多智能体协同软件开发的形式化建模与实验验证

严亚伟,汪东升

(清华大学 计算机科学与技术系,北京 100084)

摘 要: 针对多智能体协同软件开发中智能体能力未知、异构且带噪声,导致依赖任务难以高质量调度的问题,从软件工程视角构建包含任务分解、能力映射与反馈优化的五元组形式化模型$S=<T, A, F, M, \Pi>$,并提出以贝叶斯能力学习为核心驱动、UCB探索与匹配阈值为可选增强模块的迭代调度算法BayesSchedule。实验基于Pegasus真实工作流基准与Bernoulli-Gaussian失败模型,设置覆盖规模、噪声与结构条件的5组仿真实验,并与UCB1、Thompson Sampling、QL-HEFT 3种在线学习基线方法进行比较。实验结果表明,BayesSchedule在100次迭代后能力估计误差降至0.01,总调度成本与理论最优基准Oracle基本持平,任务总工期(makespan)较HEFT降低10.0%,总成本较3种在线学习基线分别降低30.2%、28.3%与67.9%。研究结果验证了反馈机制F与调度策略Π在能力不确定场景下的有效性,并为多智能体协同软件开发从"工具辅助"走向"能力编排"提供了形式化依据。

关键词: 多智能体系统;软件工程;协同开发;形式化建模;贝叶斯调度;仿真实验

DOI: 10.11907/rjdk.261340 扫描二维码阅读全文:

中图分类号: TP311.5 **文献标识码:** A **文章编号:** 1672-7800(2026)007-0001-11

Formal Modeling and Experimental Evaluation for Multi-Agent Collaborative Software Development

YAN Yawei, WANG Dongsheng

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: Addressing the challenge of high-quality scheduling of dependent tasks in multi-agent collaborative software development, which is hindered by unknown, heterogeneous, and noisy agent capabilities, we construct a five-tuple formalized model $S=<T, A, F, M, \Pi>$ from a software engineering perspective, encompassing task decomposition, capability mapping, and feedback optimization. We propose an iterative scheduling algorithm, BayesSchedule, driven by Bayesian capability learning as its core and with UCB exploration and matching threshold as an optional enhancement module. The experiments are based on the Pegasus real-world workflow benchmark and the Bernoulli-Gaussian failure model. Five sets of simulation experiments are set up, covering scale, noise, and structural conditions, and are compared with three online learning baseline methods: UCB1, Thompson Sampling, and QL-HEFT. The experimental results show that after 100 iterations, the capability estimation error of BayesSchedule drops to 0.01. The total scheduling cost is basically on par with the theoretical optimal benchmark Oracle. The total task duration (makespan) is reduced by 10.0% compared to HEFT, and the total cost is reduced by 30.2%, 28.3%, and 67.9% compared to the three online learning baselines, respectively. The research results validate the effectiveness of the feedback mechanism F and scheduling strategy Π in capability uncertainty scenarios, and provide a formalized basis for multi-agent collaborative software development to evolve from "tool assistance" to "capability orchestration".

Key Words: multi-agent system; software engineering; collaborative development; formal modeling; Bayesian scheduling; simulation experiment

0 引言

随着云计算、微服务架构与大规模分布式系统的持续演进,现代软件系统的规模、耦合度与交付不确定性同步上升。传统以人工分工、会议协同和经验判断为主的软件开发模式在大型项目中容易受到沟通链条长、任务依赖复杂和人员能力差异等因素影响。Standish Group的CHAOS

收稿日期: 2026-06-01

作者简介: 严亚伟(1990-),男,清华大学计算机科学与技术系博士研究生,研究方向为高性能计算与智算融合;汪东升(1966-),男,清华大学计算机科学与技术系教授、博士生导师,研究方向为计算机体系结构和数据安全。本文通讯作者:汪东升。

报告显示，仅约31%的软件项目能够成功交付[1]。针对5 400余个大型IT项目的研究进一步指出，大型项目平均超出预算45%、交付价值低于预期56%[2]。因此，如何在复杂任务依赖下更可靠地组织开发资源、降低交付风险，已成为软件工程亟需回应的重要问题。

近年来，以大语言模型（Large Language Model，LLM）为代表的人工智能（Artificial Intelligence，AI）技术使智能体具备任务理解、代码生成与协同决策能力，多智能体协同开发由此成为缓解上述问题的新路径[3-6]。但这一路径并不等同于简单增加智能体人手，不同智能体在代码生成、测试、架构设计等任务上的能力未知且异构，执行结果可能出现离散失败[7]。此外，软件开发任务还存在严格的先后依赖。能否在能力不确定与任务依赖并存的条件下实现可学习、可解释的任务调度，直接关系到多智能体协同开发能否从工具组合走向工程方法。

1 相关研究

1.1 产业实践现状

从产业实践看，AI辅助编程已成为开发流程的重要组成部分。多项实证研究表明，GitHub Copilot等工具能显著提升局部编码效率，但当任务上升到项目级时，需求分解、接口协同、测试反馈和进度约束会引入新的协调成本[6,8-9]。也就是说，单智能体的局部提效并不能自然推出团队级的全局最优，多智能体协同机制仍需专门设计。

围绕项目级协作，产业界已经形成若干代表性框架。例如，MetaGPT模拟软件公司的组织结构，将产品经理、架构师、工程师等角色映射为多个协作智能体，实现了从需求到代码的端到端生成[10]。AutoGPT提供模块化自主智能体平台，支持用户自定义智能体行为链[11]。TheBotCompany引入自组织智能体团队，管理者智能体可根据项目需求动态组建和调整团队[12]。国际数据公司IDC预测，到2030年，70%的开发者将与自主AI智能体合作[13]。Russinovich等[14]亦指出，AI智能体正在重新定义软件工程职业本身，开发者的角色将从直接编写代码转向指导、审查和编排智能体工作流。这些实践共同表明，多智能体协同开发已从概念验证迈向工程落地。然而，其核心调度机制、能力学习过程和评价标准仍缺乏系统化理论支撑，亟需进一步研究。

1.2 学术研究进展

软件工程领域对多智能体系统的研究可追溯至面向智能体的软件工程这一领域[15]。该方向将智能体作为基本计算单元参与到软件开发过程中，但受限于当时智能体的认知能力，其主要停留在理论建模层面。随着LLM赋予智能体自然语言理解、代码生成与上下文推理等能力，多智能体系统在实际软件开发中的应用潜力被显著释放，尤其是基于LLM的智能体能够理解自然语言任务，结合上下文进行自适应推理，并通过多轮对话完成协同决策，使自主完成复杂软件开发流程成为可能。然而，从“能协作”到“协作得好”仍有距离。智能体能力的异构性如何量化？执行反馈能否持续改善调度质量？这些问题需要在形式化模型和系统化实验中回答。

现有研究可归纳为三类。第一类关注多智能体软件工程框架。例如，He等[3]系统综述了基于LLM的多智能体系统在软件工程中的应用图景，覆盖从需求分析、架构设计到代码实现与测试的全生命周期。Tang等[16]讨论了LLM驱动智能体系统在软件工程落地中的机遇与挑战。Zhang[17]综述了多智能体工作流自动化的方法与分类。Benkovich等[18]提出面向团队式自主开发的多智能体系统Agyn。该类研究证明了多角色协作的可行性，但多停留在系统结构和功能流程层面，较少将智能体能力差异、执行不确定性和任务依赖关系放入统一模型。第二类关注协同机制与组织形态。例如，Jennings[15]的研究奠定了面向智能体软件工程的理论基础。Jