和短期内的到达模式，精准预测下一阶段的负载情况。根据预测结果，预热模块可以提前动态加载深度学习模型以消除冷启动延迟，或在闲置时及时卸载模型以减少资源浪费，从而在响应时效性与资源占用之间取得平衡。

至此，本文设计并实现了一个支持异构负载混合部署的 GPU 集群优化系统，通过干扰预测、智能调度和动态预热三大核心机制，有效解决了 GPU 资源利用率低、混合部署性能干扰大以及服务器无感知冷启动延迟高的问题。

1.4　文章结构

本文余下内容的组织结构如下：

- 第 2 章对本文的技术背景和研究动机进行说明。首先，该章介绍了深度学习集群的资源管理现状及服务器无感知计算的基本范式。接下来，该章通过动机实验详细分析了现有集群 GPU 利用率低下的成因，并揭示了在缺乏有效隔离机制下，简单的混合部署方案所面临的严重的性能干扰和冷启动延迟挑战，从而确立本文的优化方向。
- 第 3 章对 SMore 系统的整体设计进行介绍。首先，该章对系统的总体架构和工作流程进行综述。然后，该章分别详细介绍了三个核心模块的设计原理：旨在量化共置负载干扰的性能下降预测器、基于预测结果进行动态资源分配的干扰感知调度器，以及基于双周期长短时记忆网络 LS-LSTM 以解决冷启动问题的动态预热模块。
- 第 4 章对 SMore 系统的实现细节进行说明。该章基于 Python 语言实现了系统原型，并详细介绍了原型系统的软硬件环境配置、基于 Azure 生产环境跟踪日志的数据处理流程，以及预测模型训练与调度算法的工程实现细节。
- 第 5 章对 SMore 系统的性能表现展开实验分析。具体地，该章在自建 GPU 集群上，对性能下降预测器的预测准确度、干扰感知调度器在提升 GPU 利用率与保障 SLO 方面的有效性、以及预热模块在降低冷启动开销上的表现进行了多维度的测量。同时，该章将 SMore 与现有的基准调度策略进行了对比，验证了系

统在提升资源利用率的同时能有效控制性能干扰。

- 第 6 章对整篇文章进行了总结，并对未来在异构硬件资源隔离与更细粒度调度策略方面的研究方向进行了概述。

第 2 章 技术背景与研究动机

本章基于绪论研究背景，进一步阐述本文的技术背景与研究动机。首先，第 2.1 节详细介绍服务器无感知计算的基本范式，梳理其在集群资源优化领域的应用现状，并深入剖析当前基于独占式分配策略的 GPU 集群资源管理所面临的瓶颈。随后，第 2.2 节和第 2.3 节分别指出为了实现深度学习训练与服务器无感知推理任务的高效混合部署，当前系统面临的两大核心挑战：一是缺乏有效的资源隔离机制导致的严重性能干扰问题，二是 GPU 环境特有的高冷启动开销对按需弹性能力的制约。这为后续章节中本文提出的系统设计提供了理论依据与优化方向。

2.1 服务器无感知计算与集群资源管理

本节首先对服务器无感知计算范式及其在深度学习领域的应用潜力进行深入剖析。通过解构服务器无感知计算的底层执行机制，阐述其处理突发负载的优势。随后，分析当前主流 GPU 集群资源管理的通用架构与流程，结合深度学习任务的异构特征，进一步挖掘现有粗粒度管理模式在资源利用率与调度灵活性方面存在的本质缺陷。

2.1.1 服务器无感知计算范式

服务器无感知计算代表了一种革命性的云计算范式，其核心理念实现了计算资源供给与应用逻辑开发的彻底解耦。在该范式下，云平台通过控制平面自动接管基础设施的配置、补丁更新、负载均衡及弹性扩缩容等繁杂的运维事务，使得开发者仅需聚焦于以无状态函数为粒度的业务逻辑实现[14]。这种高度抽象化的服务模式极大地降低了云原生应用的开发门槛，并已成功从早期的简单 Web 服务扩展至物联网数据处理、实时流分析及科学计算等复杂场景[55]。与此同时，已有工作对主流商业服务器无感知平台（如 AWS Lambda、Google Cloud Functions、Azure Functions 与阿里云函数计算）的特性与运行时性能进行了系统的实证测量与比较，为理解不同平台在冷启动开销、执行性能等方面的差异提供了重要参考[56]。

在实际执行过程中，服务器无感知计算平台采用事件驱动的执行模型。当请求到达网关时，平台动态调度容器或沙箱环境来承载函数实例。这种机制赋予了系统毫秒级的弹性伸缩能力与按需付费的经济特性，使其特别适合处理具有高度突发性、短周期特征的负载[45,57]。然而，这种特性也引入了特定的生命周期约束：为了最大化

资源复用，函数实例在闲置一段时间后会被回收，导致后续请求触发冷启动，即需要重新经历环境初始化与代码加载过程。

近年来，随着深度学习应用的爆发，学术界与工业界开始尝试将服务器无感知计算集成到分布式 AI 框架中[58]。理论上，深度学习推理任务通常表现为轻量级、无状态且对延迟敏感的 HTTP 请求，与 FaaS 模型天然契合。而深度学习训练任务虽然通常是状态密集型的长时任务，但通过将其分解为细粒度的服务器无感知子任务，可以利用 FaaS 的高并发能力加速数据预处理或梯度计算。更重要的是，这种范式为集群资源的混合部署提供了新的契机：通过将轻量级的服务器无感知推理任务填充进重量级分布式训练任务的资源间隙中，有望打破传统静态分区的资源壁垒，在保证服务质量的前提下显著提升集群的整体资源利用率。

2.1.2　GPU 集群资源管理的局限性

尽管服务器无感知计算在 CPU 和内存资源的优化方面已取得显著进展（例如通过收集碎片化空闲资源进行批处理或机会主义执行[17-18,41]），但针对 GPU 集群的资源管理仍面临严峻挑战，现有的管理架构难以适应深度学习负载的细粒度调度需求。

目前，主流的深度学习集群调度系统（如 Kubernetes, YARN）普遍采用基于预留的粗粒度独占式分配策略。在典型的训练任务工作流中，用户需预先声明所需的 GPU 数量（例如通过 Kubernetes 的 Resource Request 机制），调度器随即将物理 GPU 静态绑定至该任务的 Pod 或容器，直至任务结束。这种占位即锁定的机制虽然简化了隔离管理，但在实际运行中导致了严重的资源碎片化现象。

具体而言，深度学习训练是一个计算与非计算操作交替进行的复杂过程。在每个训练迭代中，GPU 的高负载计算仅发生在前向传播与反向传播阶段；而在数据加载、预处理（通常由 CPU 承担）、参数同步（网络 I/O）以及 Checkpoints 保存（磁盘 I/O）阶段，GPU 计算单元往往处于闲置或低利用率状态。然而，由于独占式策略的限制，这些时间片上的空闲 GPU 资源无法被释放给其他任务使用。

另一方面，对于服务器无感知推理请求而言，其具备高并发与低延迟的要求，通常需要快速获取 GPU 资源进行计算。但在现有的调度框架下，推理任务往往因为无法抢占已被训练任务锁定的 GPU 而被迫排队等待，或者需要集群预留专门的推理节点，造成资源的极大浪费。虽然现有的基于服务器无感知的资源优化工作试图回收集群空闲资源，但它们大多局限于 CPU 和内存维度，缺乏对 GPU 复杂内部状态（如显存上下文、CUDA 流、PCIe 带宽）的感知与管理能力。例如，简单地在训练任务间隙插入推理任务可能引发严重的显存争用或上下文切换开销，进而破坏训练任务的收

敛性能。因此，如何在不干扰训练任务的前提下，实现对 GPU 空闲时间片的精准捕获与推理任务的高效调度，是当前异构集群资源管理亟待解决的难题。

2.2 深度学习集群 GPU 资源利用率低下与混合部署挑战

随着深度学习在自然语言处理、语音识别、推荐系统、图像识别和视频生成等领域的广泛应用，大型研究机构和企业纷纷建立 GPU 集群以满足日益增长的计算需求。然而，随着集群规模的扩大，服务提供商面临着高昂的运营成本。现有研究表明，多租户深度学习集群普遍面临 GPU 资源利用率低下的问题，这不仅浪费了昂贵的计算能力，也限制了整体系统的吞吐量。

导致 GPU 集群利用率低下的核心症结可归纳资源分配机制的僵与分布式 DL 任务执行特性的制约两个维度。

首先，从资源分配机制角度来看，当前主流调度器普遍采用粗粒度的独占式分配策略，通常将 GPU 视为不可分割的单元进行管理。即便是支持 MIG 等硬件分区技术的现代 GPU，其分区粒度依然较粗，难以动态契合多变的任务负载。统计显示，由于深度学习任务通常无法完全饱和 GPU 的流多处理器或显存带宽，约 90% 的被分配 GPU 其实际利用率低于 80%[10]。此外，为了保障分布式训练的同步性，Gang Scheduling 等调度策略往往强制要求资源全量就绪，导致大量资源在任务等待与调度间隙中处于闲置状态。

其次，从任务运行特性角度来看，分布式训练固有的通信与同步机制导致了计算资源的周期性空闲。在典型的云端分布式训练场景中，计算阶段与通信阶段交替进行。由于网络带宽瓶颈或参数同步延迟，GPU 计算单元在每个迭代步中往往存在显著的空闲时间窗。有研究指出，在某些受限于网络的训练任务中，高达 90% 的时间可能消耗在梯度同步等待上，使得昂贵的 GPU 算力沦为空转资源。

为了解决上述问题，现有的优化方案主要分为基于任务的调度优化和基于集群的资源收集两类，但它们在实际应用中均存在局限性。

基于任务的调度优化方案试图通过时间切片或任务打包来提升利用率。例如，Gandiva 通过时间切片和迁移技术支持 GPU 共享；AntMan 引入了机会性低优先级任务以填充空闲周期。然而，这些方案大多局限于特定的负载类型（如仅针对训练任务），或者需要对现有框架进行侵入式修改，且往往无法很好地处理不同规模负载的时间分配公平性问题。

另一类方案尝试利用服务器无感知计算的细粒度特性来收集集群中的空闲资源。

如 UnFaaSener 和 Libra 等工作，成功利用服务器无感知函数收集了空闲的 CPU 和内存资源。然而，此类工作尚未能有效触及 GPU 资源的优化。其核心障碍在于 GPU 的资源隔离机制相对不成熟，且 GPU 任务对上下文环境的依赖较重。

2.2.1　混合部署带来的性能干扰与服务质量目标挑战

基于上述分析，本文提出一种核心洞察：将短生命周期、轻量级且具有突发特征的服务器无感知推理任务与长生命周期的常规服务器式训练任务进行混合部署，是打破资源利用率瓶颈的有效途径。理论上，利用服务器无感知计算的弹性伸缩能力，系统可以精准地利用训练任务留下的算力空隙。然而，这种跨类型负载的混合部署面临着严峻的挑战：由于缺乏有效的硬件级隔离，简单的共置策略极易引发严重的资源争用，导致推理任务违反服务质量目标或训练任务收敛速度下降。

首先，针对异构负载的特性，本架构明确了传统服务器式与服务器无感知负载的定义边界。鉴于推理函数具有执行时间短、突发性强且资源占用适中的特点，本系统将其封装为服务器无感知函数，以满足其严格的截止时间和服务等级目标；相对地，由于训练任务具有长周期、状态密集的特性，系统将其归类为常规服务器式负载，作为集群的基础驻留任务。

为了验证在深度学习集群中引入服务器无感知计算进行混合部署的可行性，并量化其面临的具体挑战，本节构建了详细的动机实验。实验主要包含以下三个步骤。首先对候选的深度学习工作负载进行独占式执行，采集其资源消耗特征。基于 SM 利用率的分布情况，筛选出分别代表低、中、高三种利用率水平的典型训练任务，以及不同计算强度的推理函数，以覆盖多样化的应用场景。随后，在上述选定的训练任务运行过程中，动态注入不同的推理函数请求，模拟真实的混合部署环境，观察其对训练工作负载性能的实际影响。在执行各类工作负载的过程中，实时采集混合部署状态下的运行数据，包括推理函数的尾延迟（Tail Latency）、训练任务的迭代吞吐量以及 GPU 各项硬件指标，用于计算性能退化程度。

在实验负载的选择上，为确保结果的普适性与可复现性，本文选取了八种基于公开数据集的通用深度学习模型，涵盖计算机视觉（Computer Vision, CV）、自然语言处理（Natural Language Processing, NLP）、推荐系统（Advertisment, Ad.）及多任务学习（Multi-task Learning, MTL）四大领域，具体配置如表 2.1 所示。

在评价指标方面，实验选取系统指标与负载指标作为特征分析的基础。系统指标主要包括资源利用率，特别是 GPU 流多处理器的利用率。负载指标则是指与具体任务相关的性能参数。对于推理函数，本文选取第 99 百分位（P99）延迟作为指标；对

表 2.1 经典深度学习工作负载
Table 2.1 Typical Deep Learning Workloads

Model	Type	Dataset
VGG-16[59]	CV	CIFAR100[60]
MobileNet[61]	CV	CIFAR100
DeepViT[62]	CV	ImageNet[63]
ResNet-50[62]	CV	ImageNet
BERT[64]	NLP	CSL[65]
RoBERTa[66]	NLP	SQuAD[67]
DeepFM[68]	Ad.	Criteo
SegNet (MAN)[69]	MTL	NYUv2[70]

于训练负载，则选取完成一个 Epoch 的平均耗时作为基准。训练任务与推理任务的性能退化可通过对比其独占执行与混合部署执行时的表现来计算：

$$\delta = \frac{P_{colo} - P_{solo}}{P_{solo}} \tag{2.1}$$

在公式 2.1 中，δ 表示某一负载的性能退化程度，而 P_{colo} 和 P_{solo} 分别代表该负载在混合部署模式下与独占运行模式下的性能表现。

所有评估实验均在搭载 Nvidia RTX 3090 GPU 的 Ubuntu 20.04 操作系统上部署并执行。为确保实验结果的可靠性，每组混合部署与独占运行实验均持续运行一分钟。各类负载均直接在裸机环境中运行。随后，记录推理函数的平均 P99 延迟退化率以及训练任务的平均执行时间退化率。

为了理解不同负载的资源利用模式，本文首先考察了每个模型在单块 GPU 上独占运行时的 SM 使用率，如图 2.1 所示。DeepViT、ResNet-50、BERT 和 RoBERTa 在训练期间表现出较高的计算强度，SM 峰值使用率超过 70%。相比之下，DeepFM 和 MobileNet 的 GPU 利用率相对较低，主要保持在 30% 以下。对于推理任务，BERT 维持了约 70% 的 SM 利用率，而其他模型大多低于 40%。虽然不同负载的 GPU 使用模式随时间波动，但这种波动幅度通常保持在 20% 以内。这一现象引出了本文的**观察 I:**：尽管深度学习模型表现出多样的资源需求，但同一模型的 SM 利用率在执行期间通常保持在特定范围内。这意味着对于低利用率的训练任务，存在稳定的资源空隙可供利用。

本文采用 P99 延迟作为推理函数的性能指标，采用 Epoch 平均耗时作为训练应用的性能指标。本文选取 DeepFM（低负载）、VGG（中负载）和 RoBERTa（高负载）分别代表不同的利用率水平，以研究混合部署对其性能的影响。图 2.2 可视化了混合

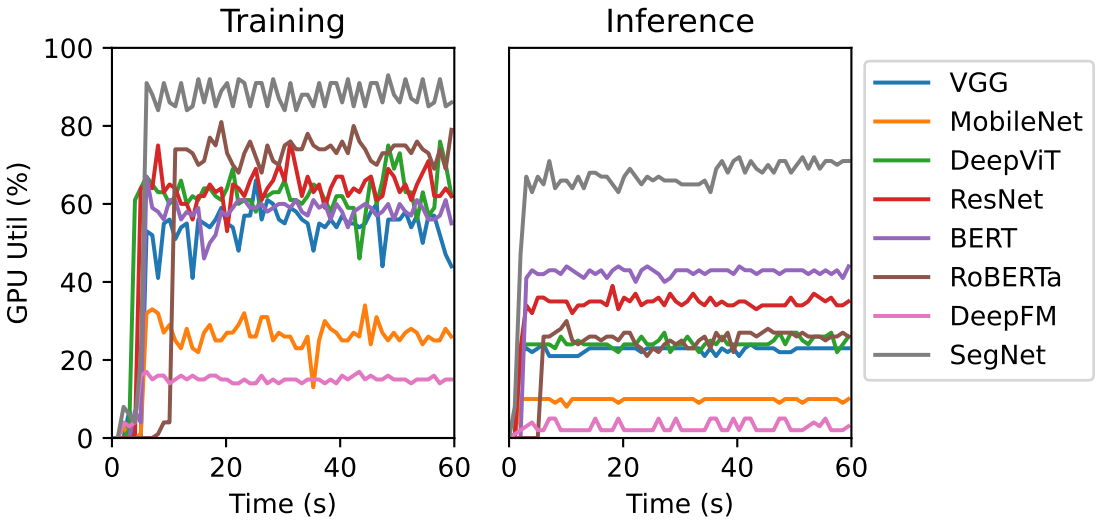

图 2.1　独占运行模式下不同模型的 GPU 利用率
Figure 2.1　GPU Utilization of Different Models Under Solo-Run

部署带来的性能退化程度。X 轴标签表示与这些训练应用共置的特定推理函数。实验表明，与这八种推理函数共置时，DeepFM 训练应用的平均 Epoch 时间并没有显著减慢（退化幅度为 0.4%-18.4%）。MobileNet、DeepFM、RoBERTa 和 DeepViT 等推理负载对 VGG 和 RoBERTa 训练负载的影响极小（低于 10%）。在约半数的情况下，共置函数的 P99 延迟增加量也处于可接受范围内。基于此实验数据，本文得出**观察 II:**：性能退化的程度随推理函数混合部署组合的不同而显著变化。实验证明，在不造成显著性能退化的前提下，进行不同负载的混合部署是完全可行的，关键在于找到合适的组合。

为了进一步研究混合部署特性，本文选取了 DeepFM/MobileNet、VGG/DeepViT 和 RoBERTa/VGG 这几种模型训练与推理负载的组合。随后，本文观察了增加并发推理请求数量对两种负载性能退化的影响。如图 2.3 所示，不同的组合表现出不同程度的性能退化。对于 DeepFM/MobileNet 组合，训练退化呈平稳增长，但在请求数从 7 增加到 8 时出现激增，而推理退化保持在较低的可接受水平（低于 10%）。对于 VGG/DeepViT 和 RoBERTa/VGG 组合，训练和推理负载的退化均随并发请求数的增加而增加。此外，在某些情况下，性能退化超过了 100%，这在实际调度中是不可接受的。数据中观察到的这种上升趋势表明，利用各类回归技术对退化增长进行建模是可行的，这可以为调度决策提供依据。由此，本文归纳得出**观察 III**：混合部署的性能干扰与并发请求数量呈正相关，且该趋势可以通过建模进行预测。这表明存在预测

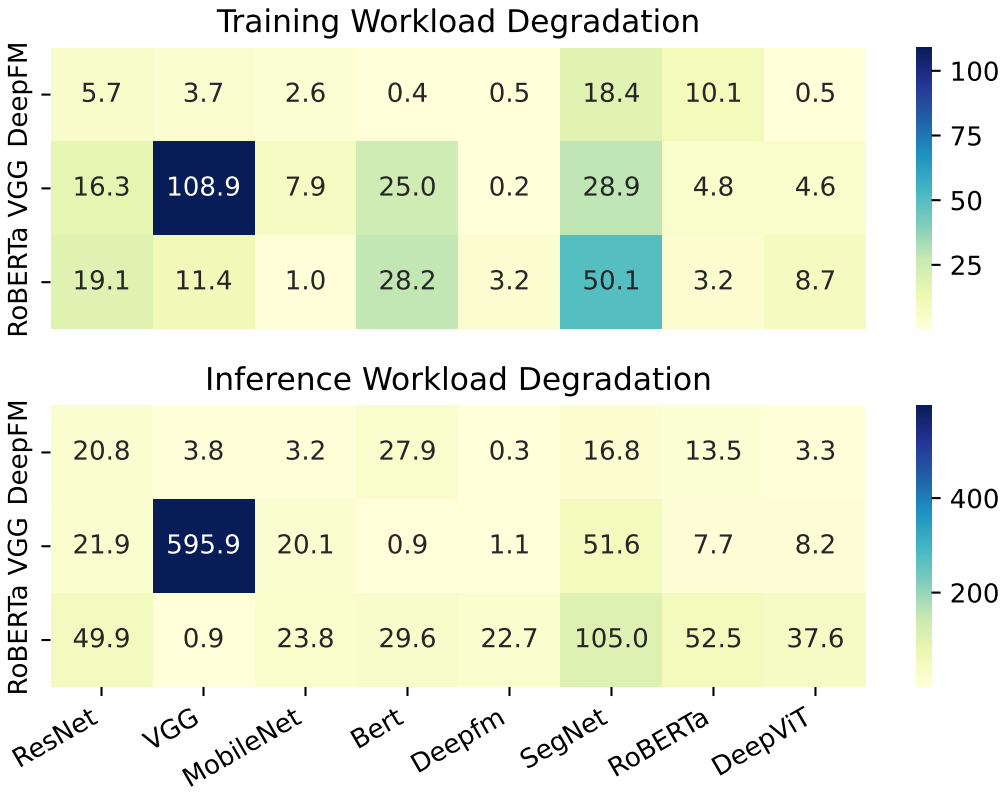

图 2.2 混合部署导致的性能退化
Figure 2.2 Performance Degradation Caused by Co-Location

多推理函数混合部署性能影响的可能性，从而可以规避不可接受的退化。

实验结果揭示了以下关键挑战：

第一，资源争用导致的性能退化。当推理函数与训练任务共置时，两者会竞争 GPU 计算单元、显存带宽以及 PCIe 带宽。如图2.3所示，不合理的共置组合会导致训练任务的迭代时间显著增加，同时也使推理函数的尾延迟恶化。实验观测表明，在某些高负载组合下，性能退化幅度超过 10%，且随着并发推理请求数量的增加，性能干扰呈非线性急剧恶化。这种不可控的性能损失在生产环境中是无法接受的。

第二，服务器无感知函数的服务等级目标难以保障。服务器无感知推理任务通常对延迟高度敏感，有着严格的服务质量目标限制（如截止时间）。在混合部署环境下，由于主负载（训练任务）通常具有更高的优先级，资源的动态争抢可能导致推理函数无法在规定时间内完成执行。这种不确定性使得基于传统规则的调度器难以做出准入决策。

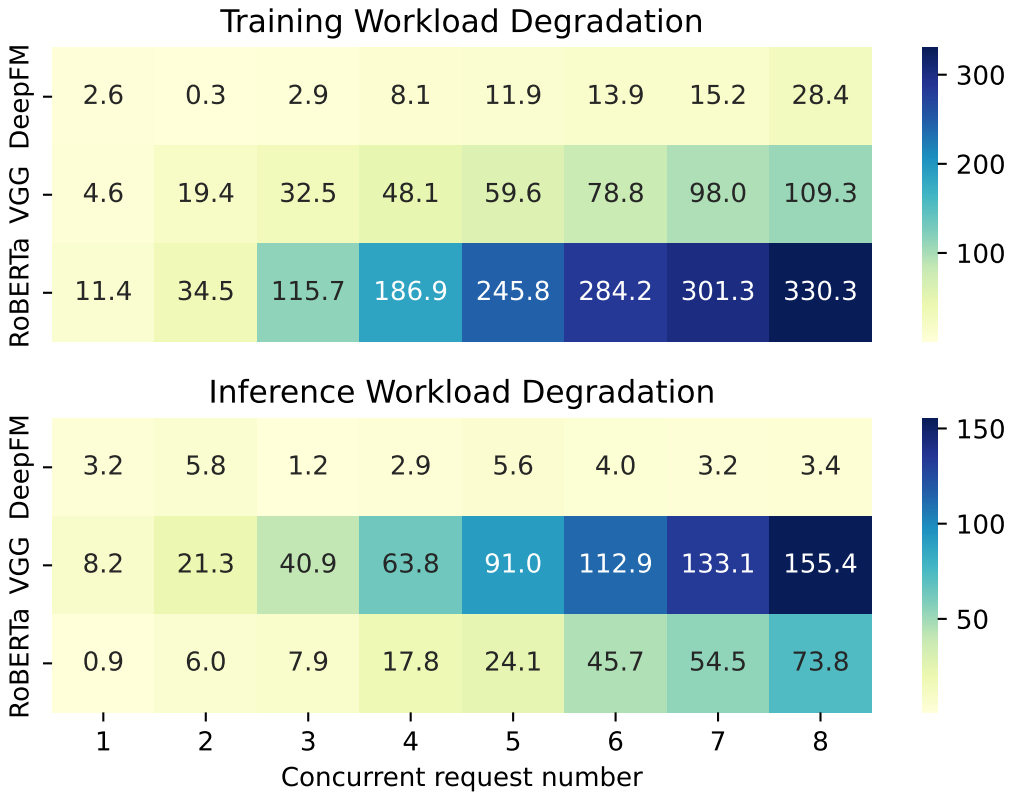

图 2.3　不同请求数量下的混合部署评估
Figure 2.3　Co-Location Evaluation with Different Request Numbers

2.3　GPU 冷启动开销对利用率的制约

除了运行时干扰，冷启动是阻碍利用服务器无感知函数优化 GPU 集群利用率的另一大挑战。与 CPU 函数不同，GPU 函数的初始化过程极为繁重，包括 GPU 上下文（Context）的创建、深度学习框架（如 PyTorch[71]、TensorFlow[72]）的加载以及模型权重的传输与加载。

为了理解 GPU 推理函数引发的冷启动开销，本文评估了若干函数从零开始所需的初始化时间。这包括 GPU 上下文初始化、PyTorch 框架初始化以及模型加载，这些共同构成了冷启动时间。此外，本文评估了初始化完成后这些函数的执行时间，即热启动时间。如表 2.2 所示，RoBERTa 的冷启动时间超过 5 秒，而 BERT 和 DeepViT 的冷启动时间也超过 2 秒。然而，这些函数的典型执行时间不到 100 毫秒。因此，与冷启动相关的初始化开销是巨大的。值得注意的是，尽管 DeepFM 的冷启动时间相对较短（0.6 秒），但这仍比其推理时间高出近两个数量级。这一数据对比鲜明地指出了**观察 IV**：冷启动显著增加了函数的执行延迟，其耗时往往远超函数实际执行时间。

这种巨大的开销可能直接导致服务器无感知函数违反截止时间（Deadline），从而制约了其利用瞬时空闲资源的能力。

表 2.2 不同推理函数的冷启动延迟
Table 2.2 Cold Start Latency of Different Inference Functions

模型 (Model)	热启动耗时 (Warm-start, ms)	冷启动耗时 (Cold-start, s)
VGG-16	3	1.2
MobileNet	9	1.0
DeepViT	11	2.4
ResNet-50	13	1.5
BERT	37	2.6
RoBERTa	10	5.2
DeepFM	8	0.6
SegNet (MAN)	50	1.6

冷启动开销和执行时间之间巨大的差异意味着，如果无法有效管理冷启动，服务器无感知函数在收集空闲 GPU 时间片时，大部分时间将消耗在无意义的初始化上，而非实际计算上。这不仅未能提升有效利用率，反而因为频繁的上下文切换和显存占用，进一步加剧了资源浪费。

2.4 本章小结

本章系统阐述了相关技术背景与研究动机。第 2.1 节首先探讨了服务器无感知计算范式，指出其在打破传统 GPU 集群资源独占限制、提升资源利用率方面的巨大潜力。然而，分析表明，将该范式应用于深度学习集群仍面临两大核心阻碍：

1. **运行时干扰**：如第 2.2 节所述，由于缺乏精细的硬件级资源隔离，混合部署下的资源争用不仅会拖慢训练任务，更会导致推理服务质量严重受损。
2. **高冷启动开销**：第 2.3 节量化分析了 GPU 环境下复杂的初始化流程（如 CUDA 上下文创建与模型加载），证实了现有启动机制难以满足服务器无感知计算对即时弹性的需求。

前文的动机实验为解决上述问题提供了关键的实证依据与设计指引。观察 I 与观察 II 揭示了通过合理筛选混合部署组合将性能退化控制在阈值内的可行性；观察 III 证明了对多推理函数并发时的性能干扰进行建模预测的可能性；而观察 IV 则强调了消除冷启动瓶颈对于降低函数执行延迟的必要性。

基于此，本文确立了下一阶段的核心研究目标：设计一套具备干扰感知能力与高

效冷启动优化机制的服务器无感知 GPU 集群管理系统。该系统的设计将致力于解决以下三大挑战，从而在保障服务质量目标的同时最大化集群资源利用率：(1) 如何在混合部署中有效控制性能退化；(2) 如何保障服务器无感知函数的严格服务等级目标；(3) 如何规避冷启动带来的巨大延迟开销。

第 3 章 架构设计

基于上一章节对深度学习集群中 GPU 资源利用率低下原因的剖析，以及混合部署面临的性能干扰与冷启动挑战的论述，本章提出了一个基于服务器无感知架构的深度学习集群资源管理系统 SMore 。该系统通过设计性能退化预测器、感知退化的调度器以及基于 LS-LSTM 的预热器三大核心组件，在维持常规深度学习任务性能稳定与保障服务器无感知推理函数任务服务质量的同时，有效利用了集群中的空闲 GPU 算力。本章将详细阐述系统的整体架构设计、核心模块的功能定义以及系统的运行工作流程。

3.1 系统架构及工作流程描述

为了应对深度学习集群中混合部署带来的资源争用、服务质量保障困难以及冷启动延迟问题，本工作提出了一个面向异构负载的资源管理架构 SMore 。图 3.1 展示了 SMore 的系统架构组成及工作流程。本节所描述的工作流程由实际深度学习集群中的几个核心目标推导而来：(1) 在不同类型负载混部场景下实现可控的性能退化上界，以支撑多任务 SLO 保障；(2) 将复杂、代价高昂的性能建模与流量建模操作后移到离线阶段，保证在线调度阶段的决策延迟足够低，不引入新的性能瓶颈；(3) 通过对请求流量的前瞻性预测，缓解服务器无感知推理函数的冷启动开销，提高整体资源利用效率。

SMore 架构由三个核心模块组成，分别是性能退化预测器、感知退化的调度器以及基于 LS-LSTM 的预热器。性能退化预测器旨在解决混合部署中的资源竞争导致性能退化难以评估的难题。通过建立服务器无感知推理函数与常规 DL 训练任务的联合性能退化预测模型，该模块能够精准预测特定混合部署组合下的性能退化水平，从而辅助调度器制定包含准入控制在内的精细化决策，避免盲目随意的混合部署导致的性能塌陷。感知退化的调度器是系统的决策核心，旨在解决多任务 SLO 保障的挑战。该调度器采用混合调度策略，根据集群实时负载状态动态选择并准入服务器无感知函数，将其智能调度至合适的训练节点进行混合部署，在最大化资源利用率的同时将性能退化控制在用户可接受的阈值范围内。预热器模块则针对服务器无感知计算固有的冷启动问题设计。通过应用基于长短期记忆网络 LS-LSTM 的时间序列预测模型，该模块能够准确预测下一周期的请求流量，并据此执行模型预加载或卸载操作，

显著降低了冷启动延迟并减少了无效的资源占用。

SMore 的整体工作流程如图 3.1 所示，在逻辑上划分为离线画像与在线服务两个阶段。

在离线画像阶段，系统的主要目标是构建性能退化预测模型与请求流量预测模型，为在线服务阶段决策支撑。首先，系统对各类型负载进行独占执行条件下的基准测试，收集其原始性能指标画像（步骤❶）。接着，为了降低全空间搜索的成本，系统随机采样部分混合部署组合（步骤❷）生成训练样本，用于训练一个初始的双路混合部署性能退化模型（步骤❸）。基于该初始模型，系统进一步推演并预测剩余未采样组合的性能退化情况（步骤❹），从而生成一张完整的性能退化查询表，以支持在线阶段的毫秒级快速查询。与此同时，系统利用收集到的生产环境历史踪迹，对预热器模块中的 LS-LSTM 模型进行离线训练（步骤❸），使其具备初步的流量感知能力。

在在线服务阶段，系统根据实时请求流进行动态调度与模型迭代。首先，用户向调度器提交服务器无感知推理请求（步骤❺）。在此过程中，预热器模块持续收集实际到达的请求数据，并以特定时间间隔周期性更新 LS-LSTM 模型（步骤❻），据此预测下一周期的流量趋势并提前预加载模型环境（步骤❼），以消除冷启动开销。同时，调度器通过监控器实时采集集群内的资源使用率与任务执行状态，周期性更新全局集群状态视图（步骤❽）。在接收到请求后，调度器查询性能退化预测模块，获取候选混合部署方案的预期性能退化值（步骤❾）。基于预测结果与当前集群状态，调度器执行准入控制与资源分配算法，将服务器无感知推理函数调度至最优的 GPU 节点与常规 DL 训练任务共存（步骤❿）。当任务执行完毕后，监控器会记录实际观测到的性能退化数据，并将这些真实样本反馈至系统数据库，用于后续重新训练和校准预测模型（步骤⓫），从而实现系统精度的持续自适应进化。

上述工作流程的合理性与正确性主要体现在时效性解耦与误差自修正机制两方面。首先，通过将高计算开销的画像构建与模型训练（步骤❶-❹）解耦至离线阶段，流程保证了在线调度路径（步骤❾-❿）仅需执行低开销的查表与推断操作，从而证明了该流程能够满足在线推理任务对毫秒级调度的时效性要求。其次，流程中引入的反馈回路（步骤⓫）为系统的正确性提供了理论保障。通过将实际观测偏差回传并触发模型微调，系统能够有效修正因工作负载漂移或初始采样不足导致的预测误差。这种持续的自适应进化机制确保了系统在长期运行中，其调度决策会逐渐收敛至最优解，从而证明了该工作流程在动态环境下的鲁棒性与正确性。

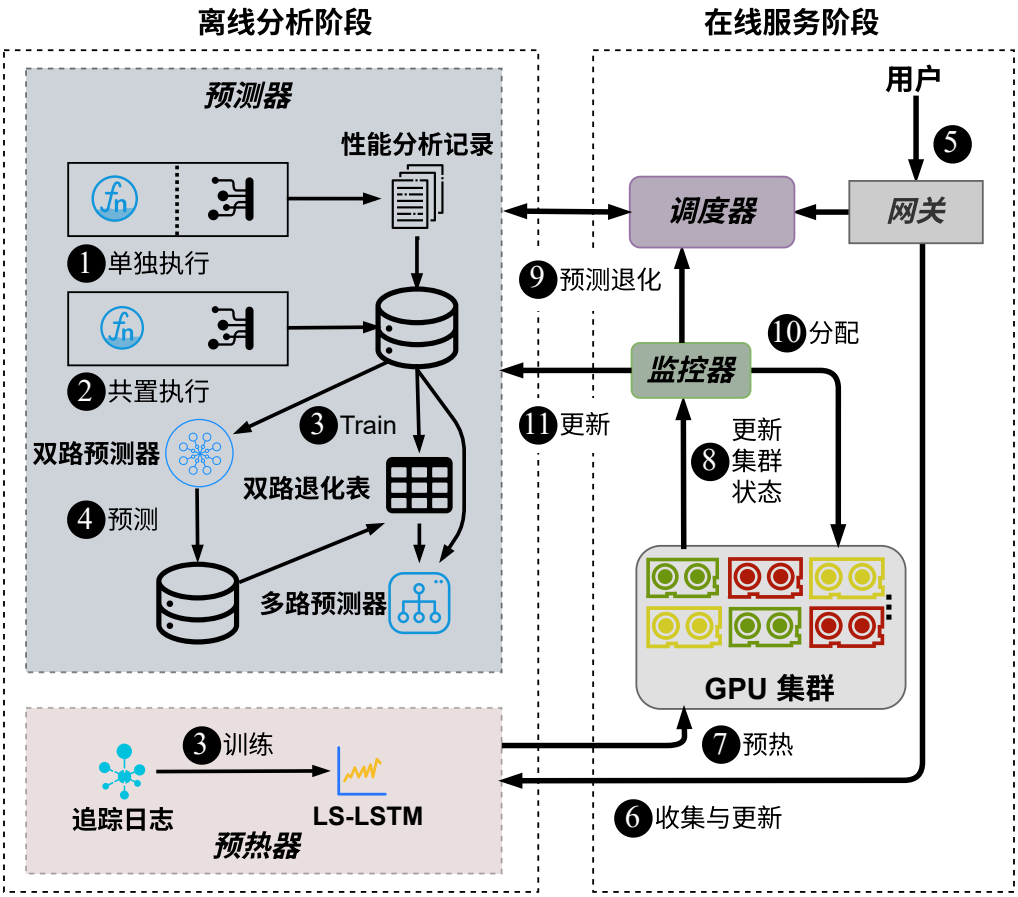

图 3.1 SMore 的架构概览与工作流程图
Figure 3.1 The Overview and Workflow of SMore

3.2　混合部署性能退化预测器

3.2.1　设计概览

为了在保障服务质量的前提下最大化资源利用率，本文提出了一种分层式的混合部署性能退化预测器。该模块旨在量化不同负载组合下的资源争用导致的性能退化程度，其核心由两个级联的子预测器组成：**双路基准预测器**与**多路聚合预测器**。

本文将性能退化预测问题形式化为一个回归问题。设 T 表示驻留的常规服务器式训练任务，$S = \{s_1, s_2, \ldots, s_n\}$ 表示待调度的 N 个服务器无感知推理函数集合。预测目标是求解映射函数 $f : \mathcal{T} \times \mathcal{S}^N \to \mathbb{R}^{N+1}$，输出训练任务的性能退化率 δ_T 以及各推理函数的性能退化率 $\{\delta_{s_1}, \ldots, \delta_{s_n}\}$。

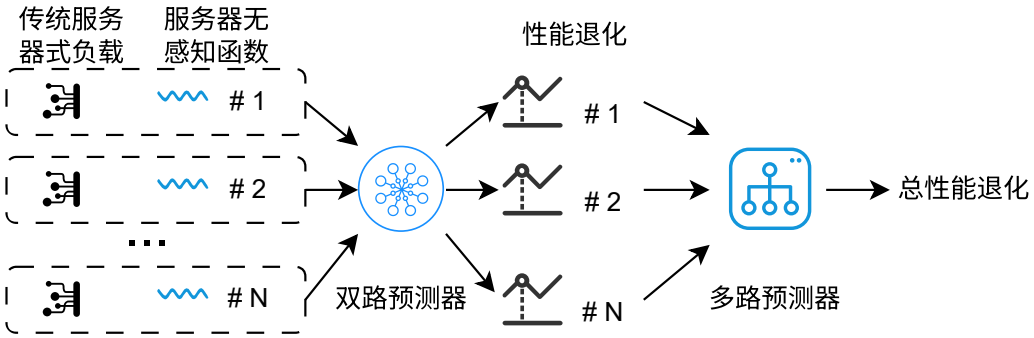

图 3.2　预测器的工作流程图
Figure 3.2　The Workflow of the Predictor

系统采用离线训练与在线增量更新结合的混合学习机制。如图 3.2 所示，预测流程分为两步：

1. **基准预测**：将 N 路混合部署问题分解为 N 个独立的两两子问题。对于任意 $s_j \in S$，通过双路预测器估算 (T, s_j) 组合下的基准退化值。
2. **聚合预测**：将上述基准值输入多路预测器，通过加权聚合模型计算多任务并发下的非线性干扰叠加效应，得到最终的性能退化预测值。

3.2.2　双路混合部署画像与基础模型构建

双路混合部署不仅是多路混合部署的基础组成单元，更是理解异构负载间资源争用机理的关键切入点。本节详细阐述如何构建常规服务器式负载 T_i 与单个服务器无感知函数 s_j 共置时的基准性能退化预测模型。

在实验设计上，本文搭建了一个受控的沙箱环境。当常规服务器式深度学习训练任务在 GPU 上持续执行时，系统会动态调度不同的服务器无感知深度学习推理函数

表 3.1 选取的深度学习模型特征
Table 3.1 Selected Deep Learning Model Features

特征 (Features)	描述 (Description)
FLOPS	浮点运算次数 (Number of floating-point operations)
Params	参数数量 (Number of Parameters)
Memory	模型占用的显存 (GPU Memory used by model)
Activations	激活函数数量 (Number of activation functions)
Num. Conv	卷积层数量 (Number of convolution layers)
Num. Linear	线性层数量 (Number of linear layers)
Batch Size	批处理大小 (Size of mini-batch)
Num. Norm	归一化层数量 (Number of Normalization layers)
Num. ReLU	ReLU 层数量 (Number of ReLU layers)
Num. Embed	嵌入层数量 (Number of Embedding layers)
Num. Pool	池化层数量 (Number of Pooling layers)
Num. Drop	Dropout 层数量 (Number of Dropout layers)

至同一物理 GPU 节点，并进行毫秒级的持续监测。在此期间，系统会屏蔽掉冷启动的影响——即仅统计函数实例环境初始化完成后的纯计算与通信阶段。这是因为冷启动主要受限于磁盘 I/O 读取效率与深度学习框架和模型的初始化开销，这属于另一类优化问题，将在第 3.4 节中详细讨论。而本节重点关注的是混合部署场景下在 GPU 侧的运行时资源竞争，主要包括对 GPU 流处理器和显存带宽的争用，以及由此引发的性能退化。

3.2.2.1 多维特征工程与向量化表示

深度学习任务的资源消耗模式具有高度的异构性。为了精准捕捉不同任务对 GPU 资源的竞争模式，本文设计了一套包含动态运行时指标与静态图结构指标的多维特征提取方案。

首先，在**系统层级**，本文通过 NVIDIA 官方提供的 NVML 工具实时采集 GPU 的流多处理器（GPU SM）利用率和显存（GPU Memory）占用率。这两个指标共同构成了资源竞争的底层背景：SM 利用率反映了计算单元的实际负载饱和度，而显存占用则直接决定了上下文切换的额外开销以及发生内存溢出（Out-of-Memory, OOM）的风险程度。

其次，在**任务层级**，单纯的系统指标不足以描述复杂的深度学习模型特性。例如，同样是高利用率，密集型矩阵运算与稀疏嵌入查找对缓存系统的压力截然不同。因此，本文定义了特征提取函数 $\phi(\cdot)$，提取表 3.1 所示的 12 维关键特征向量 $\mathbf{x} \in \mathbb{R}^{12}$。

- **计算强度特征**：包括浮点运算次数（FLOPS）与参数量（Params）。FLOPS 直接衡量了模型的算力需求，而参数量则反映了模型在训练或推理阶段的数据搬运压力。需要指出的是，由于不同硬件架构（如 Tensor Core 的使用）与深度学习框架（PyTorch, TensorFlow）的算子实现差异，理论 FLOPS 往往难以反映真实负载。因此，本文采用基于 TorchProfiler 的单次迭代分析法，通过运行一次完整的前向与反向传播来获取精确的硬件相关指标。
- **显存特征**：显存占用（Memory）不仅包含模型权重，还包含中间激活值（Activations）。高显存占用意味着在混合部署时更容易触发缺页中断或强制发生主存交换，从而引发剧烈的性能退化。
- **拓扑结构特征**：包括卷积层（Num. Conv）、线性层（Num. Linear）、嵌入层（Num. Embed）等的数量。这些特征隐含了模型的计算模式。例如，卷积层密集的 CNN 模型通常是计算受限的，容易在 CUDA Core 上产生竞争；而嵌入层密集的推荐模型通常是显存带宽受限的。

对于一个双路混合部署组合 (T_i, s_j)，本文将两者的特征向量进行拼接，构建模型的联合输入向量 $\mathbf{X}_{ij}$：

$$\mathbf{X}_{ij} = \phi(T_i) \oplus \phi(s_j) \in \mathbb{R}^{24} \tag{3.1}$$

其中 $\oplus$ 表示向量拼接操作。$\mathbf{X}_{ij}$ 在物理意义上代表了两个异构任务在共享资源环境下发生的资源竞争状态。

3.2.2.2　基于随机森林的非线性回归建模

任务间的干扰关系通常是非线性的。例如，当显存带宽占用未达到总线瓶颈时，两个任务可能相安无事；一旦突破阈值，性能将呈现断崖式下跌。传统的线性回归模型难以拟合这种阈值效应。因此，本研究在对比了不同回归模型后，最终选择随机森林（Random Forest）作为双路预测器的核心算法。

选择随机森林主要基于以下考量：

1. **非线性拟合能力**：基于决策树的集成方法能够自然地对特征空间进行划分，有效捕捉上述的阈值特征（例如，若 T_i 显存 > 10GB 且 s_j 显存 > 10GB，则退化剧增）。
2. **对小样本的鲁棒性**：相比于深度神经网络，随机森林在有限的画像样本下不易发生过拟合，且对特征的归一化不敏感。
3. **可解释性**：通过特征重要性分析，本文可以验证模型是否正确学习到了关键干扰源（如显存带宽竞争）。

假设随机森林包含 B 棵决策树 $\{h_1, h_2, \ldots, h_B\}$，模型通过 Bagging 策略对输入 $\mathbf{X}_{ij}$ 进行预测，最终的基准性能退化值 $\hat{D}_{ij}$ 为所有决策树输出的均值：

$$\hat{D}_{ij} = \frac{1}{B} \sum_{b=1}^{B} h_b(\mathbf{X}_{ij}) \tag{3.2}$$

3.2.2.3 模型评估指标与物理含义

在混合部署场景中，性能退化的分布往往具有长尾特性，且存在异方差性。例如，一个 P99 延迟维 10ms 的服务器无感知推理函数任务退化至 20ms（增加 10ms），其性能损失为 100%；而一个耗时 100s 的训练迭代退化至 100.1s（增加 100ms），其损失仅为 0.1%。若使用普通的均方误差 MSE，模型会过度关注长耗时常规服务器式任务的绝对误差，而忽略了短耗时服务器无感知函数任务的相对退化。

为了解决这一问题，本文采用均方根对数误差 RMSLE 作为损失函数与核心评估指标。其数学定义如下：

$$\mathcal{L}_{RMSLE} = \sqrt{\frac{1}{M} \sum_{k=1}^{M} (\log(y_k + 1) - \log(\hat{y}_k + 1))^2} \tag{3.3}$$

其中 M 为测试样本数量，y_k 为真实测量的退化值，$\hat{y}_k$ 为模型预测值。RMSLE 的引入具有双重优势：

- **关注相对误差**：通过对数变换，RMSLE 实际上是在衡量预测值与真实值比率的偏差，这与服务等级目标 SLO 通常以百分比定义的习惯相一致。
- **平滑离群点**：它有效抑制了训练数据中极端退化值对模型训练梯度的干扰，使模型在低退化区间和高退化区间均能保持稳定的预测能力。

3.2.3 多路混合部署预测与在线校准

在实际的生产环境中，为了极致地压榨 GPU 算力，调度器往往需要将多个服务器无感知函数同时调度至同一个运行着训练任务的节点上。然而，这种多路混合部署带来了巨大的状态空间挑战。假设服务器无感知函数库的大小为 N，并发度为 k，则潜在的组合数量约为 $O(N^k)$。面对这种指数级的组合爆炸，试图通过离线采样来覆盖所有可能的混合部署场景不仅成本高昂，在计算上也是不可行的。为此，本文基于复杂干扰可分解为基准干扰叠加的假设，提出了一种基于线性加权的聚合预测模型。该模型的核心思想是将复杂的多体资源争用问题降维分解为多个可观测的双路交互问题，并通过在线增量学习动态校准权重参数，以修正线性假设带来的偏差。

3.2.3.1　干扰叠加模型与非线性修正

本文将多路混合部署视为多个双路基础干扰的矢量叠加。对于一个驻留的常规服务器式训练负载 T_i 与一组并发的服务器无感知函数 $S_{set} = \{s_j | j \in \mathcal{J}\}$（其中 $|S_{set}| = k$）构成的混合场景，系统的资源竞争格局变得异常复杂。在这种场景下，训练任务 T_i 的性能退化并非简单的线性累加，因为硬件资源（如 PCIe 带宽、L2 缓存、DRAM 带宽）存在物理瓶颈。当多个任务的总需求未达到瓶颈时，干扰可能呈现次线性增长；而一旦突破瓶颈，系统可能陷入严重的资源颠簸，导致超线性的性能雪崩。

为了数学化地表征这一过程，本文将 T_i 的总性能退化 $Deg_i^{(k)}$ 建模为所有双路基准退化值的加权和，如公式 3.4 所示：

$$Deg_i^{(k)} = \sum_{j \in S_{set}} \gamma^{(k)}(T_i, s_j) \cdot \hat{D}_{ij} \tag{3.4}$$

在此公式中，$\hat{D}_{ij}$ 是通过前述双路预测器获得的基准退化值，代表了单一干扰源的理论强度。$\gamma^{(k)}(T_i, s_j)$ 则是引入的干扰权重系数，它特定于模型组合 (T_i, s_j) 以及当前的并发度 k。该系数充当了非线性修正因子的角色：当 $\gamma < 1$ 时，意味着存在资源掩蔽效应，即多个小任务利用了互不冲突的空闲资源槽；当 $\gamma > 1$ 时，则捕捉了资源争用的边际效应递增现象。通过这种加权机制，模型能够在保持计算简便性的同时，灵活拟合复杂的非线性干扰曲面。

对于服务器无感知函数侧的性能预测，问题则更为复杂。某一函数实例 s_j 不仅会受到较大规模训练任务 T_i 的纵向资源压制，还会受到同一批次中其他并发函数 $s_m | m \in S_{set}, m \neq j$ 的横向资源干扰。为区分并解耦这两类性质不同的干扰来源，本文构建了如公式 3.5 所示的分层预测模型：

$$Deg_j^{(k)} = \alpha^{(k)}(T_i, s_j) \cdot \hat{D}_{ji} + \sum_{m \in S_{set}, m \neq j} \beta^{(k)}(s_j, s_m) \cdot \hat{D}_{jm} \tag{3.5}$$

其中，第一项代表主任务带来的纵向干扰，由权重 $\alpha^{(k)}$ 调节；第二项代表兄弟函数带来的横向干扰，由权重 $\beta^{(k)}$ 调节。例如，在一个显存带宽受限的场景中，如果 T_i 占用了 80% 的带宽，那么 s_j 受到的纵向压制将占主导地位，此时模型倾向于赋予 $\alpha^{(k)}$ 更大的值；反之，若 T_i 处于计算密集阶段而留出了带宽，但并发的 s_m 却频繁进行数据搬运，则横向干扰将成为瓶颈，$\beta^{(k)}$ 的权重随之增加。这种分权设计使得预测器能够细粒度地感知不同来源的压力，从而做出更精准的预测。

3.2.3.2 基于梯度的在线权重自适应更新

尽管本文构建了结构化的叠加模型，但预定义的静态权重难以适应生产环境中瞬息万变的负载特征与硬件状态。为了赋予系统持续进化的能力，本文设计了一套基于梯度的在线权重更新机制。在系统初始启动阶段，由于缺乏先验知识，本文将所有权重系数向量 $\mathbf{W} = \{\gamma, \alpha, \beta\}$ 初始化为 1.0，此时模型退化为朴素的线性叠加模型。

随着系统的持续运行，监控模块会源源不断地采集真实环境下的性能数据，为模型更新提供数据支持。对于每一次实际发生的混合部署事件，本文都能获得一组预测值与真实值的数据对 (Deg^{pred}, Deg^{real})。为了最小化预测误差，本文将参数校准过程形式化为一个在线凸优化问题。定义时刻 t 的瞬时损失函数 $J(\mathbf{W})$ 为预测误差的平方的一半。

$$J(\mathbf{W}) = \frac{1}{2}(Deg^{real} - Deg^{pred}(\mathbf{W}))^2 \tag{3.6}$$

基于随机梯度下降 SGD 的思想，系统沿着损失函数的负梯度方向微调权重参数，以期在后续预测中减小误差。

以训练任务的干扰权重 $\gamma(T_i, s_j)$ 为例，其迭代更新规则推导如下：

$$\gamma_{t+1} = \gamma_t - \eta \cdot \frac{\partial J}{\partial \gamma} = \gamma_t + \eta \cdot (Deg_i^{real} - Deg_i^{pred}) \cdot \hat{D}_{ij} \tag{3.7}$$

其中 η 为学习率，控制着权重更新的步长。该更新公式具有清晰的物理直觉：$(Deg_i^{real} - Deg_i^{pred})$ 代表了预测的残差符号与幅度，而 $\hat{D}_{ij}$ 则作为一种归因因子——如果某个服务器无感知函数 s_j 的基准干扰 $\hat{D}_{ij}$ 很大，那么它应对总误差承担更大的责任，因此其对应的权重 γ 会受到更大幅度的修正。通过这种闭环反馈机制，预测器能够自动适应硬件老化、驱动更新或模型结构微调带来的环境漂移，随着样本量的积累，模型的预测精度将不断逼近真实的物理干扰规律，实现系统的自适应演进。

3.3 感知性能退化的调度器

基于上一节所述的性能预测模块，系统已经具备在混合部署场景下对服务器无感知函数与常规服务器式负载之间潜在干扰风险进行量化评估的能力。然而，仅具备预测能力并不意味着已经实现高效的资源管理。预测模型仅为系统提供了观测与表征的手段，要将这种洞察力转化为实际的集群性能收益，还需要构建一个能够实时制定调度决策的控制中枢。为在保障已准入服务器无感知函数服务等级目标的同时，严格确保常规服务器式训练任务的训练吞吐量不受到显著影响，本文提出了一种感知退化的调度策略。该策略的核心思想是，将异构负载的调度建模为在非确定性环境

中进行多目标权衡与优化的问题：在不突破既有长周期训练任务性能底线的前提下，最大化挖掘集群中的碎片化空闲资源，以承载更多短周期推理请求。

本文将该调度问题形式化为一个带约束的多目标优化问题。假设集群当前由 M 个异构的 GPU 节点组成集合 $\mathcal{G} = \{g_1, g_2, \ldots, g_M\}$，待调度队列中积压了 N 个不同类型的服务器无感知请求 $\mathcal{F} = \{f_1, f_2, \ldots, f_N\}$。调度器的核心目标是寻找一个映射关系 $\Pi : \mathcal{F} \to \mathcal{G} \cup \{\perp\}$（其中 $\perp$ 表示拒绝或推迟），使得在满足以下严格约束的前提下，最大化集群的整体吞吐量与资源利用效能：

1. **硬资源约束**：混合部署后的显存（GPU Memory）占用总量严格不得超过物理上限，以避免因内存溢出 OOM 导致的进程崩溃。
2. **软性能约束**：训练任务的性能退化率需严格控制在管理员设定的容忍阈值 θ_{train} 以内，同时推理函数的端到端延迟必须满足其 SLO_{lat} 要求，任何违反上述性能指标的调度决策均被视为无效。

如图 3.3 所示，该策略包含三个紧密耦合的核心阶段：首先是通过**基于干扰性价比的优先级排序**解决请求间的竞争问题，其次是利用**高效的候选节点采样**解决大规模集群的状态搜索问题，最后是通过**干扰感知的准入与择优**实现精细化的节点绑定。

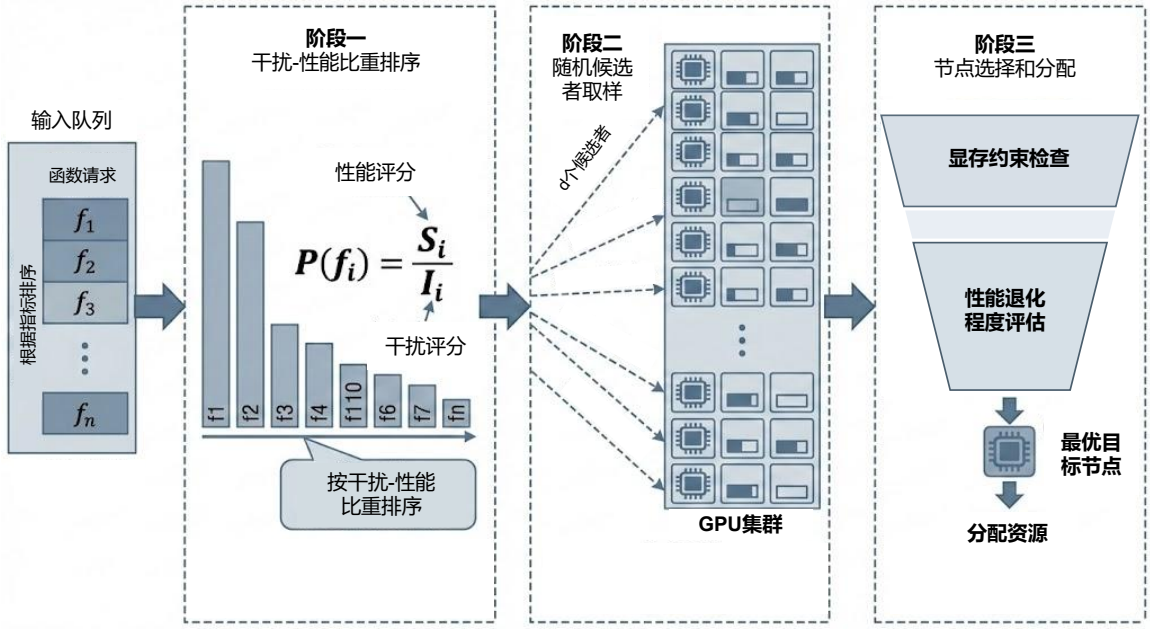

图 3.3　调度流程图
Figure 3.3　The Workflow of Scheduling

3.3.1 基于干扰性价比的优先级队列

在深度学习生产环境中，服务器无感知推理请求的到达往往呈现出高并发与突发性的特征。当请求到达速率超过调度器的瞬时处理能力时，系统维护的待调度队列长度将显著增加。此时，决定优先为哪个函数分配稀缺的 GPU 资源对于集群整体效率至关重要。传统的先来先服务（First-Come-First-Served, FCFS）策略虽然保证了公平性，但忽略了任务对资源的异构需求与干扰敏感度差异。例如，若队列头部存在一个计算密集且对干扰极不敏感的大型模型（如 BERT-Large）请求，而其后跟随多个轻量级模型（如 MobileNet）请求，FCFS 策略可能导致由于头部任务长时间占用调度器或资源块，进而引发严重的队头阻塞（Head-of-Line Blocking）现象，导致后续大量短服务器无感知函数任务的 SLO 违约。

为此，本文引入一种基于干扰性价比的贪心启发式策略，旨在优先处理那些能够带来最大资源利用率增益且引发最小系统扰动的任务。对于任意待调度的函数请求 f_i，系统利用历史画像数据估算其在混合部署环境下的行为特征，并计算优先级评分 $P(f_i)$：

$$P(f_i) = \frac{\mathbb{E}[U_{gpu}(f_i)]}{\mathbb{E}[\Delta Lat(f_i)] + \epsilon} \tag{3.8}$$

其中，分子 $\mathbb{E}[U_{gpu}(f_i)]$ 表示函数 f_i 在执行期间预期的平均 GPU SM 利用率增量，这反映了该任务对计算资源的填充能力；分母 $\mathbb{E}[\Delta Lat(f_i)]$ 表示该函数在一般性混合部署场景下预期的归一化延迟退化幅度，反映了其对干扰的敏感程度；ϵ 是一个极小的平滑常数（例如 10^{-5}），用于防止分母为零。

该公式的物理含义在于量化单位性能代价下的资源收益。评分较高的任务通常具备高资源利用且低敏感的鲁棒特性，优先调度此类任务能够快速填充集群的资源空隙而不容易触发性能违规。通过维护一个基于 $P(f_i)$ 的最大堆，调度器能够动态调整执行顺序，从宏观上最大化集群的有效产出。

3.3.2 基于随机采样的候选集构建

在确定了优化的调度顺序后，下一步是为当前的最高优先级请求 f_i 在庞大的集群中选择最佳的物理承载节点 g_j。在一个包含数千个 GPU 节点的大规模集群中，若采用全局扫描策略，其时间复杂度为 $O(M)$。这意味着调度器必须查询每一个节点的状态，这不仅会带来显著的调度延迟，还可能因为状态信息的同步滞后导致陈旧状态问题，即调度器基于过期的负载信息做出了决策。

为了在调度质量与决策延迟之间取得最佳平衡，本文摒弃了全量扫描，转而采用

一种基于“d-路随机采样（Power-of-d-Choices）”的轻量级策略来构建候选集。具体而言，调度器不遍历整个资源池 $\mathcal{G}$，而是依据均匀分布随机采样 d 个节点（通常取 $d = 2$ 或 $d = 5$）构成候选集 $\mathcal{G}_{cand}$。这一策略源于负载均衡领域的经典理论，已被证明能够在极低的开销下，以指数级的概率避免选择到负载最重的最差节点。

通过将搜索空间从 M 缩减至常数 d，本文将调度算法的时间复杂度降低至 $O(1)$，这使得系统具备了极好的水平扩展性。此外，随机采样有效地分散了调度压力，避免了集中式调度中常见的羊群效应，即多个并发调度器同时争抢同一个最佳节点导致资源竞争加剧。

3.3.3　干扰感知的准入与择优

在构建了包含 d 个节点的候选集 $\mathcal{G}_{cand}$ 后，系统需要针对其中每个节点 g_j 执行精细化的两阶段评估机制，以确定是否存在满足所有约束的可行解，并在可行解中择优。

第一阶段：硬资源约束检查。显存是深度学习任务中最关键且不可压缩的资源。一旦显存分配超过物理上限，将直接触发 CUDA 驱动的 OOM 错误，导致正在运行的训练任务崩溃，这是绝对不允许发生的。因此，系统首先检查显存容量约束。令 $Mem_{used}(g_j)$ 为节点当前已被占用的显存量，$Mem_{req}(f_i)$ 为函数请求预测的显存峰值需求，Mem_{cap} 为 GPU 的物理显存总量。节点必须满足：

$$Mem_{used}(g_j) + Mem_{req}(f_i) \leq \sigma \cdot Mem_{cap} \tag{3.9}$$

其中 $\sigma \in (0, 1)$（例如取 0.95）是一个安全阈值系数。引入 σ 的目的在于为显存碎片以及 CUDA 上下文开销预留缓冲空间，防止因边缘效应导致的分配失败。

第二阶段：软性能准入控制与多目标择优。对于通过硬约束检查的节点，系统调用第 3.2 节所述的性能预测器，计算混合部署后的预期干扰状态。设 $\hat{D}_{train}(g_j, f_i)$ 为预测的后台训练任务性能退化率，$\hat{D}_{func}(g_j, f_i)$ 为预测的推理函数性能退化率。为了保障 SLO，本文定义严格的准入控制条件（Admission Control）：

$$\text{Admit}(g_j, f_i) \iff \begin{cases} \hat{D}_{train}(g_j, f_i) \leq \theta_{train}^{max} \\ \hat{Lat}_{base}(f_i) \cdot (1 + \hat{D}_{func}(g_j, f_i)) \leq SLO_{deadline} \end{cases} \tag{3.10}$$

其中 $\hat{Lat}_{base}$ 是函数在独占资源下的基准执行时间。该逻辑构成了系统的熔断机制：如果某个节点上的混合部署预测结果显示会导致训练任务严重拖慢（超过 θ_{train}^{max}）或者推理请求超时（超过 $SLO_{deadline}$），则该节点被判定为不可行。若候选集中所有节点均不可行，请求将被拒绝并重新压入队列等待，直到资源释放。

最后，在所有满足上述准入条件的可行节点集合 FeasibleSet 中，本文采用加权最小化策略选择最终的目标节点 g_{target}：

$$g_{target} = \arg \min_{g_j \in \text{FeasibleSet}} \left(\lambda \cdot \hat{D}_{train}(g_j, f_i) + (1 - \lambda) \cdot \hat{D}_{func}(g_j, f_i) \right) \tag{3.11}$$

其中 $\lambda \in [0, 1]$ 是一个可配置的偏好权重因子。当集群更看重护航任务的稳定性时，可调大 λ；反之，若更关注推理服务的响应速度，则调小 λ。这种设计使得调度策略能够灵活适应不同的业务场景需求，实现了训练与推理性能之间的最优权衡。

算法 3.1 服务器无感知函数节点选择策略

Input: 待调度请求 req, 全局资源池 $\mathcal{G}$, 采样数 d, 快速模式标志 $fast\_mode$
Output: 目标节点 g_{target} 或 $\emptyset$
/* 阶段二：随机节点选择 */
$\mathcal{G}_{cand} \leftarrow$ SampleNodes($\mathcal{G}$, d)
$\mathcal{S}_{feasible} \leftarrow \emptyset$ // 构建候选集
/* 阶段三：准入控制与决策 */
for $g_j \in \mathcal{G}_{cand}$ **do**
  // 步骤 3.1：资源硬约束 (公式 3.9)
  if not CheckHardConstraints(g_j, req) **then**
    continue
  end
  // 步骤 3.2：性能退化预测
  $\langle \hat{D}_{train}, \hat{D}_{func} \rangle \leftarrow$ PredictDegradation(g_j, req)
  // 步骤 3.3：软约束 (公式 3.10)
  if CheckSoftConstraints($g_j, \hat{D}_{train}, \hat{D}_{func}$) **then**
    if $fast\_mode$ **then**
      return g_j // 首次适配
    end
    $score \leftarrow \lambda \cdot \hat{D}_{train} + (1 - \lambda) \cdot \hat{D}_{func}$
    $\mathcal{S}_{feasible} \leftarrow \mathcal{S}_{feasible} \cup \{(g_j, score)\}$
  end
end
// 步骤 3.4：Optimal Selection
if $\mathcal{S}_{feasible} \neq \emptyset$ **then**
  $g_{target} \leftarrow \arg\min_{(g,s) \in \mathcal{S}_{feasible}}(s)$
  return g_{target}
else
  return $\emptyset$ // 无满足条件节点
end

算法 3.1 形式化地描述了 SMore 的核心资源分配流程。当服务器无感知请求 Req 到达时，调度器首先依据公式 3.8 动态计算其优先级，以缓解异构负载下的队头阻塞

问题。随后的调度决策遵循随机采样、约束过滤和效用择优的三阶段范式：首先，算法从全局资源池中执行 d-路随机采样构建候选集，这种无状态（Stateless）的采样策略有效规避了高并发场景下维护全局状态一致性带来的同步开销。其次，候选节点需经过双重准入测试：第一重为硬资源约束检查（步骤 3.1），旨在通过物理显存上限快速剔除不可行解，防止 OOM 异常；第二重为软性能约束准入（步骤 3.3），利用预测模型量化量化混合部署后的干扰风险，仅保留满足训练任务退化阈值 θ_{train}^{max} 及函数 $SLO_{deadline}$ 的节点。最后，算法根据系统负载采用自适应决策策略：在高负载模式下采用首次适配（First-Fit ）策略，一旦发现满足双重约束的节点即刻返回，以最小化调度时延；而在常规模式下采用最佳适配（Best-Fit ）策略，遍历所有可行解并选择综合干扰代价最低的节点，以最大化集群的整体效用。若候选集中无任何节点满足条件，该请求将执行退避策略重回队列。

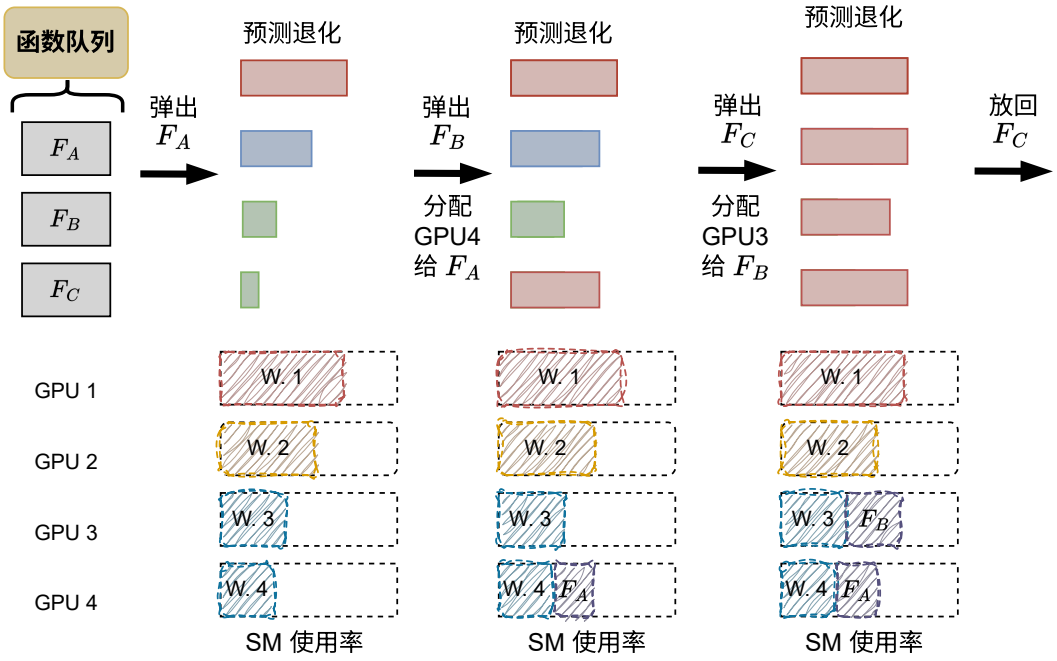

图 3.4　调度示例图
Figure 3.4　The Example of Scheduling

图 3.4 直观展示了调度过程中的资源放置决策流程。假设当前等待队列中存在三个函数实例 F_A、F_B 与 F_C。当函数到达时，系统首先将其加入等待队列，并依据公式 3.8 计算各函数的优先级。在本示例中，F_A 的优先级得分最高，因此被优先取出进行调度。基于先前的画像分析，系统提取函数模型的关键特征，并利用性能预测器

评估其与四个候选 GPU 上现有负载协同部署时可能产生的性能退化。预测结果显示，将 F_A 分配至 GPU 4 所产生的性能影响最小，且满足其服务等级目标，故最终将 F_A 调度至 GPU 4。随后，系统取出 F_B，以相同方式评估其在各 GPU 上的预期性能影响，最终选择将其分配至 GPU 3。在处理 F_C 时，预测结果表明，在任何候选 GPU 上部署该函数，均会导致训练任务性能退化超过阈值 $\theta_{\text{train}}^{\max}$，或无法满足其截止期限要求（$SLO_{\text{deadline}}$）。因此，系统触发拒绝机制，将 F_C 重新压入队列，等待下一轮调度机会。

3.4 基于 LS-LSTM 的预热器

3.4.1 冷启动延迟的构成与混合 LSTM 策略

在服务器无感知深度学习推理场景中，冷启动不仅是性能瓶颈的主要来源，更是导致长尾延迟难以预测的根本原因。与传统的轻量级 Web 函数（通常仅需毫秒级启动）显著不同，深度学习函数的运行时环境初始化极其沉重，往往涉及复杂的软硬件交互。为了系统地分析这一瓶颈，本文将一次完整的冷启动延迟 T_{cold} 形式化分解为四个串行的关键阶段：

$$T_{cold} = T_{alloc} + T_{env} + T_{gpu\_ctx} + T_{model} \tag{3.12}$$

其中，T_{alloc} 代表容器编排系统（如 Kubernetes）进行资源调度、镜像拉取及容器沙箱创建的时间；T_{env} 涉及 Python 解释器启动及大型依赖库（如 TensorFlow、PyTorch 或 MXNet）的导入与初始化开销；T_{gpu_ctx} 特指 GPU 上下文（CUDA Context）的创建时间，这一过程需要设备驱动程序在 GPU 显存中建立页表、分配统一内存寻址空间，并进行设备同步，其实际耗时通常在几百毫秒至数秒量级；而 T_{model} 则是将庞大的模型权重参数从持久化存储（磁盘或网络存储）经由 CPU 内存搬运至 GPU 显存（H2D Copy）的时间，受限于 PCIe 总线带宽和磁盘 I/O 吞吐。实测数据表明，对于参数量巨大的现代模型，$T_{gpu\_ctx} + T_{model}$ 往往占据总启动时间的 80% 以上，导致冷启动总时长可能高达 10 秒甚至更多，这远高于实际的推理执行时间 T_{exec}（通常为几十毫秒）。因此，为了严格保障在线推理服务的服务等级目标，单纯依靠传统的按需启动策略是不可行的，尽可能消除 T_{cold} 成为了系统设计的关键。

然而，消除冷启动的主要手段——即提前预热并保持容器运行——引入了巨大的成本挑战。与廉价的 CPU 实例不同，GPU 资源极其昂贵且稀缺。盲目的预热会导致高昂的算力资源在无请求期间空转，造成严重的经济浪费。为了在降低冷启动率与减少资源浪费之间寻求最优的折中方案，本文提出了一种基于双周期长短期记忆网

络 LS-LSTM 的混合预测策略。由于生产环境中的服务器无感知流量通常表现出明显的双重特性——即叠加在长周期日夜潮汐规律之上的短周期随机突发脉冲，单一尺度的预测模型（如 ARIMA 或简单的 RNN）难以同时捕捉这两种频域截然不同的特征。LSTM 网络通过引入遗忘门、输入门和输出门机制，有效解决了传统循环神经网络 RNN 在处理长序列时的梯度消失问题，能够从历史时间序列数据中学习复杂的长距离依赖关系，因此非常适合用于对这种非平稳的函数到达模式进行建模。

为了量化预热策略在成本与性能之间的权衡效果，本文定义了两个核心评估指标并构建了优化目标函数。首先是资源浪费率 R_{waste}，它从时间维度衡量了预热容器处于空闲状态的比例，反映了系统的超额配置成本。假设系统中有 M 个容器实例，令 $T_{idle}^{(i)}$ 表示第 i 个容器在生命周期内的累积空闲等待时间，$T_{total}^{(i)}$ 为其总生命周期时长，则系统整体的资源浪费率定义为：

$$R_{waste} = \frac{\sum_{i=1}^{M} T_{idle}^{(i)}}{\sum_{i=1}^{M} T_{total}^{(i)}} \tag{3.13}$$

其次是冷启动率 R_{cold}，即在总请求中未能命中预热容器而被迫经历完整冷启动流程的请求比例，这直接反映了用户感知的服务质量受损程度。令 N_{cold} 为发生冷启动的请求数，N_{total} 为处理的总请求数，则：

$$R_{cold} = \frac{N_{cold}}{N_{total}} \tag{3.14}$$

理想的预热器应当在尽可能降低 R_{waste} 的同时，将 R_{cold} 维持在极低的水平。本研究的优化目标可以形式化为一个带约束的最小化问题：即在满足服务质量约束 $R_{cold} \leq \epsilon$（其中 ϵ 为容忍阈值，例如 1%）的前提下，最小化资源浪费率 R_{waste}。通过 LS-LSTM 提供的精准预测，系统能够动态调整预热窗口的大小，从而逼近这一理论最优解。

3.4.2　预热工作流与预测融合

为了实现上述优化目标，本文设计了一套闭环的预热控制系统。如图 3.5 所示，该系统通过感知、预测与决策三个递进环节组成的流水线，实现了对 GPU 资源的精细化管理。工作流始于**数据采集与预处理阶段**。系统监控模块以守护进程 Daemon 的方式驻留在集群控制平面，实时持续跟踪每个服务器无感知函数的全生命周期事件，并将实时监控到的函数到达请求转化为固定时间窗口内的历史序列向量 $\mathbf{x}_t$。为了将离散的到达事件转化为可供神经网络处理的时序数据，本文将连续的时间轴划分为固定的时间窗口 Δt（例如 1 分钟）。设 t 时刻的实际到达请求数为 r_t，本文构建长度

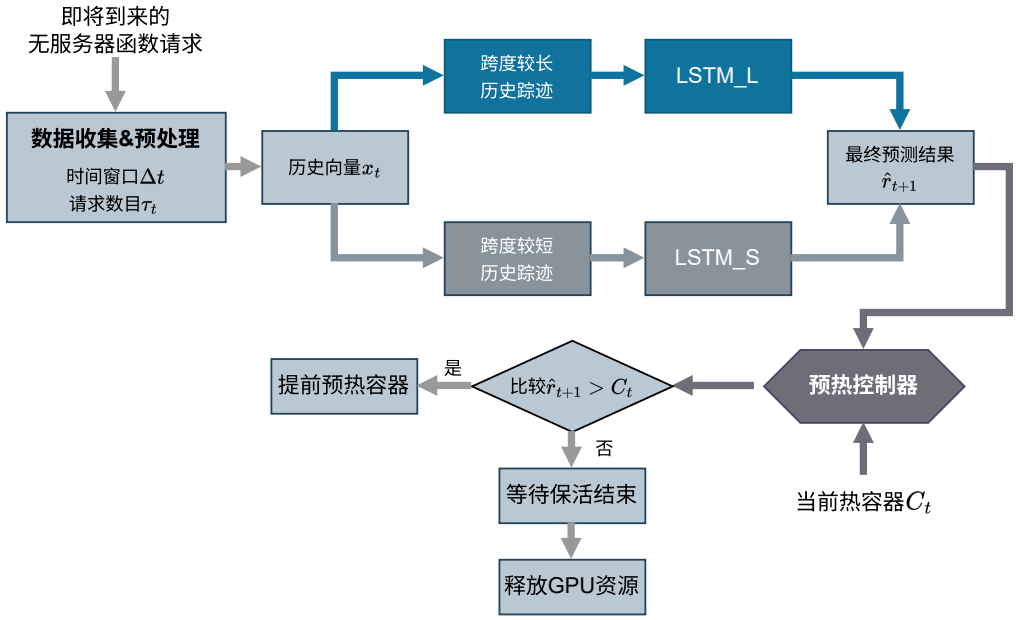

图 3.5 基于 LS-LSTM 的主动预热器工作流程图
Figure 3.5 The Workflow of the LS-LSTM Based Prewarmer

为 n 的历史滑动窗口序列 $\mathbf{x}_t$ 作为模型的输入特征向量：

$$\mathbf{x}_t = \left[\frac{r_{t-n} - \mu}{\sigma}, \frac{r_{t-n+1} - \mu}{\sigma}, \ldots, \frac{r_{t-1} - \mu}{\sigma}\right] \tag{3.15}$$

其中，为了加速 LSTM 网络的收敛并避免数值不稳定性，本文引入了 Z-Score 归一化处理，μ 和 σ 分别为历史窗口内请求数的均值与标准差。

随后，预处理后的向量被并行输入至 **LS-LSTM 预测模块**。其中包括长期模型（LSTM-L）与短期模型（LSTM-S）：LSTM-L 使用跨度较长（如过去 24 小时至 7 天）的历史踪迹 $\mathcal{D}_{long}$ 进行离线训练，旨在捕捉以天为周期的宏观日夜潮汐规律（Diurnal Patterns）；而 LSTM-S 则聚焦于微观层面的负载变化，使用最近（如过去 1 小时）的高频数据 $\mathcal{D}_{short}$ 进行训练，旨在敏锐感知当前的突发脉冲特征。为了防止模型老化，系统采用滑动窗口机制，定期使用新采集的数据对模型权重进行在线更新。

两者的输出在**加权融合层**进行汇聚。系统通过动态平衡因子 α 生成下一时刻的最终预测值 $\hat{r}_{t+1}$：

$$\hat{r}_{t+1} = \alpha \cdot \hat{r}_{L,t+1} + (1 - \alpha) \cdot \hat{r}_{S,t+1} \tag{3.16}$$

其中 $\alpha \in [0, 1]$ 是一个可动态配置的权重，用于调节系统对长期趋势与短期波动的敏感度。默认情况下 α 设为 0.5，但在检测到非周期性剧烈震荡时，系统会自动减小 α

以增加短期模型的权重。

最终，**预热控制器**依据 $\hat{r}_{t+1}$ 与当前温热容器存量 C_t 的差值执行弹性伸缩决策。当预测需求上升（$\hat{r}_{t+1} > C_t$）时，系统提前触发包含 GPU 上下文初始化与模型加载在内的主动扩容，以确保后续请求命中热容器，从而彻底消除冷启动延迟。当预测需求下降（$\hat{r}_{t+1} < C_t$）时，为了防止预测抖动导致的资源震荡，系统采用带有冷却期的缩容策略，按策略释放冗余资源，从而在保障服务质量的前提下实现 GPU 资源利用率的最大化。

3.5　本章小结

本章对提出的基于服务器无感知架构的深度学习集群资源管理系统的设计进行了详细介绍。首先，对该系统的整体架构以及工作流程进行概述。然后，分别对该架构中的三个核心模块进行设计，分别是混合部署性能退化预测模块、感知退化的调度策略以及基于 LS-LSTM 的主动预热机制。混合部署性能退化预测模块与感知退化的调度策略相结合，通过量化异构负载间的干扰风险并实施精细化的准入控制，在保障服务等级目标的前提下实现了高效的资源复用。基于 LS-LSTM 的主动预热机制利用长短期记忆网络捕捉流量的长短期特征，有效平衡了深度学习推理任务的冷启动延迟与资源空闲浪费之间的矛盾。

第 4 章 系统实现

基于第 3 章中 SMore 的系统架构设计与关键算法原理，本章详细阐述了系统的工程实现细节。为了验证所提方法的有效性，本文在 Linux 环境下构建了一个功能完整的原型系统。系统整体采用 Python 作为主要开发语言，代码量超过 3000 行。其中，深度学习相关的训练负载与预测模型基于 PyTorch 框架构建，服务器无感知函数的运行时环境通过 Flask[①]轻量级 Web 框架进行封装，而组件间的高效通信则由 gRPC 协议支撑。

本章的组织结构如下：第 4.1 节首先概括了原型系统的整体实现架构，详细描述了控制平面与数据平面的分离设计以及基于 Docker 的协同工作流程；第 4.2 节具体阐述了服务器无感知函数执行环境的构建，说明了基于 Flask 的标准化接口设计与容器化封装方式；第 4.3 节介绍了性能退化预测器的工程实现，包括离线数据采集管线、特征工程以及模型的训练与持久化过程；第 4.4 节深入说明了感知性能退化的调度器实现，涵盖了决策循环、代价评估函数以及节点状态管理等核心逻辑；第 4.5 节阐述了基于 LS-LSTM 的预热器模块的落地方式及其与调度器的交互接口；最后，第 4.6 节分析了系统内部的通信机制设计，包括核心数据结构定义、消息交互流程以及事件驱动的处理模型。

4.1 原型系统实现架构及工作流程描述

在具体实现上，如图4.1所示，本原型系统采用控制平面与数据平面分离的架构，并依托 Docker 对 GPU 资源与容器生命周期进行集中管理。控制平面主要由调度器、性能退化预测服务以及预热相关策略模块组成，负责维护全局资源视图、做出混合部署决策；数据平面则由部署在各 GPU 节点上的轻量级节点代理进程、常规深度学习训练任务容器以及服务器无感知函数任务容器构成，负责实际计算负载的承载以及监控数据的采集与上报。两平面之间通过基于 HTTP/RPC 的接口进行交互，从而在 Docker 环境下实现对异构负载的协同运行。

在控制平面中，调度器作为系统的逻辑中枢，对外统一接收服务器无感知推理函数的提交请求，对内维护全局 GPU 资源状态与任务分布信息。调度器通过周期性拉取各节点代理上报的 GPU 利用率、显存占用及任务执行时延等监控数据，构建实时

① https://github.com/pallets/flask/

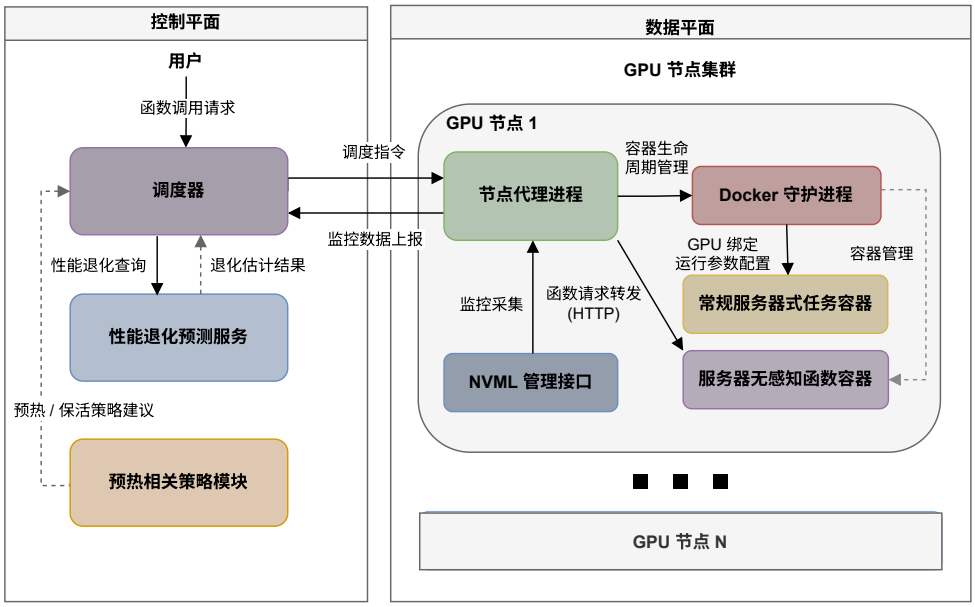

图 4.1　SMore 的原型系统实现概览图
Figure 4.1　Overview of the Prototype System Implementation of SMore

集群状态视图；当新的函数请求到达时，调度器调用性能退化预测服务，结合当前候选 GPU 节点上正在运行的训练任务与函数实例信息，查询不同混合部署组合的预期性能退化水平，并在此基础上执行准入控制和节点选择。性能退化预测服务以独立的在线服务形式部署，对外提供统一的查询接口，其内部加载离线阶段训练得到的性能退化模型以及由模型推演生成的性能退化查询表，通过键值检索与轻量推理，为调度器返回给定任务组合和资源状态下的性能退化估计。预热相关策略模块基于历史请求轨迹和流量预测模型，对不同函数的预热、保活与回收策略进行建模和分析，为系统在面对冷启动问题时提供策略依据与优化空间。

在数据平面侧，系统在每台 GPU 服务器上部署一个常驻的节点代理进程，并通过 Python 的 Docker API 接口对本地 Docker 守护进程进行编程式控制。节点代理一方面充当轻量级监控器，定期调用 NVIDIA 的 NVML 管理接口采集本节点 GPU 利用率、显存占用以及其他关键性能指标，并将聚合后的状态数据上报至控制平面；另一方面，节点代理也承担容器生命周期管理的执行角色，在本地通过 Docker API 创建、启动、停止或销毁深度学习任务容器与函数任务容器。常规深度学习任务以长期驻留的训练任务容器形式存在，其创建由现有集群作业管理系统触发，节点代理在拉起该类容器时，根据既定的 GPU 绑定信息配置 Docker 运行参数和相关环境变量，

实现训练任务对特定 GPU 设备的使用。服务器无感知函数则以 Flask 函数容器形式运行，容器内部运行无状态的 Flask 应用进程，并在启动阶段完成目标模型的加载和 CUDA 上下文初始化，通过统一的 HTTP 端点对外提供推理服务。

在上述控制平面和数据平面协同工作下，服务器无感知函数请求在现有深度学习训练任务背景负载下的典型执行时序可以概括为：首先，用户或上层系统通过统一接口将函数调用请求提交至调度器。本系统假定深度学习训练任务由既有集群作业管理流程独立提交和调度，训练容器已在 GPU 节点上长期驻留并作为高优先级背景负载存在，调度器并不干预其创建与放置过程，而是仅将这些既有训练任务视为混合部署时的资源占用与性能约束条件。

当新的服务器无感知函数请求到达时，调度器基于当前由各节点代理上报的监控信息构建全局状态视图，并调用性能退化预测服务，结合候选 GPU 节点上已经运行的训练任务与函数实例情况，评估在不同共置组合下的预期性能退化水平。在综合考虑当前 GPU 负载、预测的性能退化以及函数的 SLO 约束之后，调度器选择一个满足性能要求且能够提升整体资源利用率的目标节点，并向该节点的代理进程下发调度指令。

目标节点上的代理进程在收到指令后，将该请求转发至本节点上对应的 Flask 函数容器的 HTTP 端点执行，实现与背景训练任务的在线共置。函数请求完成执行后，节点代理会记录本次调用的实际响应时延、执行期间的 GPU 使用轨迹以及潜在的性能退化观测值，并将这些数据回传至控制平面，用于后续性能退化模型的校准以及调度与预热策略的进一步优化。通过这种以训练任务为背景负载、由调度器专注管理服务器无感知函数请求的轻量级原型实现架构，系统在无需改动现有训练作业调度机制的前提下，实现了对服务器无感知函数的统一管理和在深度学习训练任务之上的高效混合部署。

4.2 服务器无感知函数执行环境实现

原型系统在 Docker 与 Flask 之上构建了一套轻量级的服务器无感知函数执行环境。每个函数运行时对应一个独立的 Docker 容器，容器内运行统一的 Flask 应用程序，通过命令行参数绑定到指定 GPU 与监听端口，从而形成一个容器等于一个 GPU 绑定函数运行时的抽象。上层控制逻辑使用 Python 的 Docker API 拉起和销毁这些容器，并维护其 `<ip, port>` 等信息，实现对函数实例的集中管理。

容器内部的 Flask 应用对外提供一组语义明确、形式统一的 HTTP 接口，用于支

表 4.1　服务器无感知函数运行时主要接口
Table 4.1　Main APIs of the Serverless Function Runtime

接口路径	含义与说明
/	运行时健康检查接口，使用 HTTP GET 方法访问。返回固定字符串或简单 JSON，用于快速判断容器内 Flask 服务是否已成功启动并处于就绪状态，供上层调度模块和实验脚本进行连通性检测。
/load_model	模型加载接口，通过 HTTP POST 调用。请求中携带模型标识（如 model）、用户标识（uid）以及可选的配置参数（批处理大小等），运行时在绑定的 GPU 上按需构建对应的 PyTorch 模型、加载权重并完成 CUDA 上下文初始化，将其实例缓存到本地模型字典中，为后续推理请求提供热启动环境。
/delete_model	模型卸载接口，用于显式释放某个模型实例占用的显存资源。请求中指定待卸载的 model 与 uid 后，运行时会从本地缓存中移除对应实例，并在安全条件下触发显存回收，为新的模型加载或其他租户的请求腾出空间。
/predict	通用推理接口，是运行时对外提供的核心函数调用入口。采用 HTTP POST 方法，请求中携带 uid、model、批处理大小（bs）以及必要的输入数据。运行时根据 uid 与 model 在本地模型缓存中查找对应实例，若尚未加载则可根据策略自动调用内部加载逻辑；随后在绑定 GPU 上执行一次前向推理，记录端到端处理时延等性能指标，并将推理结果与观测数据以 JSON 形式返回。
/status	运行状态查询接口，以 HTTP GET 方式访问。返回当前容器内已加载模型列表、每个实例的最近一次访问时间、粗粒度显存占用情况以及简单的队列长度等信息。

撑模型的完整生命周期管理与在线推理服务的承载。该接口集覆盖了健康状态监测、模型动态加载与卸载、实时推理请求处理，以及运行时状态查询等核心功能。这些接口的设计保持语义相对稳定、调用方式一致，便于上层调度系统或实验脚本进行集成与自动化调用。表 4.1 列出了当前运行时中定义的主要接口及其对应功能说明。

在上述接口之上，容器内部的 Flask 应用程序在启动时首先解析命令行参数，设置 CUDA_VISIBLE_DEVICES 完成 GPU 绑定，并通过 gevent.pywsgi.WSGIServer 在指定端口启动异步 WSGI 服务。应用内部维护一个按 GPU 维度划分的模型缓存字典，将不同用户与不同模型类型组合映射为独立的模型实例键值；/load_model 和 /delete_model 负责显式控制该字典中模型实例的创建与销毁，而 /predict 在处理请求时则基于该缓存完成模型选择与推理执行。

当前运行时支持包括卷积神经网络（如 VGG-16、MobileNet、ResNet-50）、视觉

Transformer（DeepViT）、推荐模型（DeepFM）、语义分割模型（SegNet）以及自然语言处理 BERT/RoBERTa 等典型深度学习工作负载，能够在单块 GPU 上实现多模型、多用户的高密度共置。每次 `/predict` 调用在完成一次前向推理后，都会记录从请求接收到响应发送之间的端到端时延，并将该指标连同简单的状态信息一并返回。调度模块与实验脚本通过批量调用这些接口，在不同共置组合、批处理大小与负载强度下收集响应时间与资源使用数据，为后续章节中的性能退化建模与调度策略评估提供了细粒度、可重现的运行基础。

通过上述基于 Docker 的容器抽象以及一组功能清晰、语义稳定的标准化 HTTP 接口，原型系统在不依赖完整 FaaS 平台和大规模编排系统的前提下，既满足了多模型、多租户共置场景下对函数执行环境的核心需求，又实现了结构简洁、功能完备且易于扩展的服务器无感知函数运行时支撑。

4.3 性能退化预测器的实现

本节从工程实现角度说明混合部署性能退化预测器在原型系统中的落地方式，重点介绍数据管线与模型训练过程。

4.3.1 离线数据管线实现

为训练退化预测模型，原型系统实现了一条完全离线的数据采集与预处理管线，用于自动生成双路共置场景下的画像数据。该管线由一组 Python 脚本驱动，在 Docker 容器中按预定义配置批量运行不同的训练任务与推理函数组合，核心流程包括：

- **基线运行：**对每个训练任务或推理函数，单独占用 GPU 运行一段时间，记录稳定阶段的吞吐量或延迟分布，作为后续计算退化程度的独占基线。
- **共置实验：**选取若干 (T_i, s_j) 组合，在同一张 GPU 上同时启动两个容器：训练容器循环执行若干迭代，推理容器通过 Flask 应用暴露的 `/predict` 接口，由驱动脚本以指定 QPS 模式持续注入请求。
- **指标采集：**训练脚本与推理服务在容器内部定期写出每轮迭代耗时、每个时间窗口的 QPS、平均/尾延迟等日志；节点侧监控进程基于 NVML 轮询 GPU SM 利用率、显存使用率等全局指标，并为每轮实验打上模型 ID、批处理大小、目标 QPS 等元数据。
- **数据整理：**实验结束后，离线处理脚本使用 Pandas 对多源日志按时间戳与实验 ID 进行对齐，过滤预热与不稳定阶段，合并为统一格式的记录行，最终输出为

若干 csv 文件，供后续特征构造和模型训练使用。

该数据管线以配置文件驱动，支持通过简单修改实验列表增删模型与组合，保证了预测器训练数据的可复现性和可扩展性。

4.3.2　特征与标签构造

在实现层面，原型系统依据第 3.2节中定义的特征与标签，将原始日志处理为可供回归模型直接训练的特征矩阵与标签向量。

静态特征通过离线脚本一次性计算生成，支持多次复用，具体包括：

- 基于 PyTorch 及 ptflops 工具计算的理论 FLOPs 与模型参数量（经对数变换与归一化处理）；
- 通过遍历 nn.Module 统计得到的各类层数量，如卷积层、全连接层、归一化层、池化层等；
- 在给定批次大小下，执行单次前向传播（或前向 + 反向传播）所测得的峰值显存占用及激活规模等指标。

所有静态特征被整合为静态特征表，并以模型 ID 作为键值索引，便于在训练数据构建和系统运行时高效查询。

动态特征与标签通过共置实验的运行日志生成。针对每次共置实验，在对日志进行时间对齐与滑动窗口划分后，从训练/推理日志与 GPU 监控数据中提取模型组合、批处理大小、SM 利用率、显存使用率、QPS、迭代耗时、请求延迟等指标，并在每个窗口内进行均值或分位数聚合，形成与静态特征拼接后的完整特征向量。根据任务/函数在独占运行和共置运行下的性能对比，计算吞吐量退化率和延迟膨胀率等标签值，并将训练任务和推理函数的性能退化分别记录至相应标签文件。

最终，数据处理管线生成若干特征文件与标签文件供模型训练使用，其中字段命名与设计章节中的符号一一对应，便于对照与分析。

4.3.3　模型训练与持久化

性能退化预测器采用统一的离线训练和模型文件持久化结合的实现方式，由独立训练脚本完成，其主要步骤为：

1. 从特征与标签文件中加载数据，并按配置划分训练集和验证集；
2. 使用 scikit-learn 提供的回归模型实现，通过配置文件设定模型相关的超参数；
3. 在验证集上以 RMSLE 为主评估指标，同时记录 MAE 等辅助指标，在不同超参

数组合间进行比较；

4. 选择验证效果最优的模型，将其通过 `joblib` 或 `pickle` 序列化为二进制文件，分别保存训练退化预测器和推理退化预测器，文件名中包含特征版本与时间戳，便于后续版本管理与回溯。

模型训练过程与在线系统实现相互解耦：当数据分布或算法配置发生变化时，仅需重新执行训练脚本并替换对应的模型文件，即可完成预测器的升级，无需引入额外的在线学习组件或常驻服务进程。

通过将离线特征处理管线与批量训练流程相结合，并辅以模型文件的持久化管理，性能退化预测器在工程上形成了一套相对独立而又易于演进的实现路径，为后续章节中的通信机制和调度策略提供了稳定、统一的预测能力输出接口。

4.4 感知性能退化的调度器实现

在实现层面，感知性能退化的调度器的总体架构由三部分组成：单线程决策循环、事件队列、节点状态缓存，其总体流程如图4.2所示。原型系统在控制平面内启动一个专门的调度线程作为全局决策器，该线程从进程内的调度事件队列中持续取出待处理请求事件；每条事件对应一次服务器无感知函数调用请求或其超时/完成通知。围绕这一调度线程，系统在内存中维护一个抽象的节点状态表，用于记录每块 GPU 上当前驻留的训练任务信息、已共置的推理请求数量及其模型类型、近期观测到的尾延迟统计等。调度线程在每次处理事件时，会基于当前节点状态表和性能退化预测结果选择目标 GPU，并通过 RPC 接口将请求下发到对应的函数运行容器中执行。

在数据结构设计方面，调度器为每块物理 GPU 构建了一个 NodeEntry 结构，用于统一描述该 GPU 的静态属性与动态状态。静态属性包括 GPU 的唯一标识、型号和显存容量等；动态状态则主要包括：当前是否运行训练任务及其标记、已共置推理请求的数量、已共置模型的种类统计，以及最近一个时间窗口内的平均响应时间和 P99 尾延迟等观测指标。调度器同时维护一个全局的 RequestRecord 表，为每个正在排队或执行的服务器无感知函数请求记录其到达时间、模型标识、目标 SLO 以及当前所处的生命周期阶段（排队、运行、完成或超时）。通过这两类数据结构，调度器可以在一次决策循环中快速获取候选 GPU 的负载状态以及请求本身的时延约束，为后续的代价评估提供输入。

为了支持感知退化的放置决策，调度线程在内部集成了一个轻量级的代价评估函数。该函数接收当前请求和候选节点这两个参数作为输入，结合离线训练得到的性

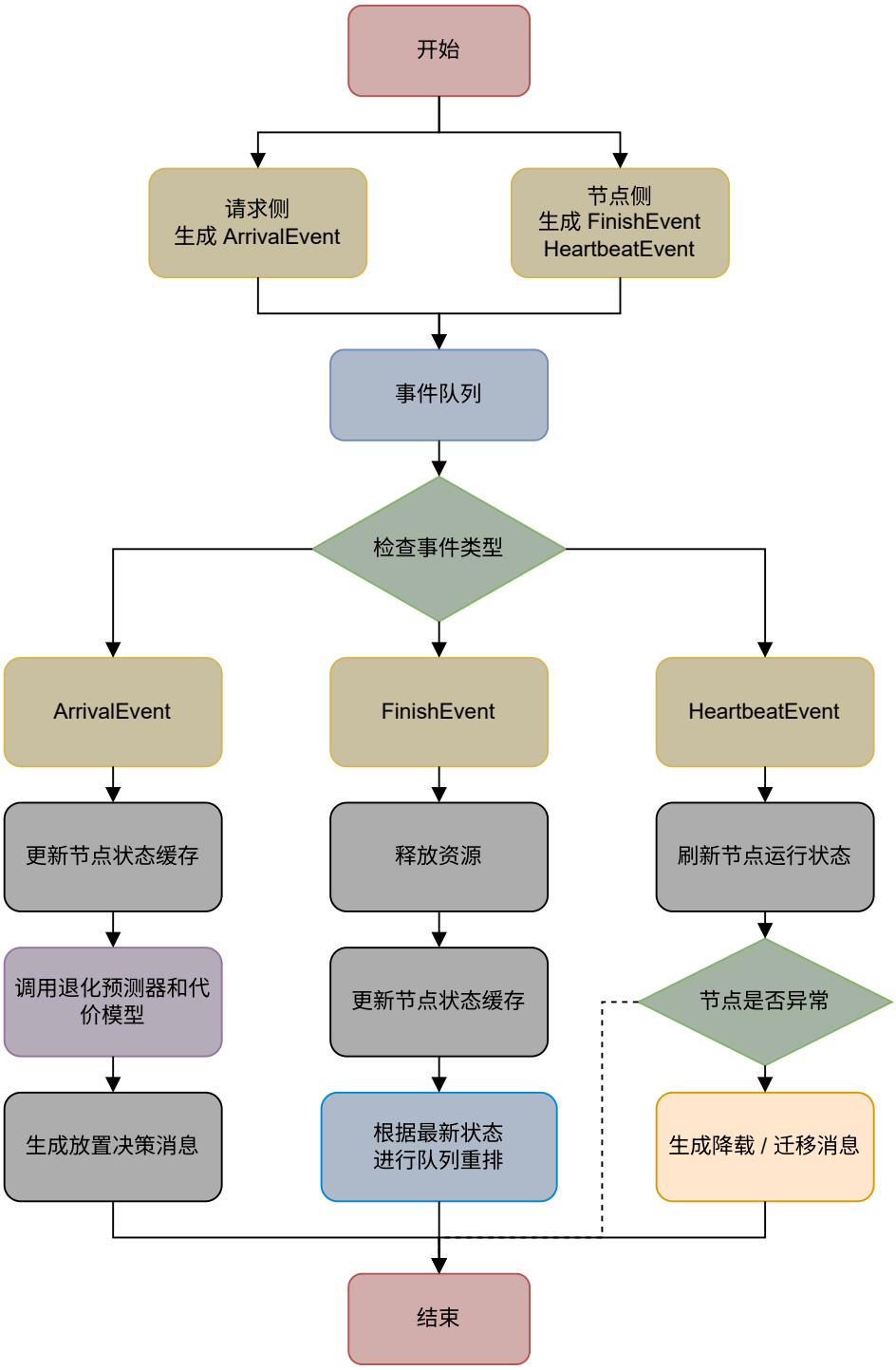

图 4.2　感知性能退化的调度器流程图
Figure 4.2　Degradation-Aware Scheduler Flowchart

能退化预测器以及预加载的干扰查找表，对不同放置方案的代价进行估算。具体而言，调度器首先根据请求的模型类型、批大小等静态特征，以及候选节点上当前共置的模型集合，从退化表中查询或通过预测器推断其在该节点上的预期延迟放大系数；随后，以此估计训练任务吞吐量的潜在下降比例和当前请求尾延迟相对其 SLO 的安全裕度。调度器将这两部分损失归一化后加权求和，形成单一的代价值；在实现上，代价评估函数被设计为纯计算函数，不依赖外部 I/O，从而保证单次评估的开销在亚毫秒级。

在事件处理逻辑上，调度器采用一个长生命周期的决策循环，对全局调度事件队列进行循环消费。事件类型主要包括三类：请求到达事件（ArrivalEvent）、请求完成事件（FinishEvent）和请求超时事件（TimeoutEvent）。对于到达事件，调度线程会将请求登记到 RequestRecord 表中，并立即触发一次放置决策：遍历当前所有可用 GPU 节点，跳过处于离线或资源饱和状态的节点，对其余节点逐一调用代价评估函数，最终选择代价最小且预计能满足 SLO 的节点作为目标；若所有候选节点均无法满足时延约束，则调度器直接拒绝该请求并更新统计信息。对于完成事件，调度线程更新对应 GPU 的 NodeEntry 状态，将其共置推理数量减一，同时从请求表中移除该条记录。对于超时事件，调度器在逻辑上将对应请求标记为失败，并触发一次本地统计更新，用于后续调整代价评估中的权重参数。

在函数调用调度层面，调度器采用全局排队与节点本地串行队列结合的混合结构。全局事件队列负责吸纳所有新到达的服务器无感知函数请求，保证调度线程可以按到达顺序进行统一决策；当某请求被分配到具体 GPU 后，调度器会通过 RPC 将其转发至该 GPU 上运行的 Flask 函数容器，并将请求加入该容器对应的本地队列。在实现中，每个 Flask 容器内部维护一个按模型划分的请求队列，确保同一模型实例的请求在容器内串行执行，避免由于模型权重复用和中间状态共享导致的并发冲突。调度器仅在全局层面负责请求与 GPU 的匹配，不参与容器内的细粒度调度，从而将全局调度逻辑与本地执行逻辑清晰分离。

在放置策略实现上，调度器将感知退化的启发式决策显式编码为一个多维代价比较过程。具体过程为：对于每个候选 GPU，调度器首先从 NodeEntry 中读取其当前训练/推理负载情况，然后调用性能退化预测接口，获得将当前请求放置在该 GPU 上时，训练吞吐量的预期退化比例以及请求自身尾延迟的预期增加量；随后，调度器基于配置文件中预设的权重参数，将训练吞吐量损失和请求时延风险合成为一个标量代价，并在此基础上增加一个与当前共置请求数量相关的惩罚项，用于反映高密度共

置场景下的额外不确定性。不同 GPU 的代价值计算完成后，调度器从中选取综合代价最小的 GPU 作为放置目标，并将这一结果连同预测的退化信息一并记录到调度日志中，便于后续分析。

为了保证调度器内部状态与实际执行环境之间的一致性，系统构建了一个基于异步回调的状态同步机制。具体实现上，执行节点在每次完成函数调用后，会向控制平面发送一条简要的结果上报消息，其中包含请求标识、实际执行时延以及执行所在的 GPU 标识。调度器在收到该消息后，更新对应请求的状态，并以滑动窗口方式维护各 GPU 的历史时延统计信息。当发现某个节点的实际退化情况长期偏离离线退化表或预测模型的估计时，调度器会通过调整代价函数中的惩罚权重或直接标记该节点为 “高风险”，在后续放置决策中降低其优先级。通过这种反馈闭环，调度器既能保持决策过程的轻量性，又能在长时间运行过程中逐步修正对各节点干扰特性的认知。

此外，针对调度器自身的性能评估需求，系统还实现了一个简化的离线回放版本。在该版本中，请求到达序列和节点状态演化由预先记录的日志文件驱动，真实的 RPC 调用和容器执行被抽象为函数调用和时间累加操作，而核心的放置逻辑、代价评估和状态更新流程保持不变。通过这一实现，系统可以在单机环境下高效重放大规模负载轨迹，对比不同方案的决策效果和时间开销，为后续实验章节中的调度策略分析提供工程层面的支撑。

4.5　基于 LS-LSTM 的预热器实现

本节从实现角度介绍基于 LS-LSTM 的预热器模块在原型系统中的落地方式，重点说明其在系统中的位置、与各组件的交互接口以及核心控制逻辑，其整体架构如图4.3所示。

在系统架构上，预热器作为控制平面中的独立功能模块，与感知性能退化的调度器共置于同一节点，对外仅通过少量 HTTP 接口提供负载预测与预热建议服务。预热器不直接管理 GPU 容器的创建与销毁，而是向调度器返回结构化的预热与保活参数，由调度器结合当前资源占用与性能退化预测结果，统一做出最终决策并驱动底层 Docker 实例的拉起与回收。

在与函数执行环境的交互方面，预热器并不嵌入到具体的 Flask 应用或模型容器内部，而是通过读取统一格式的调用日志间接获取负载信息。服务器无感知函数执行环境在每次请求完成后，将函数标识、请求时间戳和处理耗时等关键信息追加写入本地日志文件；节点侧的简单收集脚本周期性地将这些日志汇总到控制节点。预热器进

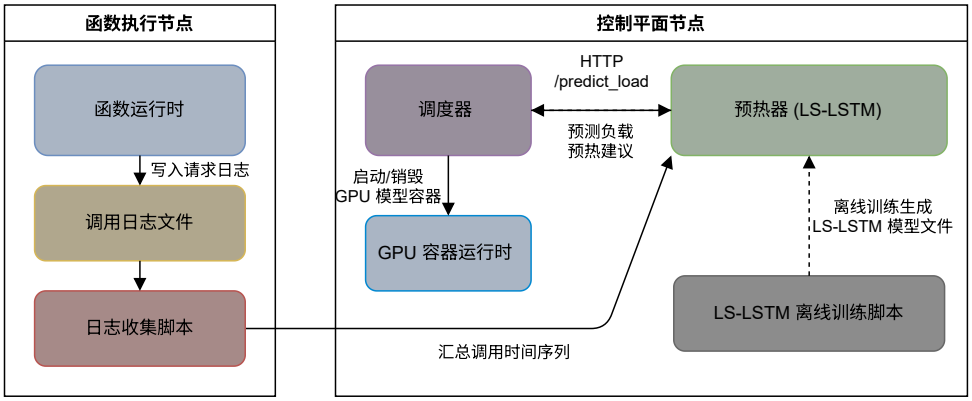

图 4.3 预热器架构图
Figure 4.3 Prewarmer Architecture

程中的数据处理线程负责解析这些增量日志，将其按函数划分并聚合为时间序列形式，以便在需要进行预测时为 LS-LSTM 模型构造输入。

为了避免与性能退化预测器的实现逻辑重复，预热器在训练与模型管理上采用了尽量简化的工程路径。离线训练阶段由独立脚本完成：脚本从预处理好的时间序列文件中加载数据，基于 PyTorch 实现的 LS-LSTM 结构进行批量训练，并在训练结束后将模型参数以及必要的归一化配置保存为单一模型文件。在线运行时，预热器进程在启动阶段一次性加载该模型，并在后续运行过程中保持常驻内存，不再进行动态更新或在线学习。与性能退化预测器相比，预热器的训练流程更加轻量，其目标是为整个系统提供一个稳定、可复用的针对请求到达模式的刻画能力。

在在线服务形态上，预热器通过一个简洁的 HTTP 接口向调度器暴露预测能力。当前原型系统实现了形如 `/predict_load` 的查询端点：调度器在评估某个函数的部署状态时，将函数标识以及最近若干个时间片的请求计数打包为请求发送给预热器；预热器在内部完成数据归一化、张量构造和模型前向计算后，返回未来一小段时间窗内的负载估计值。为了降低接口复杂度，返回结果被组织为固定结构的 JSON 对象，其中包括每个时间片的预测请求数以及简单的置信指标，调度器可以在不关心具体模型细节的前提下直接使用这些信息。

在预热控制逻辑上，预热器并不试图在内部实现完整的资源管理策略，而是将自身定位为预测与建议角色。当前实现中，预热器在得到负载预测序列后，依据一组预设阈值与经验规则，将预测值转换为预热相关的若干建议参数。上述参数以统一字段的形式返回给调度器，调度器再结合性能退化预测器给出的干扰估计和当前 GPU 利用率，对是否执行模型加载、维持或卸载等操作做出综合判断。通过这种职责划分，

预热器与调度器之间形成了清晰的接口边界，便于后续替换更复杂的预测模型或策略算法而不影响其余模块。

总体来看，基于 LS-LSTM 的预热器实现遵循模块职责单一和接口清晰的原则：一方面复用与性能退化预测器相似的离线训练与模型持久化基础设施，降低整体工程复杂度；另一方面在在线阶段仅承担请求负载预测与预热参数建议的职责，通过轻量的 HTTP 接口与调度器协同工作，从而在不引入额外系统耦合的前提下，为原型系统提供可控、可演进的冷启动优化能力。

4.6　各模块通信机制设计与实现

该原型系统的通信机制在实现层面采用如下总体结构：通过集中式控制端进行统一调度，配合部署在各节点的轻量级客户端代理，并以规范化的统一消息格式完成数据交换。控制平面中的全局调度器作为通信核心，统一负责与各计算节点上的函数执行环境、训练任务监控组件以及性能退化预测模块之间的消息交互；每台计算节点上部署一个本地通信代理进程，用于与调度器维持长连接并转发本节点内部各模块的状态与控制指令。所有跨进程消息均采用统一的数据结构进行编码，保证了不同组件间的数据语义一致性与接口可扩展性。

在数据结构设计方面，通信子系统围绕节点、请求和预测三类核心实体定义了对应的消息结构。针对节点状态，系统设计了 `NodeStatus` 结构，用于描述单块 GPU 的静态属性与运行时状态，包括 GPU 标识、型号、总显存、当前显存占用、训练任务标记以及已共置服务器无感知函数请求数量等字段。针对服务器无感知函数请求，系统使用 `InvokeRequest` 与 `InvokeResult` 结构分别表示一次服务器无感知函数调用的到达事件和完成事件，记录请求标识、模型类型、输入规模、目标 SLO、到达时间戳以及实际执行时延等信息。对于性能退化预测，系统定义了 `PredictQuery` 与 `PredictResponse` 结构来描述调度器与预测器之间的查询—响应交互，前者携带候选 GPU 当前共置模型集合、训练任务类型等特征，后者返回训练吞吐量损失比例与推理时延放大系数等估计值。通过这些结构化定义，通信层可以在不关心上层实现细节的前提下，完成不同模块间语义清晰的数据传递。

在通信拓扑上，系统整体采用以全局调度器为中心的星型结构。调度器进程内部集成一个网络服务模块，监听固定端口并维护与各计算节点代理、性能预测模块之间的长连接；计算节点侧的通信代理则作为轻量客户端，负责周期性上报本节点的 `NodeStatus`，并接收调度器下发的函数调用指令与控制命令。在一次典型的调用路

径中，用户请求首先通过本地 HTTP 接口进入某个节点的 Flask 函数容器，由容器内的代理线程将请求摘要转交给本节点通信代理；后者将其封装为 `InvokeRequest` 消息发送至全局调度器。调度器完成放置决策后，通过同一连接将目标 GPU 及相关参数回传给对应节点代理，由代理转发给目标 GPU 上的函数容器执行。函数执行完成后，容器将结果与时延信息封装为 `InvokeResult` 反馈给节点代理，再由代理上报至调度器，用于更新全局状态视图。

在事件处理逻辑上，通信模块同样采用长生命周期处理线程与事件队列结合的结构，以解耦网络 I/O 与核心逻辑。在调度器侧，网络层为每条入站连接配置一个接收线程，该线程仅负责完成消息的解码与基本合法性检查，并将解析后的消息对象压入进程内的通信事件队列；而真正的调度决策、状态更新和日志记录，则由独立的调度线程与监控线程从队列中异步消费这些事件完成。例如，当接收到来自节点代理的 `InvokeRequest` 消息时，网络线程仅负责将其转换为内部的请求到达事件，而不会在 I/O 线程中直接调用性能退化预测器或更新节点状态表。相应地，在计算节点侧，通信代理维护一个本地事件队列，用于协调与 Flask 应用容器、训练任务监控脚本之间的消息传递，保证节点内部不同模块之间的交互不会阻塞与调度器的外部连接。

在请求与响应的具体处理流程上，通信模块对不同类型的消息采用差异化的路径优化。对于高频的状态心跳与 GPU 利用率上报，节点通信代理以固定周期采样本地监控脚本暴露的指标，将其聚合为紧凑的 `NodeStatus` 消息按批次发送，以减少网络报文数量和序列化开销；调度器侧则采用滑动窗口的方式更新节点状态缓存，仅保留最近若干次上报结果参与后续决策。对于延迟敏感的函数调用指令，调度器在生成放置决策后立即构造对应的响应消息，将目标 GPU 标识、模型类型以及必要的会话参数封装后，通过现有连接发送至请求来源节点，节点代理在收到后第一时间写入容器本地队列，减少不必要的中间缓冲。对于与性能退化预测器的交互，由于其频率低于函数调用但对决策质量影响较大，调度器在构造 `PredictQuery` 时会附加当前时间戳，并在等待超时后自动回退到本地退化查找表，以避免预测服务异常造成整体决策阻塞。

4.7 本章小结

本章对 SMore 的原型系统实现细节进行了全面介绍。首先，阐述了系统采用的控制平面与数据平面分离的总体架构，并详细说明了基于 Docker 和 Flask 构建的标准化服务器无感知函数执行环境及其关键接口设计。其次，深入解析了系统的核心智

能决策组件实现，涵盖了性能退化预测器的离线数据管线与模型持久化流程、感知性能退化的调度器决策逻辑与代价评估方法，以及基于 LS-LSTM 的预热器模块的工程落地。最后，介绍了支撑各组件协同工作的通信机制，包括以调度器为中心的星型拓扑结构、统一的数据消息定义以及事件驱动的处理模型。基于上述实现方案，下一章将在实际的测试平台上开展全面的实验，以评估系统在真实场景下的性能表现。

第 5 章 实验与分析

本章基于构建的 SMore 原型在包含 NVIDIA RTX 3090 的物理集群测试平台上开展实验，旨在全面验证系统在异构负载混合部署场景下的有效性与鲁棒性。首先，第5.1 节详细阐述了实验的评估方法，包括硬件测试平台的具体配置、常规服务器式与服务器无感知负载的选取标准以及实验跟踪日志的生成方式。接着，为了验证系统核心组件的可靠性，第5.2 节与第5.3 节分别对性能退化预测模型的精度及 LS-LSTM 冷启动预热策略的有效性进行了独立评估。在此基础上，第5.4 节在单应用与多应用并发场景下，对 SMore 的整体性能进行了真实测试平台验证，重点量化分析了混合部署对训练任务的性能退化影响、服务器无感知请求的准入率以及集群 GPU 资源利用率的提升效果。随后，第5.5 节将 SMore 与 Random、EDF-util 及 ElasticFlow 等现有调度基线进行了对比实验，论证了本系统在保障服务质量（SLO）与提升资源效率方面的优势。最后，第 5.6 节通过大规模模拟实验与支持 MIG（多实例 GPU）特性的 A100 平台实验，进一步验证了系统的调度低开销特性与硬件泛化能力。

5.1 实验配置

本节首先对 SMore 原型系统的硬件运行环境及实验测试平台配置进行说明。随后，详细介绍实验所采用的深度学习工作负载（包括常规服务器式训练任务与服务器无感知推理任务）及其特性参数。最后，阐述服务器无感知流量跟踪日志的生成方法及实验中用于对比的基准调度策略。

5.1.1 实验环境与测试平台

为了评估 SMore 在异构负载混合部署场景下的性能表现，本文在一个配备了 8 张 Nvidia RTX 3090 GPU 的物理节点上构建了实验测试平台。该节点的详细硬件规格如表 5.1 所示。在软件环境方面，操作系统选用 Ubuntu 20.04 LTS。为了聚焦于量化分析负载间的干扰影响，所有的常规服务器式训练负载与服务器无感知函数调用均托管于同一物理节点内，通过 SMore 调度器执行准入控制与资源分配。

表 5.1　实验测试平台硬件配置
Table 5.1　Experimental Testbed Hardware Configuration

组件 (Component)	规格 (Specification)
CPU 处理器	Intel Xeon Gold 6248R (2× Socket)
CPU 核心数	96 物理核心 (192 线程)
CPU 频率	3.00 GHz (Base)
LLC 缓存	71.5 MB (Shared)
内存容量	512 GB DDR4
存储设备	11 TB NVMe SSD
GPU 加速卡	8× Nvidia RTX 3090
GPU 显存	24 GB GDDR6X (per GPU)
CUDA 核心数	10496 (per GPU)
操作系统	Ubuntu 20.04 LTS

5.1.2　工作负载与流量生成

在负载选择方面，本文选取了计算机视觉、自然语言处理及推荐系统及多任务学习领域中具有代表性的深度学习模型，涵盖了不同的计算密集度与显存占用特征。实验负载被划分为两类：

1. **常规服务器式负载：**代表长时运行的深度学习训练任务。本文选取了不同架构的模型，以覆盖从低到高的广泛 GPU 利用率范围，从而全面评估系统在不同资源压力下的鲁棒性。
2. **服务器无感知负载：**代表短时、突发性的推理任务。这些任务的模型批处理大小被限制在 32 以内，以模拟典型的在线推理场景。

为了保证实验结果在真实生产环境中的普适性与代表性，本文构建了一个包含八种典型深度学习工作负载的基准测试集，详细配置如表 2.1 所示。该模型库跨越了计算机视觉（CV）、自然语言处理（NLP）、推荐系统（Ad.）以及多任务学习（MTL）四大核心应用领域，旨在全面覆盖从轻量级边缘模型到大规模预训练模型的不同复杂度层级。具体而言，在 CV 领域，不仅选取了 VGG-16 和 ResNet-50 等经典的卷积神经网络架构，还包含了面向移动端的轻量化模型 MobileNet 以及基于 Transformer 架构的 DeepViT；在 NLP 领域，则重点关注 BERT 和 RoBERTa 等参数量巨大的主流语言模型；此外，还引入了工业界广泛使用的推荐模型 DeepFM 以及多任务分割网络 SegNet (MAN)。

为了模拟真实的生产环境流量，本文使用了 Azure Functions Public Dataset[49] 提供的公开跟踪日志。该数据集包含了 14 天内不同函数的调用统计信息。针对 Azure

数据集中调用分布较为稀疏、难以直接复现高并发压测场景的问题，本文采用了一种基于分布缩放的流量生成方法：首先，统计原始跟踪日志中一分钟间隔内的函数到达分布；随后，将该时间窗口压缩至秒级，并根据预设的负载因子进行线性缩放，从而生成具有目标 RPS 的测试序列。对于服务器无感知任务的延迟服务等级目标，实验将其随机设置为各函数 P99 延迟的 1 至 4 倍。

5.2 性能退化预测模型验证

本节对第 3.3 节中构建的混合部署性能退化预测模型的预测精度进行详细分析。该预测模型是 SMore 调度器执行准入控制和资源分配决策的核心依据，其预测准确性直接决定了系统能否在保障常规服务器式负载服务质量的同时，最大化接纳服务器无感知请求。为了系统地评估性能退化预测器在复杂异构混部环境下的准确性与鲁棒性，本节首先阐述实验数据集的构建过程，进而分析不同训练集采样规模对模型泛化能力的影响，最后对比不同回归算法的性能差异，并深入剖析其背后的资源竞争机理。

5.2.1 数据集构建与实验设置

为了覆盖尽可能广泛的干扰场景，本文构建了一个包含 1024 个高保真样本的专用基准数据集。基于上述多样化的模型库，系统在数据采集阶段通过全排列方式遍历了不同类型的共置组合，从而构建了覆盖广泛资源特征的高保真数据集。这种组合策略确保了测试场景能够囊括典型的资源竞争模式，既包含以 ResNet、VGG 为代表的计算密集型负载，也包含以 DeepFM、BERT 为代表的访存密集型负载。对于每一组常规服务器式训练任务与服务器无感知推理函数的共置对，实验通过网格搜索方式遍历了不同的批处理大小配置，并记录了各自在独立运行与混合运行时的吞吐量及响应时延。最终的数据集样本由三部分组成：训练任务的资源特征向量、推理函数的动态特征向量以及作为标签的归一化性能退化率。该数据集旨在全面映射异构负载在 GPU 共享环境下的状态空间，为训练高精度的预测模型提供坚实的数据基础。

5.2.1.1 数据采样效率与开销权衡分析

在实际生产环境中，通过离线剖析（Profiling）获取全量共置组合的性能数据往往是不切实际的，其时间与算力成本极高。因此，探究以最小的数据采集成本实现最优预测精度具有重要的工程意义。本文引入采样比例因子 α ($\alpha \in [0,1]$)，表示从总样本空间中随机抽取 α 比例的数据作为训练集，剩余数据作为测试集，以此评估模型

表 5.2　不同回归模型在混合部署退化预测任务上的性能对比

Table 5.2　Performance Comparison of Different Regression Models for Co-Location Degradation Prediction

指标 (Metric)	随机森林 (Random Forest)	Bagging	线性回归 (Linear)
RMSLE	**0.305**	0.314	0.870
MAE	**0.473**	0.494	0.721

的少样本学习能力。

实验结果显示，随着 α 值的增加，预测模型在测试集上的误差呈现出典型的边际收益递减趋势。具体而言，当 $\alpha \leq 0.1$ 时，由于训练样本过于稀疏，无法覆盖特征空间的主要流形，导致预测误差较大；而当 α 提升至 0.2 时，模型性能出现显著的拐点，均方根对数误差（RMSLE）迅速收敛至 0.3 左右；此后继续增加 α 值，误差下降的幅度趋于平缓。这一发现表明，SMore 所设计的特征工程方案有效提取了负载间资源干扰的核心规律，使得预测器仅需约 20% 的稀疏采样数据即可训练出具备良好泛化能力的模型。这一特性显著降低了系统冷启动时的离线分析开销，使得在上线新模型时能够快速建立性能预测能力。

5.2.1.2　回归模型性能对比与机理分析

基于选定的采样比例（$\alpha = 0.2$），本文进一步对比了三种代表性的机器学习回归算法——随机森林、Bagging 集成算法以及线性回归——在性能退化预测任务上的表现。评估指标选取均方根对数误差 RMSLE 和平均绝对误差 MAE，旨在衡量预测值与真实值之间的偏离程度，实验结果如表 5.2 所示。

从表 5.2 的数据可以观察到，随机森林模型在各项指标上均取得了最佳性能，其 RMSLE 仅为 0.305，MAE 为 0.473。相比之下，线性回归模型的表现最差，其 RMSLE 高达 0.870。这一显著差异揭示了 GPU 混合部署场景下干扰机制的复杂性：不同深度学习算子对计算单元（CUDA Cores）、显存带宽及缓存资源的争抢呈现出高度的非线性特征。线性模型难以拟合这种复杂的资源竞争关系，而基于决策树的集成方法（如随机森林）能够更好地处理高维特征间的非线性交互，从而实现更精准的预测。

从表 5.2 的量化数据中可以观察到，基于决策树的集成学习方法显著优于简单的线性模型。其中，随机森林模型在各项指标上均取得了最佳性能，其 RMSLE 仅为 0.305，相较于线性回归模型（RMSLE=0.870）降低了约 65%。Bagging 算法表现次之，但仍大幅优于线性回归。相较而言随机森林模型凭借其对非线性映射和高维特征交互的强大拟合能力，被选定为 SMore 的默认性能退化预测器，为调度决策提供了可

靠的量化依据。

5.3 冷启动优化策略验证

本节旨在系统验证 SMore 中集成的 LS-LSTM 智能预热机制在真实云环境下的有效性。作为连接上层无服务器应用与底层 GPU 资源的关键组件，预热器的核心挑战在于如何在极具动态性的负载波动中寻找到服务质量与资源效率的最佳平衡点。具体而言，实验重点考察 LS-LSTM 算法是否能够在显著降低函数冷启动率（即减少用户感知延迟）的同时，有效抑制因过度预热或无效保活而导致的显存资源浪费。

5.3.1 实验设置与基准对比

为了确保评估结果具有工业界的参考价值，本文选取了微软 Azure Functions 公开数据集中的真实生产环境跟踪日志作为驱动负载。这些跟踪日志涵盖了从周期性心跳任务到高度突发性用户请求等多种流量模式。实验构建了一个包含模拟时间轴的回放环境，并将本文提出的 LS-LSTM 策略与学术界两类主流的基于统计学的直方图策略进行了横向对比：

1. **LSTH (Long-Short Term Histogram)[45]**：该策略维护请求到达间隔的长期与短期直方图分布。它倾向于捕捉负载的统计学规律，通过计算分位点来动态调整容器的预热窗口与保活时长。
2. **HHP (Hybrid Histogram Policy)[49]**：该策略侧重于分析固定时间窗口内的空闲时间分布。它通过识别历史上使得冷启动概率最小化的空闲阈值，来确定最佳的资源回收时机，是一种典型的反应式策略。

评估指标主要包含两个维度：**冷启动率**，定义为触发冷启动的请求数占总请求数的比例，直接反映用户体验；**资源浪费率**，定义为容器处于活跃状态但未处理请求的时间占总活跃时间的比例，反映了集群资源的无效占用情况。

5.3.2 流量预测模型的拟合度分析

准确的流量预测是实现主动式预热的前提。图 5.1 展示了 LS-LSTM 模型在随机抽样的一条真实函数跟踪日志上的预测表现。图中橙色曲线代表真实的请求到达数，蓝色曲线代表模型的单步预测值，绿色虚线划分了训练集与测试集的时间边界。

从时序拟合的角度来看，LS-LSTM 展现出了对复杂流量模式的极强捕捉能力。首先，在训练集与测试集的过渡阶段，模型并未出现明显的精度跳变，证明了其并未简单记忆历史数据，而是学习到了流量演变的潜在规律，具备良好的泛化性。其次，

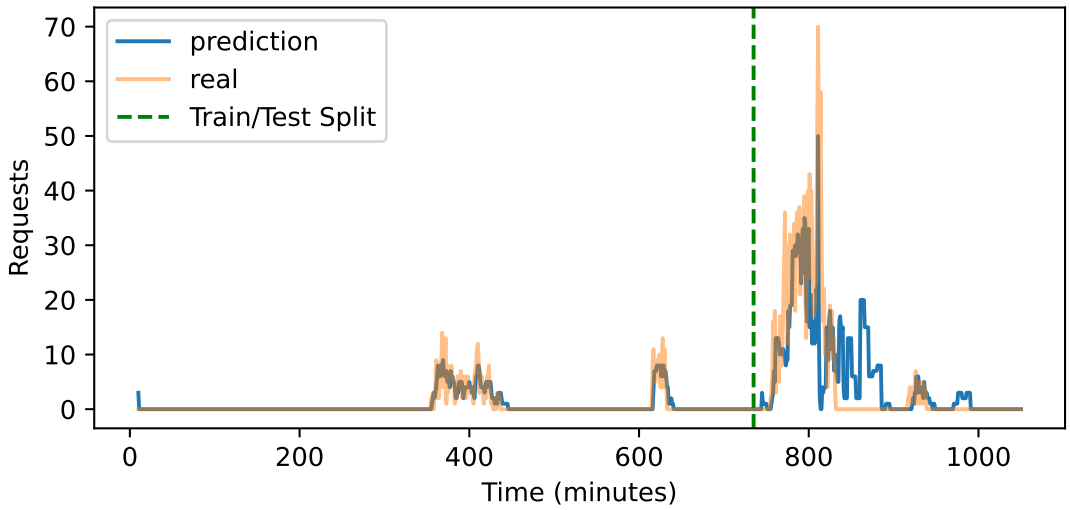

图 5.1　LS-LSTM 在真实函数跟踪日志上的流量到达预测表现
Figure 5.1　Prediction Performance of LS-LSTM on Real-World Function Traces

模型精准地复现了流量的两个关键特征：周期性与突发性。在图示的第 750 至 850 分钟区间内，面对高强度的连续突发请求，预测曲线紧密跟随真实峰值，这意味着系统能够提前感知负载洪峰并预留充足的 GPU 实例；而在第 450 至 600 分钟的静默期，模型预测值迅速归零，这为调度器安全回收资源提供了高置信度的信号。这种基于深度神经网络的时序建模能力，从根本上优于基于简单移动平均或线性回归的传统方法。

5.3.3　冷启动与资源效率的权衡分析

基于高精度的预测结果，本文进一步量化分析了 LS-LSTM 策略在不同负载特征下的综合效能。为了深入剖析算法的适应性，实验选取了两类具有代表性行为特征的函数进行详细对比分析：

- **Function 1（准周期型负载）**：请求到达呈现出一定的规律性，突发程度较低，易于通过历史统计进行推断。
- **Function 2（高突发型负载）**：请求间隔极不均匀，且伴随不可预测的长时间静默与瞬间爆发，属于典型的长尾异构负载。

实验结果如图 5.2 所示，左侧子图展示了冷启动率对比，右侧子图展示了对应的资源浪费情况。

针对 Function 1（准周期型负载），LS-LSTM 策略展现了激进的性能优化能力。如图所示，相比于 LSTH 和 HHP 接近 18% 的冷启动率，LS-LSTM 将该指标显著降低

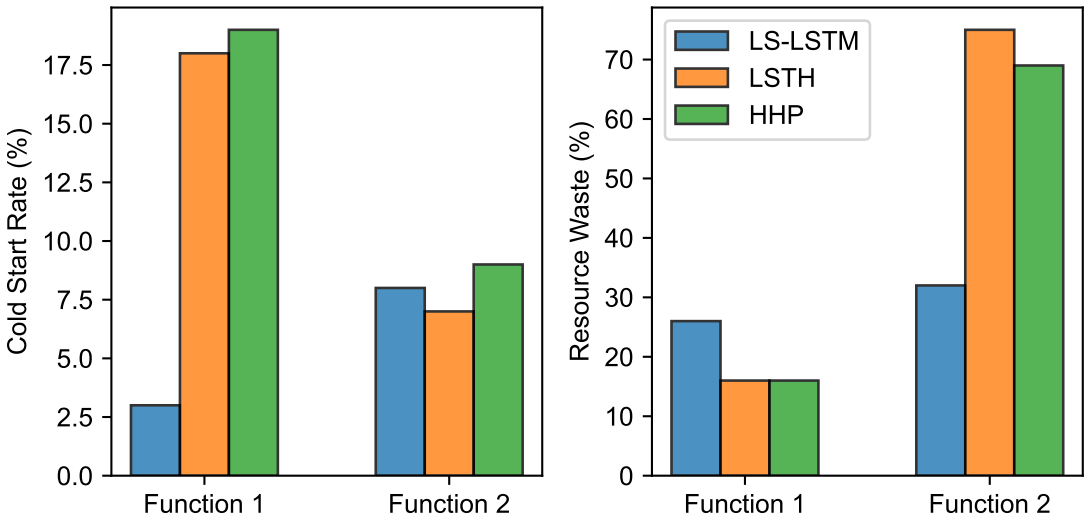

图 5.2 LS-LSTM 与现有基线策略在冷启动率与资源浪费上的对比

Figure 5.2 Comparison of Cold Start Rate and Resource Wastage between LS-LSTM and Baseline Policies

至 3% 以下。这一改进主要得益于 LS-LSTM 对周期性模式的精确锁定，使得预热操作能够精确地发生在请求到达前的毫秒级窗口内。虽然这种激进的预热策略导致资源浪费略高于基线方法（约高出 10%），但在追求极致低延迟的在线推理场景中，以少量的资源冗余换取数量级的延迟下降是极具性价比的权衡。

针对 Function 2（高突发型负载），LS-LSTM 在性能与成本上实现了协同优化，这也正是本文方法的优势核心所在。从右侧子图可以看出，传统的 LSTH 和 HHP 策略在处理此类不规则负载时失效严重。由于无法预知下一次请求何时到达，这些基于直方图的方法倾向于采取保守策略，即大幅延长容器的保活时间以覆盖潜在请求。这导致资源浪费率飙升至 70% 以上，意味着 GPU 显存绝大部分时间都在空转。相比之下，LS-LSTM 凭借对静默期的准确预判，敢于在请求间隙果断释放资源。实验数据显示，在保持与基线相当甚至更低的冷启动率前提下，LS-LSTM 将资源浪费率从 75% 大幅压缩至 32% 左右。这意味着在相同的物理集群上，SMore 能够支持两倍以上的并发函数实例，极大地提升了深度学习集群的资源整合密度。

综合以上结果分析得到，LS-LSTM 预热策略克服了传统统计学方法在面对高动态、长尾负载时的局限性。它通过时间换空间的智能预测，不仅保障了关键推理任务的 SLO，更在本质上消除了盲目保活带来的资源黑洞，证明了其作为服务器无感知计算核心调度组件的必要性与优越性。

5.4　混合部署系统性能验证

本节旨在通过真实的物理测试平台实验，全面评估 SMore 在异构负载混合部署场景下的综合性能。实验设计遵循由简入繁的原则：首先在单节点单 GPU 环境下，以利用率相对稳定的常规服务器式训练任务为基准，细粒度剖析混合部署对任务延迟与资源利用率的微观影响；随后扩展至多节点多 GPU 集群环境，模拟真实的生产级并发场景，验证系统在复杂负载组合下的鲁棒性与资源调度效率。

5.4.1　单应用混合部署性能分析

5.4.1.1　实验设置与基准场景

为了构建高保真的异构混部环境，本文在单个 GPU 节点上部署了 MobileNet 模型作为高优先级的长时驻留任务，模拟生产环境中常见的在线推理服务或轻量级训练作业。该模型具有适中的计算密度，能够为混部实验留出可观测的资源空隙。

干扰负载方面，实验利用 Azure Functions 生产环境跟踪日志生成器，构造了一个持续 20 秒、包含约 390 个并发请求的脉冲式流量负载。这些请求涵盖了从计算密集型（如 VGG）到访存密集型（如 BERT）的多种深度学习推理函数。系统核心目标是在严格保障主负载性能（即性能退化不超过 10%）的前提下，最大化服务器无感知函数的准入量与执行效率。

5.4.1.2　混合部署下的服务质量保障分析

在资源共享环境中，核心挑战在于如何隔离异构负载间的资源争抢。图 5.3 展示了在混合部署期间，被系统准入的服务器无感知函数相对于其独占执行时的延迟退化率的累积分布函数。

从统计数据来看，SMore 展现出了优异的干扰控制能力。首先，曲线呈现出陡峭的上升趋势，表明绝大多数任务运行在低干扰区间。具体而言，超过 50% 的函数性能退化率被控制在 10% 以内，这意味着对于半数以上的请求，用户几乎感知不到混合部署带来的延迟损耗。其次，从尾部延迟来看，约 95% 的任务退化率低于 100%（即延迟未翻倍）。这说明即便是面对不可避免的瞬时资源拥塞，调度器也能通过细粒度的上下文切换与流控制，有效防止任务因长期无法获取资源而导致的执行停滞。这一结果验证了本文提出的干扰感知调度算法能够精准识别 GPU 的微秒级空闲时间窗，实现了异构负载在时间维度上的安全隔离。

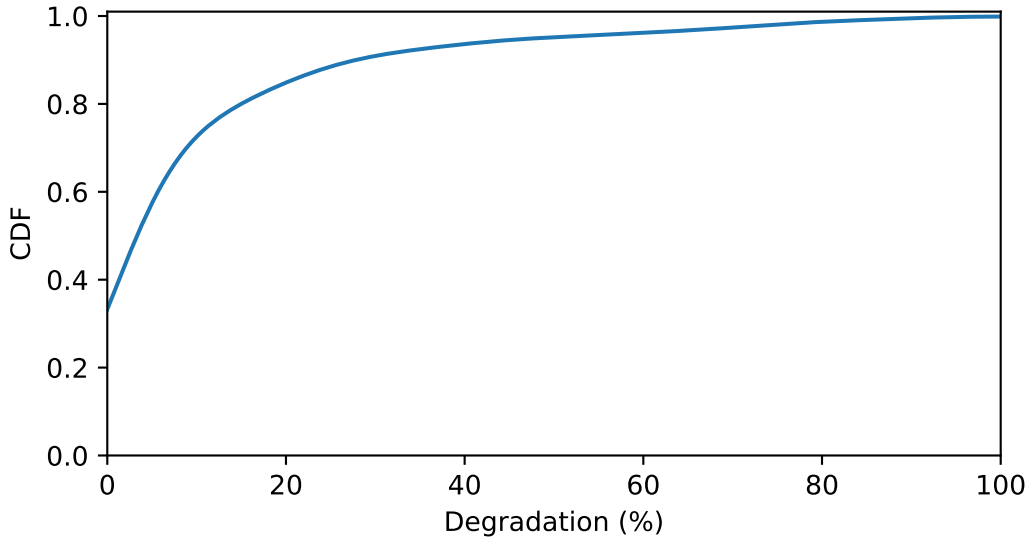

图 5.3 混合部署场景下服务器无感知函数的延迟退化分布
Figure 5.3 The Degradation Distribution of Serverless Functions under Co-Location

5.4.1.3 调度策略的吞吐量与选择性分析

调度器的核心效能在于其在资源饱和约束下的并发处理能力。图 5.4 记录了实验时间轴上服务器无感知请求的提交量与系统实际准入量的动态变化。

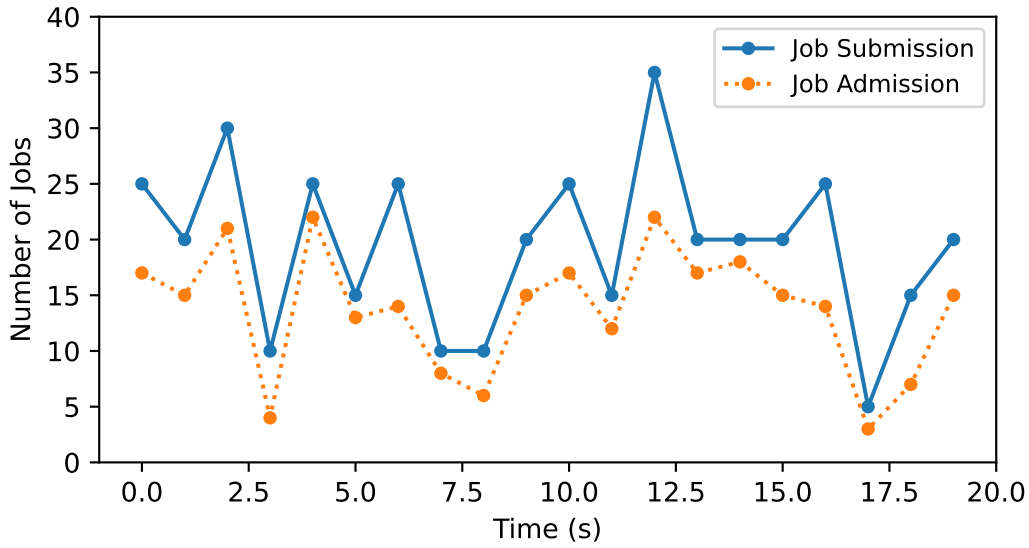

图 5.4 SMore 在单应用场景下的任务提交与准入统计
Figure 5.4 Number of Submitted and Admitted Tasks by SMore over Time

分析图 5.4 可知，系统在高并发场景下保持了极高的吞吐鲁棒性。在流量相对平

稳的区间（如 0-2s, 13-16s），准入曲线与提交曲线几乎完全重合，表明系统能够消化绝大部分由于主负载计算间隙产生的碎片资源。即便在流量突发的峰值时刻（如第 12 秒，瞬间并发约 35 个请求），系统依然维持了约 65% 的准入率，仅拒绝了那些预计会导致严重资源冲突的请求。

为了进一步揭示调度器的决策逻辑，图 5.5 统计了不同类型服务器无感知函数的平均准入比例。实验结果呈现出鲜明的模型敏感性特征：

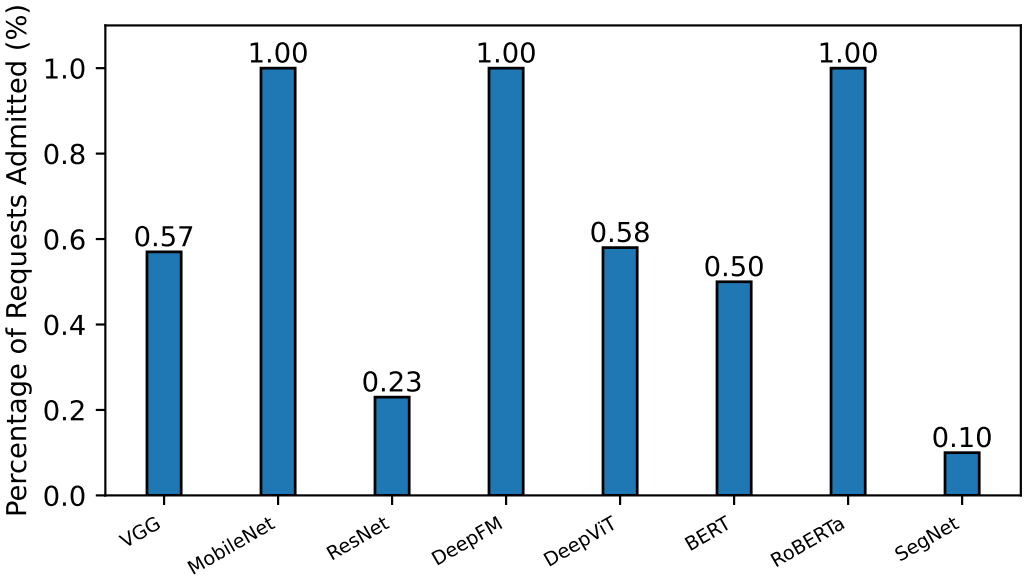

图 5.5　SMore 对不同类型函数的动态准入比例
Figure 5.5　Admitted Ratio of Different Function Types by SMore over Time

- **高准入组（MobileNet, DeepFM, RoBERTa）**：这类模型的准入率接近 100%。原因在于它们要么是轻量级网络（MobileNet），要么是稀疏计算特征（DeepFM），与主负载 MobileNet 的资源竞争较小，容易被塞入计算流水线的空隙中。
- **受限准入组（VGG, DeepVIT, BERT）**：准入率被控制在 50% 左右。这类模型属于典型的计算或访存密集型负载，若无限制准入，极易引发 L2 Cache 颠簸或显存带宽饱和。
- **低准入组（SegNet）**：准入率仅为 10%。SegNet 作为高分辨率语义分割模型，对显存容量和算力均有极高需求。调度器的预测模型准确判断出其高干扰风险，从而执行了严格的准入控制。

这一差异化准入策略表明，SMore 并非盲目追求并发量，而是能够在准入收益与干扰代价之间做出有意识的权衡：对低干扰模型保持近乎完全准入的同时，有效抑制高干扰模型的并发度，从而在整体吞吐和主负载性能之间取得更为平衡的系统效益。

5.4.1.4 资源利用率提升效果

引入服务器无感知混合部署的根本动力在于挖掘 GPU 的闲置算力。图 5.6 直观对比了仅运行主负载与开启混合部署两种模式下的 GPU 利用率时序变化。

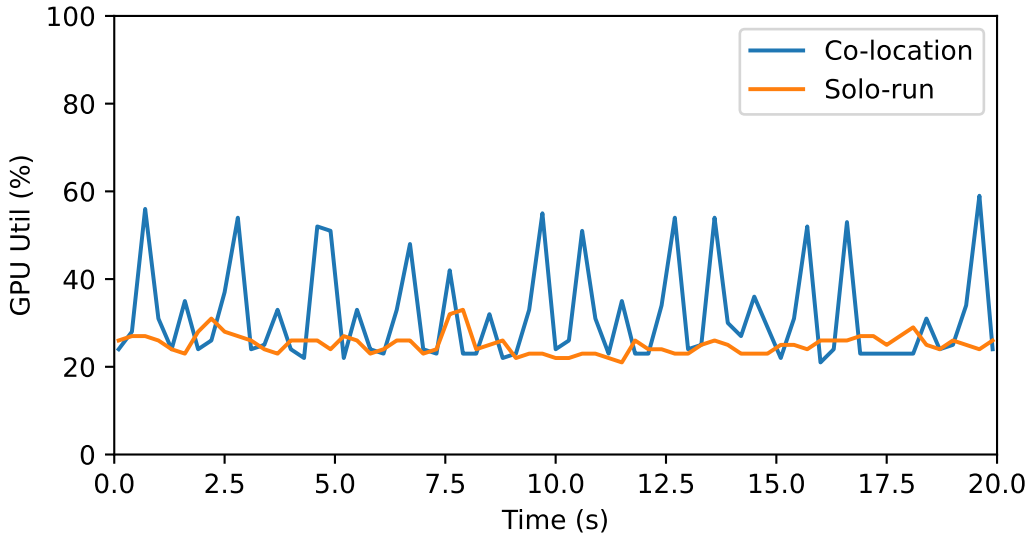

图 5.6 独占执行与混合部署模式下的 GPU 资源利用率对比
Figure 5.6 Resource Utilization Comparison between Exclusive and Co-Location Execution

在独占执行模式下，由于 MobileNet 主负载存在固有的 CPU-GPU 数据传输开销及 Kernel 启动间隙，GPU 利用率长期维持在 20%-30% 的低水平，造成了巨大的算力浪费。而在混合部署模式下，SMore 成功通过资源收割机制填补了这些微小的空闲时间片。每当有服务器无感知请求被准入，GPU 利用率便会出现显著跃升，峰值多次突破 50% 甚至达到 60%。这种利用率的动态波动与前文的任务准入曲线高度吻合，证实了系统成功将原本被闲置的 GPU 周期转化为了实际的服务器无感知计算产出，在未扩容硬件的前提下显著提升了集群的整体算力密度。

5.4.2 多应用并发混合部署性能分析

在验证了单节点环境下的系统基本功能后，本节将实验范畴扩展至包含 8 个 GPU 节点的集群环境，旨在评估 SMore 在模拟生产级多租户场景下的水平扩展性与资源调度鲁棒性。

5.4.2.1 实验配置与负载场景构建

为了全面覆盖数据中心中多样化的资源竞争模式，实验设计了六种具有代表性的基准测试场景。首先，在传统服务器式主负载的选择上，本文选取了三种资源特征

迥异的深度学习模型：

- **MobileNet**：代表低计算负载，GPU 占用率低，留有大量资源空隙。
- **DeepFM**：代表访存密集型或稀疏计算负载，对 CUDA Core 占用不高，但对显存带宽敏感。
- **RoBERTa**：代表高算力负载，GPU 计算单元接近饱和。

其次，为了模拟不同压力下的显存碎片化程度，实验设置了同构与异构两类批处理配置。其中，异构模式指的是在同一节点上动态混合不同批处理大小的训练任务，这种配置会加剧显存空间的碎片化，从而对调度器的资源装箱能力提出更高挑战。在此背景下，系统向集群注入了高强度的服务器无感知背景流量，峰值到达率提升至 12,000 requests/min，以测试系统在极端压力下的极限性能。

5.4.2.2　性能隔离与退化控制分析

图 5.7 对比了不同负载配置下，常规服务器式主任务与务器无感知函数的的平均性能退化程度。从图中可以观察到两个关键现象：

第一，**主任务的性能零损耗**。图中蓝色柱状图表示的传统服务器式负载的性能退化在所有场景下均维持在极低水平（< 6%），部分场景（如 MobileNet-same）甚至接近于 0。这表明 SMore 的调度策略成功将服务器无感知负载限制在次等的资源层级，确保了关键业务的运行稳定性。

第二，**服务器无感知任务的受控退化**。橙色柱状图表示的服务器无感知负载的性能退化显示，即便是在干扰最激烈的 **BERT-mix** 场景下，服务器无感知函数的平均退化率也被控制在 39% 左右（即执行时间增加约 1.4 倍）。考虑到服务器无感知任务属于尽力而为的服务类型，这一性能损耗在可接受的范围内。值得注意的是，在 **MobileNet-same** 等低负载场景下，退化率相对较高（33%），这是因为调度器为了提升吞吐量，在该场景下激进地准入了大量并发函数，导致服务器无感知任务内部产生了排队竞争，但这恰恰体现了系统在资源充裕时追求高吞吐的调度倾向。

5.4.2.3　集群级任务准入与负载均衡

图 5.8 记录了集群环境下的任务提交量与准入量随时间的变化曲线。与单机实验相比，多节点集群展现出了显著的资源池化效应。

实验数据显示，系统准入曲线密跟随提交曲线，在绝大多数时间窗内实现了接近 100% 的任务接纳率。这种优异的表现归功于全局调度器的负载均衡能力：当某个节点因运行大批处理大小的 BERT 任务而资源枯竭时，调度器能够迅速将新到达的服务器无感知请求重定向至运行 MobileNet 的空闲节点。这种跨节点的资源互补机制，

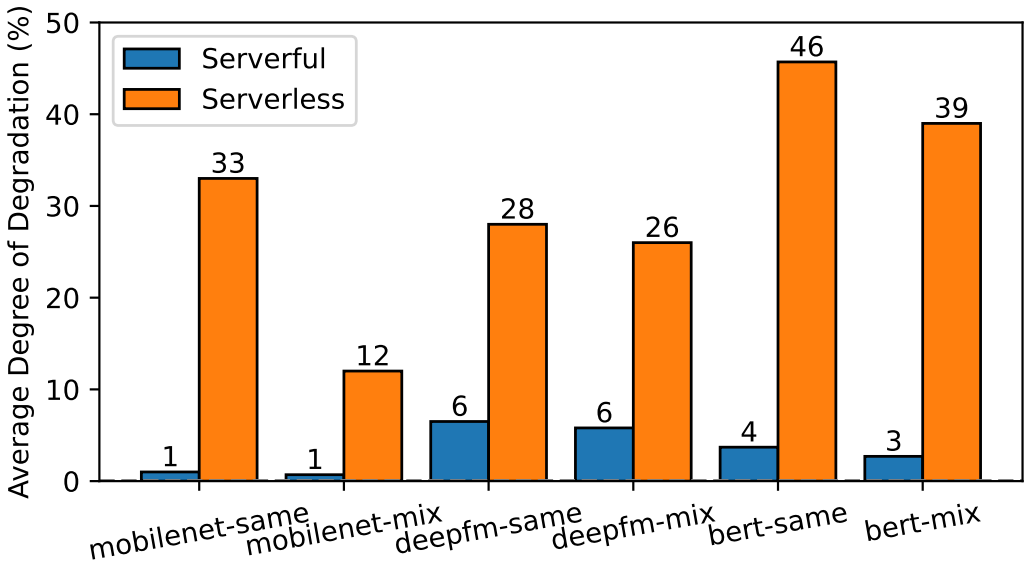

图 5.7 多应用场景下独占与混合部署的性能退化对比

Figure 5.7 Comparison of Degradation between Exclusive and Co-Location Execution in Multi-Application Scenarios

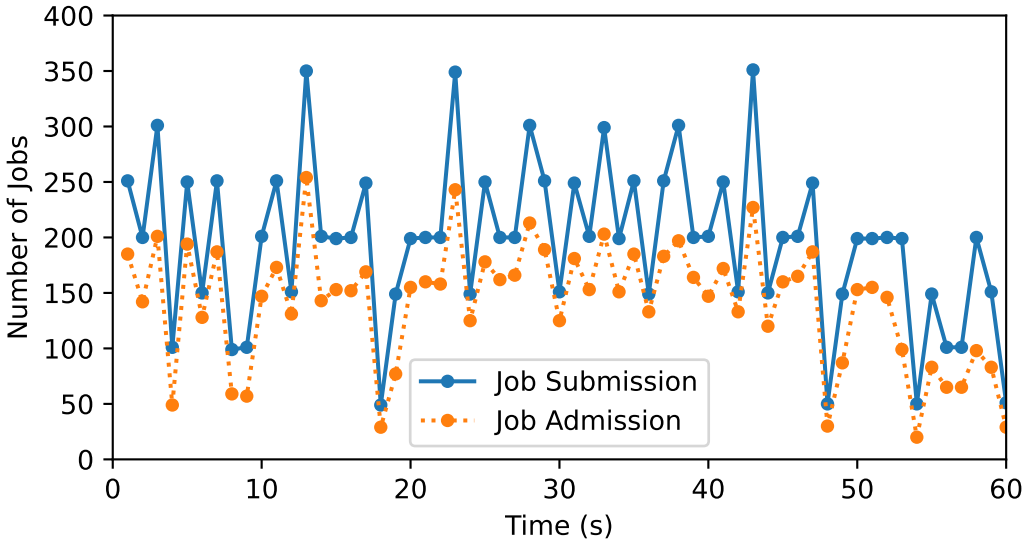

图 5.8 集群环境下 SMore 的任务提交与准入统计

Figure 5.8 Number of Submitted and Admitted Functions by SMore over Time in Cluster Environment

有效平抑了局部负载热点，证明了 SMore 具备良好的水平扩展能力，能够充分挖掘大规模集群中的分布式资源碎片。

5.4.2.4　资源利用率收益与饱和度分析

图 5.9 总结了六种配置下 GPU 流处理器 SM 利用率的提升效果，深刻揭示了混合部署在不同算力饱和度下的边际收益规律。

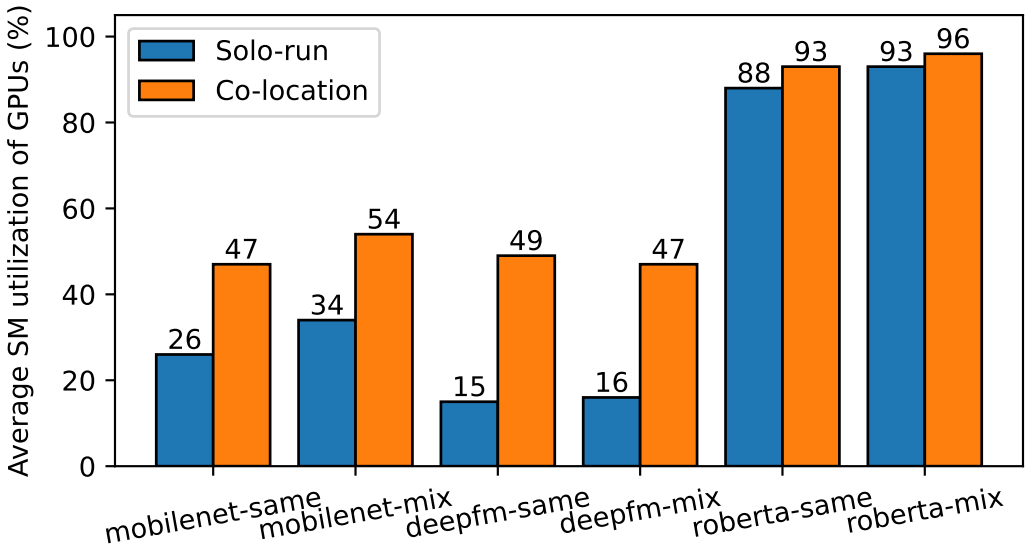

图 5.9　多应用场景下各配置组合的 GPU 利用率提升效果

Figure 5.9　Comparison of Utilization between Exclusive and Co-Location Execution in Multi-Application Scenarios

分析该图表可得出以下结论：

- **低负载场景下的收益最大化**：对于 DeepFM 这类具有稀疏计算特征的模型，独占执行时 SM 利用率仅为 15%-16%。引入 SMore 后，利用率激增至 47%-49%，提升幅度超过 3 倍。这证明系统成功识别并填充了大量的计算气泡。
- **中等负载场景的稳健提升**：对于 MobileNet，混合部署将利用率从约 30% 提升至 50% 以上，收益依然显著。
- **高负载场景的安全边界**：对于 RoBERTa，独占利用率已高达 88%-93%。在此极端饱和的状态下，SMore 仅将利用率微幅提升了约 3%-5%。这一现象并非系统失效，而是调度器安全意识的体现。预测模型准确判断出此时 GPU 已无多余算力，因此严格限制了服务器无感知任务的准入，从而避免了强制塞入任务导致的系统崩溃或主任务严重的性能雪崩。

综合以上结果分析得到，SMore 在多应用并发环境下，不仅能够显著提升集群整体资源利用率，更重要的是它具备感知饱和度的智能决策能力，能够在追求效率与保障稳定性之间动态寻找最佳平衡点。

5.5 调度策略对比分析

本节通过横向对比实验，进一步论证 SMore 在保障服务质量与提升资源效率方面的综合优势。为了确保对比的公平性与权威性，本文基于 TGS[73] 提供的联合执行框架，复现了三种代表性的调度策略作为基线，并将其与 SMore 在相同的高负载环境（请求到达率设为 16,000 requests/min）下进行性能比较。

5.5.1 基线方法定义与实验设置

为了全面评估调度器的性能边界，本文选取了涵盖静态规则到动态感知不同演进阶段的三种策略：

1. **Random（随机调度）**：作为最朴素的对照组，该策略遵循先来先服务（FCFS）原则。任务到达后被随机分配至任意显存满足要求的 GPU，完全忽略潜在的微架构资源争抢与性能退化风险。
2. **EDF-util（基于利用率阈值的最早截止时间优先）**：该策略结合了实时系统中的 EDF 算法与静态资源预留机制。系统优先调度截止时间紧迫的请求，但设置了严格的准入控制：仅当目标 GPU 当前的常规服务器式负载利用率与新请求预估利用率之和低于设定阈值（本实验设为 80%）时，才允许调度。
3. **ElasticFlow[24]**：代表了当前服务器无感知计算领域的先进水平。ElasticFlow 同样基于 EDF 进行任务排序，但引入了拓扑感知的贪婪放置算法，旨在最小化单 GPU 的吞吐量干扰。为了适配本文的混合部署场景，本文将 ElasticFlow 的核心逻辑适配到 TGS 框架中，使其具备基本的异构混部调度能力。

图 5.10 综合展示了各调度策略在四个关键指标上的性能表现：截止时间满足率（Deadline Satisfaction Rate, DSR）、GPU 利用率提升（GPU Utilization Improvement, GUI）、常规服务器式负载退化（Serverful Degradation, SFD）以及服务器无感知函数退化（Serverless Degradation, SLD）。

5.5.2 服务质量保障能力分析

截止时间满足率 DSR 是衡量服务器无感知计算平台可用性的核心指标。如图 5.10 左侧所示，调度策略是否具备细粒度干扰感知能力对 DSR 具有决定性影响。

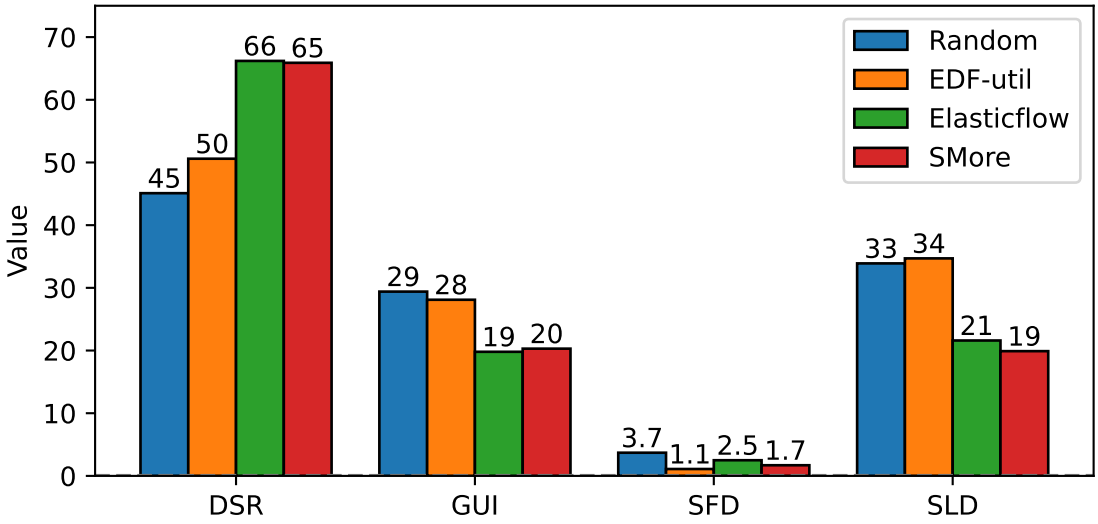

图 5.10　SMore 与现有基线调度策略的多维性能对比

Figure 5.10　Performance Comparison between SMore and Baseline Scheduling Methods across Multiple Metrics

非感知类策略的局限性：Random 策略的 DSR 仅为 45%，表现最差。由于缺乏对资源竞争的建模，大量计算密集型任务被错误地调度到了繁忙节点，导致执行时间失控。EDF-util 虽然提升至 50%，但其静态阈值机制（Utilization Threshold）过于粗糙，无法反映 Cache 或带宽层面的隐式冲突，导致大量本可准入的短任务被误杀，或准入后依然发生拥塞。

感知类策略的优越性：相比之下，ElasticFlow（66%）与 SMore （65%）展现出了断层式的优势。这表明引入性能预测机制是保障 SLO 的必要条件。值得注意的是，SMore 在 DSR 指标上与 ElasticFlow 持平，但在后续分析的代价指标（SFD/SLD）上取得了显著突破，这意味着 SMore 是以更低的系统损耗达成了相同的高服务质量。

5.5.3　主任务保护与任务隔离性分析

在混合部署系统中，不影响主任务是需要严格遵守的前提。图 5.10 中的 SFD（主任务退化率）和 SLD（服务器无感知任务退化率）揭示了各策略在隔离性上的本质差异。

常规服务器式负载退化 SFD：

- Random 策略造成了高达 3.7% 的主任务性能损耗，这在生产环境中通常是不可接受的。
- ElasticFlow 将该指标降低至 2.5%，但由于其干扰模型主要针对同构推理任务设

计，对异构训练任务的特征捕捉不够敏锐。

- **SMore 实现了 1.7% 的极低干扰**，仅次于极度保守的 EDF-util（1.1%）。需要指出的是，EDF-util 的低 SFD 是通过牺牲大量 DSR（拒绝任务）换来的，而 SMore 是在维持高 DSR 的前提下实现的。这证明了本文提出的时空联合干扰预测模型能够精准识别安全的执行空隙，真正做到了对主任务的透明运行。

服务器无感知函数退化 SLD：SMore 以 19% 的 SLD 优于 ElasticFlow（21%），并远低于 Random（33%）和 EDF-util（34%）。这意味着被 SMore 调度的函数，其执行效率不仅受主任务影响最小，而且在函数间的互干扰也得到了最优控制。

5.5.4 资源利用率提升结果分析

从结果数据来看，Random（29%）和 EDF-util（28%）在 GPU 利用率提升（GUI）指标上获得了最高数值。然而，这两种策略的利用率提升主要依赖于在装箱决策中忽略干扰风险、强行提高集群负载，其代价是显著的性能退化和极低的服务质量保障：Random 的 SFD 为 3.7%，SLD 为 33%，SLO 达成率（DSR）仅为 45%。在实际工程环境中，这类通过牺牲任务性能和用户体验换取的高利用率缺乏实际应用价值。

相比之下，SMore（20%）和 ElasticFlow（19%）在 GUI 指标上虽略低于 Random 和 EDF-util，但两者在 SFD、SLD 以及 DSR 等指标上均表现出更好的服务质量与稳定性，体现了更具工程可用性的有效利用率提升。进一步对比可以发现，在 SFD 和 SLD 均优于 ElasticFlow 的前提下，SMore 仍然实现了更高的 GUI（20% vs 19%）。在超大规模集群场景下，即便 1% 的利用率提升也对应着显著的资源成本节约。

这一结果表明，SMore 并非通过保守的调度策略来避免干扰，而是依托更精细的干扰边界识别与建模能力，在保证性能隔离和 SLO 达成率的前提下采用了相对更激进的装箱策略。因此，相较于 ElasticFlow，SMore 在集群稳定性与资源利用率之间实现了更优的综合权衡。

5.6 系统扩展性与泛化能力验证

前文的实验主要基于现有的物理测试平台验证了 SMore 在标准 GPU 架构下的性能优势。本节将进一步从集群规模扩展性和新型硬件适配性两个维度，评估系统的鲁棒性与未来适用性。具体而言，本文将通过大规模数值模拟验证调度算法的时间复杂度，并通过引入 NVIDIA Multi-Instance GPU (MIG) 技术验证系统在硬件虚拟化环境下的泛化能力。

5.6.1　大规模集群调度开销模拟

调度器的决策延迟是衡量分布式资源管理系统可扩展性的关键指标。特别是对于生命周期极短的服务器无感知函数而言，调度开销必须被严格控制在极低的范围内（通常要求为亚毫秒级），以避免因调度滞后抵消 GPU 加速带来的性能收益。鉴于物理测试平台规模（8 GPU）的局限性，本节基于真实测得的性能基准数据，构建了一个高保真的离散事件模拟器，旨在评估 SMore 在从 64 扩展至 1024 个 GPU 节点规模时的性能表现。

5.6.1.1　模拟环境与参数设置

模拟实验的核心逻辑完全复用了 SMore 的代码，包括准入控制算法与资源搜索策略，仅将底层的物理节点交互层替换为模拟桩。为了覆盖不同的集群运行状态，实验设计了两种典型的负载场景：

- **低负载模式**：模拟集群资源相对空闲的状态。此时，调度器需要扫描较大的解空间以执行负载均衡策略，试图找到最优的放置节点。
- **高负载模式**：模拟集群资源接近饱和的状态。此时，调度器面临激烈的资源竞争，核心目标转变为快速寻找可行解或执行快速拒绝。

实验统计指标为每处理 1000 个服务器无感知请求的累计调度时间（ms），以此来平滑单次测量的抖动。

5.6.1.2　调度延迟随规模变化的趋势分析

图 5.11 展示了在不同集群规模与负载压力下的调度开销统计结果。从宏观趋势来看，SMore 展现出了卓越的水平扩展能力。

首先，**亚毫秒级的单次决策响应**是系统高性能的直接体现。实验数据显示，即便在集群规模扩大至 1024 个 GPU 节点的最坏情况（低负载模式）下，处理 1000 个请求的总耗时约为 795 ms。换算为单次调度开销，平均仅需 0.795 ms。考虑到典型的服务器无感知深度学习推理任务的执行时间通常在数百毫秒至数秒量级，这一调度开销在端到端延迟中的占比不足 0.1%，几乎可以忽略不计。这证明了 SMore 的轻量级调度架构完全能够支撑千卡规模的数据中心应用。

其次，数据呈现出显著的负载敏感性特征。对比两条曲线可知，高负载模式下的调度开销显著低于低负载模式。以 1024 节点为例，高负载下的每千次请求耗时仅为 106 ms（即单次约 0.1 ms），速度提升了近 7.5 倍。

这一现象深入揭示了 SMore 采用的首次适应启发式搜索策略的高效性。在低负

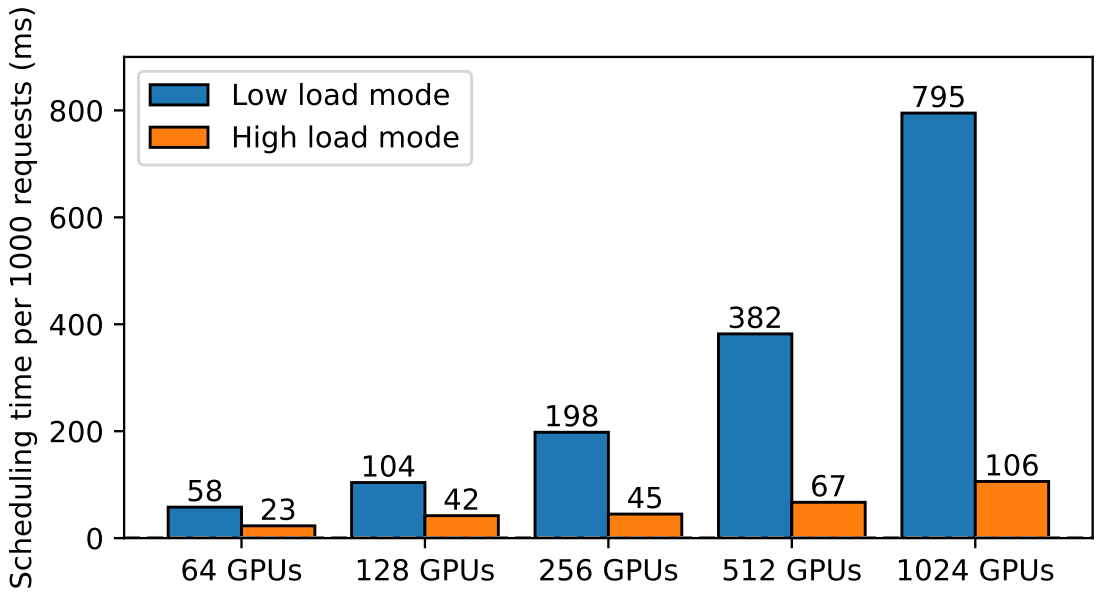

图 5.11 不同集群规模与负载压力下的调度延迟模拟结果

Figure 5.11 Scheduling Overhead under Large-Scale Simulation Experiments with Varying Cluster Sizes and Loads

载环境下，由于集群中存在大量可用节点，调度器为了实现更优的负载均衡，往往需要遍历更多的候选节点并进行打分比较，从而导致了随集群规模线性增长的搜索开销。相反，在高负载环境下，资源碎片化严重，调度器采用激进的剪枝策略：一旦扫描到首个满足资源约束的空隙节点，便立即终止搜索并锁定资源；或者在全局资源视图判断饱和时实现 $O(1)$ 复杂度的快速拒绝。这种早停机制使得调度器在系统最繁忙、压力最大的时刻，反而能够保持最高的决策效率，从而有效避免了调度器本身成为系统的性能瓶颈。

5.6.2 基于 MIG 的异构资源适配性验证

随着数据中心 GPU 硬件架构的演进，以 NVIDIA A100/H100 为代表的计算加速卡引入了多实例 GPU（Multi-Instance GPU, MIG）技术。该技术允许将单个物理 GPU 在硬件层面划分为多个完全隔离的计算实例，每个实例拥有独立的显存带宽、二级缓存和计算单元。为了验证 SMore 是否能够适应这种硬隔离环境下的资源管理模式，并具备对下一代硬件特性的前瞻兼容性，本节开展了基于 MIG 的异构资源适配性实验。

5.6.2.1　实验环境与分区配置

实验平台选用单块 NVIDIA A100-PCIE-40GB 加速卡。为了构建具有代表性的基线环境，本文采用了 MISO[74] 系统的分区策略。MISO 是一种专为 MIG 设计的动态分区调度器，能够根据任务需求将 GPU 划分为不同规格的实例。在此基础上，SMore 被部署为上层调度器，负责监控 MISO 分配后的实例状态，并利用实例内部的剩余算力碎片或暂时空闲的实例进行服务器无感知函数的微调度。

实验设计了三种典型的 MIG 分区拓扑，涵盖了从粗粒度到细粒度的不同组合（Ng 代表该实例包含 N 个 GPU 切片）：

1. **4g-3g**：模拟双租户的大模型训练场景，资源被大致均分。
2. **3g-2g-2g**：模拟混合负载场景，包含一个中型任务和两个小型任务。
3. **4g-2g-1g**：模拟长尾分布场景，存在极小粒度（1g）的实例。

实验中持续注入峰值速率为 4,500 requests/min 的服务器无感知推理流量，重点考察系统在硬件强约束下的资源收割能力。

5.6.2.2　硬隔离环境下的性能表现

图 5.12 详细记录了在不同 MIG 分区配置下，系统的截止时间满足率 DSR、GPU 利用率提升 GUI、常规服务器式负载退化 SFD 与服务器无感知函数退化 SLD。

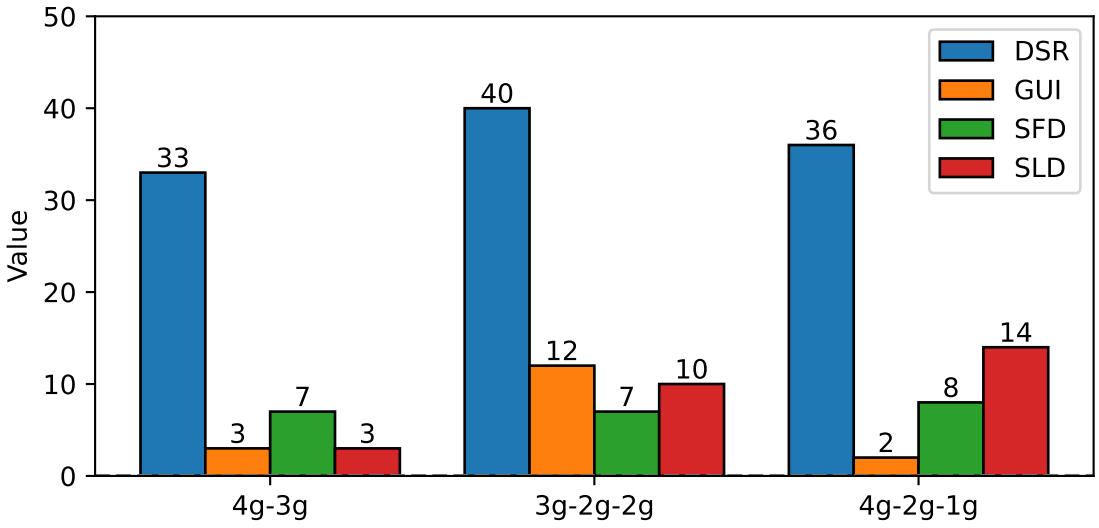

图 5.12　启用 MIG 的 A100 GPU 上不同负载组合的性能对比
Figure 5.12　Comparison Results of Different Workload Groups on MIG-Enabled A100 GPUs

从图 5.12 中的橙色柱状图 GUI 可以看出，SMore 在所有分区配置下均实现了正向的资源利用率提升，但提升幅度表现出显著的拓扑敏感性。

- 在 **3g-2g-2g** 的配置下，系统取得了最显著的利用率收益（GUI ≈ 12%），同时保持了较高的动态调度成功率（DSR ≈ 40%）。该拓扑结构提供了三个可并发执行的独立实例，为服务器无感知任务的并行调度创造了更多机会。此外，2g 与 3g 实例的显存容量适中，既能保障主推理任务的稳定运行，又留有足够的资源余量以容纳额外的无感知函数，从而实现了资源闲置间隙的有效填充。
- 与之形成对比的是 **4g-2g-1g** 配置，其利用率增益相对有限（GUI ≈ 2%）。通过对系统日志的深入分析发现，1g 实例的显存容量（约 5 GB）过于局促，难以在承载主任务的同时嵌入服务器无感知函数，致使该实例在调度层面实质上处于不可用状态，从而制约了系统整体的资源利用潜力。

相较于前文基于 CUDA MPS 的软件隔离方案（图 5.10），在 MIG 硬件隔离环境下，SMore 的动态调度成功率（DSR，33%–40%）略有下降，但其对常规服务器负载的性能退化影响（SFD）则维持在更优水平（7%–8%）。这一反差现象揭示了 MIG 技术的双重特性：

- **隔离性增强与稳定性保障（低 SFD）**：MIG 在物理硬件层面实现了计算单元与缓存（如 L2 Cache）的严格隔离，切断了不同实例间的直接干扰路径。因此，即使 SMore 的预测或调度存在一定偏差，也不会对主任务造成灾难性的性能干扰。这印证了 SMore 与 MIG 技术具备内在的互补性：MIG 提供底层的物理隔离保障，而 SMore 则在逻辑调度层实现效率优化。
- **资源刚性划分与调度灵活性受限（低 DSR）**：由于 MIG 实例间的算力与显存资源无法动态共享，当某一实例出现拥塞时，调度器无法像在统一资源池中那样，进行全卡维度的全局负载均衡。这导致部分服务器无感知请求因在单一实例内等待超时而被迫丢弃，从而降低了整体的调度成功率。

综合上述实验结果表明，SMore 不仅兼容传统的共享 GPU 架构，亦能有效适配基于 MIG 的先进硬件虚拟化环境。即使在 MIG 技术所施加的严格硬件边界约束下，该系统仍可通过精细化的微调度策略，在不破坏实例间物理隔离性的前提下，从静态划分的资源碎片中挖掘出 2% 至 12% 的额外计算效能。这充分证明了该系统在面对未来数据中心日益复杂的异构计算硬件与多样化虚拟化技术时，具备良好的泛化能力与持续的演进潜力。

5.7　本章小结

本章在 SMore 原型系统上开展了多维度的对比实验，对系统中的退化预测、冷启动优化及混合部署调度三个核心模块的性能分别进行了验证。实验结果表明，在干扰感知方面，基于随机森林的性能退化预测器表现出优异的精度，其 RMSLE 控制在 0.305 左右，能够利用少量样本有效拟合复杂的干扰关系。在冷启动优化方面，LS-LSTM 策略相比传统直方图方法显著降低了资源浪费，在部分高突发场景下将资源空转时间从 75% 降低至 32%。在系统整体性能方面，SMore 通过混合部署成功将集群 GPU 资源利用率提升了 3% 至 34%，同时将服务器无感知函数的性能退化控制在可接受范围内。与 ElasticFlow 等基线调度器相比，SMore 在保障常规服务器式负载不受干扰的前提下，显著提高了服务器无感知请求的截止时间满足率。此外，大规模模拟与 A100 MIG 实验进一步证实，本系统的调度开销极低（单次调度 <1ms），且具备良好的硬件泛化能力，能够适应未来大规模异构集群的资源管理需求。

第 6 章 总结与展望

6.1 全文总结

本文提出了一个面向深度学习集群的高效混合部署系统 SMore，旨在通过将常规服务器式的深度学习训练任务与服务器无感知的推理函数进行混合部署，以最大化 GPU 集群的资源利用率，同时解决混合部署带来的性能干扰与服务器无感知冷启动延迟高昂的问题。此外，为了在保障服务质量的前提下实现高效调度，本文设计了一套涵盖预测、调度与预热的完整架构。SMore 系统共包含三个主要模块，分别是混合部署性能退化预测模块、感知退化的调度模块以及基于序列学习的预热模块。

混合部署性能退化预测模块针对深度学习任务与服务器无感知函数在共享 GPU 资源时产生的复杂干扰问题，通过数据驱动的方法实现了对性能退化的精准量化。本文分析了不同模型结构、批处理大小以及任务组合对 GPU 计算单元和显存的竞争模式。具体地，本文结合了离线训练与在线更新机制，利用随机森林算法构建了预测模型。该模块能够根据输入的负载特征，快速推断出潜在的混合部署方案对常规服务器式任务迭代时间和服务器无感知函数执行延迟的具体影响，从而为调度决策提供可靠的数据支撑。

感知退化的调度模块利用预测模块提供的退化信息，在给定的延迟约束 SLO 和集群资源限制下，为 SMore 执行最优的任务准入与放置决策。针对传统调度器往往忽略混合部署干扰导致 SLO 违约的问题，本文设计了一种贪婪的准入控制与资源分配算法。该算法优先接纳那些对当前运行的常规服务器式负载干扰最小、且自身能满足截止时间的服务器无感知请求。通过精细化的调度策略，该模块有效地隔离了不同负载间的性能干扰，确保了关键任务的执行效率，同时最大化了集群的吞吐量。

基于序列学习的预热模块旨在解决服务器无感知计算中冷启动导致的响应延迟问题。本文观察到传统的固定阈值或简单的直方图策略难以适应波动剧烈的请求流量，导致资源浪费或预热不及时。因此，本文引入了双周期长短时记忆网络 LS-LSTM 来捕捉函数调用请求的时间依赖性与周期性模式。该模块能够精准预测下一个请求的到达时刻，并据此提前加载模型容器，从而在显著降低冷启动率的同时，将因预热而产生的闲置资源浪费控制在最低水平。

为了验证 SMore 系统的性能，本文在配备 8 块 NVIDIA RTX 3090 GPU 的真实测试平台以及大规模模拟环境中开展了实验，从不同角度对其性能进行了全面分析。

本文首先验证了预测模块的准确性，实验结果表明，在仅使用 20% 的采样空间下，随机森林模型能够达到约 0.3 的 RMSLE，优于其他线性模型。接着，本文将 SMore 与 TGS、ElasticFlow 等基线方法进行了对比。相比于独占执行模式，SMore 将 GPU 资源利用率提升了 34%，且将常规服务器式负载的性能退化控制在可接受范围内。与不感知退化的随机调度和 EDF 算法相比，SMore 显著提高了服务器无感知函数的截止时间满足率。最后，大规模模拟实验证明了系统具有良好的可扩展性，单次调度开销低于 1 毫秒；而在启用 MIG 的 A100 GPU 上的实验进一步证实了该架构在新型硬件上的通用性与有效性。

6.2　未来展望

本文在深度学习集群环境下，开展了常规服务器式与服务器无感知负载混合部署的高效资源管理框架的研究。本文结合深度学习任务的计算特征与服务器无感知计算的流量模式，设计了感知退化的调度系统 SMore 。然而，随着硬件架构的演进和大模型技术的爆发，相关领域的优化工作层出不穷，为后续的系统增强提供了新的思路。本文从以下两个方面对未来的研究展望进行了总结。

当前的工作主要侧重于提升通用型中低端 GPU 的利用率，且核心优化目标集中在 GPU 计算单元的占有上。虽然在现有实验条件下取得了显著成效，但为了进一步提升系统的可解释性与控制精度，未来需要对计算节点中的其他关键资源（如 CPU 算力、内存带宽、PCIe 带宽等）进行更细致的建模与分析。同时，随着高端 GPU（如 NVIDIA A100、H100）在数据中心的普及，其具备的多实例 GPU（MIG）等硬件隔离特性为混合部署提供了新的机遇。因此，如何将 SMore 扩展至高端 GPU 平台，并结合多维度的资源分析以实现更细粒度的资源隔离与调度，是未来的研究方向之一。

此外，现有工作主要基于传统的深度学习模型进行混合部署实验。随着大语言模型的日渐流行，其巨大的显存需求往往超过单张 GPU 的容量，通常需要依赖模型并行或流水线并行技术跨多卡部署。这种趋势使得 GPU 显存资源成为混合部署中最严峻的瓶颈，极大地限制了可部署服务器无感知函数的空间。因此，探索将大模型作为常规服务器式负载时的混合部署策略，设计更高效的模型切换机制与显存管理策略（如动态显存卸载或分页技术），以解决大模型背景下的资源争用问题，也是未来重要的研究方向。

参考文献

[1] YE Z, GAO W, HU Q, et al. Deep Learning Workload Scheduling in GPU Datacenters: A Survey [J]. ACM Computing Surveys, 2024, 56(6): 1-38.

[2] CHOWDHARY K. Natural Language Processing[M]//Fundamentals of Artificial Intelligence. Springer, 2020: 603-649.

[3] MALIK M, MALIK M K, MEHMOOD K, et al. Automatic Speech Recognition: A Survey[J]. Multimedia Tools and Applications, 2021, 80(6): 9411-9457.

[4] KO H, LEE S, PARK Y, et al. A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields[J]. Electronics, 2022, 11(1): 141.

[5] LI Y. Research and Application of Deep Learning in Image Recognition[C]//Proceedings of the IEEE International Conference on Power, Electronics and Computer Applications. 2022: 994-999.

[6] SUN C, MYERS A, VONDRICK C, et al. Videobert: A Joint Model for Video and Language Representation Learning[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 7464-7473.

[7] WENG Q, XIAO W, YU Y, et al. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters[C]//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation. 2022: 945-960.

[8] JEON M, VENKATARAMAN S, PHANISHAYEE A, et al. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads[C]//Proceedings of the USENIX Annual Technical Conference. 2019: 947-960.

[9] GAO Y, HE Y, LI X, et al. An Empirical Study on Low GPU Utilization of Deep Learning Jobs[C]//Proceedings of the IEEE/ACM International Conference on Software Engineering. 2024: 1-13.

[10] XIAO W, REN S, LI Y, et al. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. 2020: 533-548.

[11] ZHANG Y, GOIRI Í, CHAUDHRY G I, et al. Faster and Cheaper Serverless Computing on Harvested Resources[C]//Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles. 2021: 724-739.

[12] ZHAO H, HAN Z, YANG Z, et al. HiveD: Sharing a GPU Cluster for Deep Learning with Guarantees[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. 2020: 515-532.

[13] LUO L, WEST P, KRISHNAMURTHY A, et al. PLink: Discovering and Exploiting Locality for Accelerated Distributed Training on the public Cloud[C]//Proceedings of Machine Learning and Systems: vol. 2. 2020: 82-97.

[14] JONAS E, SCHLEIER-SMITH J, SREEKANTI V, et al. Cloud Programming Simplified: A Berkeley View on Serverless Computing[J]. arXiv preprint arXiv:1902.03383, 2019.

[15] LI Y, LIN Y, WANG Y, et al. Serverless Computing: State-of-the-Art, Challenges and Opportunities[J]. IEEE Transactions on Services Computing, 2022, 16(2): 1522-1539.

[16] 杨光, 刘杰, 曲慕子, 等. 服务器无感知计算系统性能优化技术研究综述[J]. 软件学报, 2025, 36(01): 47-78.

[17] SADEGHIAN G, ELSAKHAWY M, SHAHRAD M, et al. UnFaaSener: Latency and Cost Aware Offloading of Functions from Serverless Platforms[C]//Proceedings of the USENIX Annual Technical Conference. 2023: 879-896.

[18] YU H, FONTENOT C, WANG H, et al. Libra: Harvesting Idle Resources Safely and Timely in Serverless Clusters[C]//Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing. 2023: 181-194.

[19] FU T, YANG Z, YE Z, et al. A Survey on the Scheduling of DL and LLM Training Jobs in GPU Clusters[J]. Chinese Journal of Electronics, 2025, 34(3): 881-905.

[20] HU Q, SUN P, YAN S, et al. Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021: 1-15.

[21] GU J, CHOWDHURY M, SHIN K G, et al. Tiresias: A GPU Cluster Manager for Distributed Deep Learning[C]//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation: vol. 19. 2019: 485-500.

[22] WANG M, MENG C, LONG G, et al. Characterizing Deep Learning Training Workloads on Alibaba-PAI[C]//Proceedings of the IEEE International Symposium on Workload Characterization. 2019: 189-202.

[23] SERGEEV A, DEL BALSO M. Horovod: Fast and Easy Distributed Deep Learning in TensorFlow [J]. arXiv preprint arXiv:1802.05799, 2018.

[24] GU D, ZHAO Y, ZHONG Y, et al. ElasticFlow: An Elastic Serverless Training Platform for Distributed Deep Learning[C]//Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems: vol. 2. 2023: 266-280.

[25] NARAYANAN D, HARLAP A, PHANISHAYEE A, et al. PipeDream: Generalized Pipeline Parallelism for DNN Training[C]//Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles. 2019: 1-15.

[26] MURRAY D G, SIMSA J, KLIMOVIC A, et al. tf.data: A Machine Learning Data Processing Framework[J]. Proceedings of the VLDB Endowment, 2021, 14(12): 2945-2958.

[27] PENG Y, ZHU Y, CHEN Y, et al. A Generic Communication Scheduler for Distributed DNN Training Acceleration[C]//Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles. 2019: 16-29.

[28] SAPIO A, CANINI M, HO C Y, et al. Scaling Distributed Machine Learning with In-Network Aggregation[C]//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation. 2021: 785-808.

[29] JIN Y, WANG H, ZHONG R, et al. Graph-Centric Performance Analysis for Large-Scale Parallel Applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(7): 1221-1238.

[30] XIAO W, BHARDWAJ R, RAMJEE R, et al. Gandiva: Introspective Cluster Scheduling for Deep Learning[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. 2018: 595-610.

[31] NARAYANAN D, SANTHANAM K, KAZHAMIAKA F, et al. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. 2020: 481-498.

[32] QIAO A, CHOE S K, SUBRAMANYA S J, et al. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. 2021: 1-18.

[33] CHAUDHARY S, RAMJEE R, SIVATHANU M, et al. Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning[C]//Proceedings of the European Conference on Computer Systems. 2020: 1-16.

[34] MAHAJAN K, BALASUBRAMANIAN A, SINGHVI A, et al. Themis: Fair and Efficient GPU Cluster Scheduling[C]//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation. 2020: 289-304.

[35] SU S, ZHANG Y, WEN Y, et al. Sia: Heterogeneity-Aware, Goodput-Optimized ML-Cluster Scheduling[C]//Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles. 2023: 1-16.

[36] YU P, CHOWDHURY M. Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications[C]//Proceedings of Machine Learning and Systems. 2019: 1-14.

[37] ZHU J, YANG H, ZHANG W. MGM: A Fine-grained GPU-Sharing Framework for Large-Scale LLM Serving[C]//Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems: vol. 2. 2023: 689-704.

[38] STRATI F, MA X, KLIMOVIC A. Orion: Interference-Aware, Fine-Grained GPU Sharing for ML Applications[C]//Proceedings of the European Conference on Computer Systems. 2024: 1075-1092.

[39] ZHANG C, YU M, WANG W, et al. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving[C]//Proceedings of the USENIX Annual Technical Conference. 2019: 1049-1062.

[40] SHILLAKER S, PIETZUCH P. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing[C]//Proceedings of the USENIX Annual Technical Conference. 2020: 419-433.

[41] SURESH A, GANDHI A. ServerMore: Opportunistic Execution of Serverless Functions in the

Cloud[C]//Proceedings of the ACM Symposium on Cloud Computing. 2021: 570-584.

[42] KANNAN R, SUBRAMANIAN L, RAJU A, et al. GrandSLAM: Guaranteeing SLAs for Microservices Execution in Serverless Computing[C]//Proceedings of the European Conference on Computer Systems. 2021: 358-374.

[43] JIA Z, WITCHEL E. Nightcore: Efficient and Scalable Serverless Computing for Latency-Sensitive, Interactive Microservices[C]//Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021: 152-166.

[44] KULKARNI S, LIU G, RAMAKRISHNAN K, et al. Aquatope: QoS-and-Cost-Aware Resource Management for Multi-stage Serverless Workflows[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. 2023: 535-552.

[45] YANG Y, ZHAO L, LI Y, et al. INFless: A Native Serverless System for Low-Latency, High-Throughput Inference[C]//Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 768-781.

[46] YU M, WANG A, CHEN D, et al. FaaSwap: SLO-Aware, GPU-Efficient Serverless Inference via Model Swapping[J]. arXiv preprint arXiv:2306.03622, 2023.

[47] PAN S, ZHAO H, CAI Z, et al. Sustainable Serverless Computing with Cold-start Optimization and Automatic Workflow Resource Scheduling[J]. IEEE Transactions on Sustainable Computing, 2023, 9(3): 329-340.

[48] ZHAO H, PAN S, CAI Z, et al. faaShark: An End-to-End Network Traffic Analysis System Atop Serverless Computing[J]. IEEE Transactions on Network Science and Engineering, 2023, 11(3): 2473-2484.

[49] SHAHRAD M, FONSECA R, GOIRI I, et al. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider[C]//Proceedings of the USENIX Annual Technical Conference. 2020: 205-218.

[50] MOHANTY S, LAGAR-CAVILLA A, BHARDWAJ K. IceBreaker: Warming Serverless Functions better with Heterogeneity-aware Policies[C]//Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 832-846.

[51] DU D, YU T, XIA Y, et al. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting[C]//Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 467-481.

[52] CADDEN J, UNGER T, YIGITBASI E, et al. SEUSS: Skipping Redundant Paths to Speed Up Serverless Functions[C]//Proceedings of the European Conference on Computer Systems. 2020: 1-15.

[53] OLTKAR M, DOHARE O. SOCK: Rapid Container-Based Systems for Serverless Workloads[C]//Proceedings of the USENIX Annual Technical Conference. 2021: 487-500.

[54] BAI Z, ZHANG Z, ZHU Y, et al. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications[C]//Proceedings of the USENIX Symposium on Operating Systems Design and

Implementation. 2020: 499-514.

[55] EISMANN S, SCHEUNER J, VAN EYK E, et al. A Review of Serverless Use Cases and Their Characteristics[J]. arXiv preprint arXiv:2008.11110, 2020.

[56] 温金凤, 陈震鹏, 柳熠, 等. 服务器无感知平台性能度量研究[J]. 软件学报, 2025, 36(05): 1974-2005.

[57] CAI Z, CHEN Z, MA R, et al. SMSS: Stateful Model Serving in Metaverse with Serverless Computing and GPU Sharing[J]. IEEE Journal on Selected Areas in Communications, 2023, 42(3): 799-811.

[58] SHI H, ZHENG W, LIU Z, et al. Automatic Pipeline Parallelism: A Parallel Inference Framework for Deep Learning Applications in 6G Mobile Communication Systems[J]. IEEE Journal on Selected Areas in Communications, 2023, 41(7): 2041-2056.

[59] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition[C]//Proceedings of the International Conference on Learning Representations. 2015: 1-14.

[60] KRIZHEVSKY A, HINTON G. Learning Multiple Layers of Features from Tiny Images[R]. University of Toronto, 2009.

[61] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[62] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[63] DENG J, DONG W, SOCHER R, et al. ImageNet: A Large-Scale Hierarchical Image Database [C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2009: 248-255.

[64] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding[C]//Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: vol. 1. 2019: 4171-4186.

[65] LI Y, ZHANG Y, ZHAO Z, et al. CSL: A Large-scale Chinese Scientific Literature Dataset[C]// Proceedings of the International Conference on Computational Linguistics. 2022: 3917-3923.

[66] LIU Y, OTT M, GOYAL N, et al. Roberta: A Robustly Optimized Bert Pretraining Approach[J]. arXiv preprint arXiv:1907.11692, 2019.

[67] RAJPURKAR P, JIA R, LIANG P. Know What You Don't Know: Unanswerable Questions for SQuAD[C]//Proceedings of the Annual Meeting of the Association for Computational Linguistics: vol. 2. 2018: 784-789.

[68] GUO H, TANG R, YE Y, et al. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction[C]//Proceedings of the International Joint Conference on Artificial Intelligence. 2017:

1725-1731.

[69] XIE Z, CHEN J, FENG Y, et al. End to End Multi-Task Learning with Attention for Multi-Objective Fault Diagnosis under Small Sample[J]. Journal of Manufacturing Systems, 2022, 62: 301-316.

[70] SILBERMAN N, HOIEM D, KOHLI P, et al. Indoor Segmentation and Support Inference from RGBD Images[C]//Proceedings of the European Conference on Computer Vision. 2012: 746-760.

[71] PASZKE A, GROSS S, MASSA F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library[C]//Advances in Neural Information Processing Systems: vol. 32. 2019: 8024-8035.

[72] ABADI M, AGARWAL A, BARHAM P, et al. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems[J]. arXiv preprint arXiv:1603.04467, 2016.

[73] WU B, ZHANG Z, BAI Z, et al. Transparent GPU Sharing in Container Clouds for Deep Learning Workloads[C]//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation. 2023: 69-85.

[74] LI B, PATEL T, SAMSI S, et al. Miso: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters[C]//Proceedings of the ACM Symposium on Cloud Computing. 2022: 173-189.

致 谢

在上海交通大学计算机学院，我度过了充实而难忘的两年半硕士学习生活。在这里，我完成了系统的专业课程学习与科研训练，不断提升自身的学术素养与工程实践能力。

首先，衷心感谢我的导师马汝辉教授。在整个研究生阶段的学习和科研过程中，导师不仅在学术上给予了我细致入微的指导，更在做人和做事方面给予了我深刻的启发。每当我在科研道路上陷入迷茫和困惑时，导师总能以严谨的治学态度、深厚的专业知识和敏锐的洞察力，为我指明方向、理清思路。导师对细节的严格要求、对创新的不断追求以及对科研工作的执着精神，将一直激励我在今后的工作和学习中不断进取。感谢导师在课题选择、论文撰写和职业规划等方面的耐心指导和无私帮助，祝愿马汝辉老师工作顺利、科研卓著、生活美满。

感谢实验室的各位老师和同学们，尤其要感谢蔡子诺学长。在学术、科研和生活各个方面，实验室的各位都给予了我许多宝贵的建议和经验分享。实验室提供的良好的科研氛围、完善的科研条件以及融洽的团队环境，为我的成长提供了坚实的平台和广阔的空间。在与实验室同门们的交流与合作中，我学会了如何更高效地开展团队协作、如何更清晰地表达自己的想法，也感受到了团队互帮互助的温暖氛围。这段在实验室的学习与生活经历，将成为我未来人生道路中弥足珍贵的回忆。祝愿师兄以及实验室的同门们科研顺利、前程似锦。

特别感谢我的女朋友，在我整个硕士学习期间对我的理解、陪伴与支持。在科研压力较大、进展不顺利的时候，是她不断地鼓励我、安慰我，让我重新树立信心、调整心态，帮助我以更加平和和坚定的心态面对挑战。她的体贴与包容，让我在紧张的科研生活之外拥有一份温暖的牵挂与动力。我们一起在校园中留下的点滴回忆，将会成为我人生中最美好的记忆之一。也希望未来的日子里，我们能够继续携手同行，共同迎接生活中的每一个阶段。

同时，衷心感谢我的父母对我一直以来无条件的支持和理解。感谢你们在我求学道路上的默默付出和辛勤付出，是你们给了我追求梦想的勇气和底气。无论我身在何处，你们始终是我最坚实的后盾。在我面对困难和挫折时，是你们用朴实而坚定的关怀支撑着我、鼓励着我，使我能够义无反顾地走到今天。你们的教导与支持将是我人生路上最宝贵的财富。我也会努力工作与生活，以更加成熟和负责的态度来回馈你们

的养育之恩。

最后，感谢在这段时光里陪伴我的朋友们。你们让我的研究生生活不再只是实验室与论文的循环。无论是课间的闲聊、深夜的畅谈，还是在校园里散步、聚餐、运动的时刻，都为这段紧张的学习时光增添了许多轻松与快乐。感谢你们在我低落时的安慰与鼓励，在我取得小小进步时的真诚祝贺。希望我们之间的友谊能够在毕业之后继续延续，在今后的工作与生活中依然保持联系、互相支持。

两年半的硕士生涯即将告一段落，这段经历不仅让我收获了知识与能力，更让我学会了坚持与担当、合作与感恩。再次感谢所有在此期间给予我关心、帮助与支持的人，是你们共同构成了我这段珍贵的研究生时光。

学术论文和科研成果目录

学术论文

[1] Liu J, Cai Z, Liu Y, et al. SMore: Enhancing GPU Utilization in Deep Learning Clusters by Serverless-Based Co-Location Scheduling[J]. IEEE Transactions on Parallel and Distributed Systems, 2025, 36(5): 903-917.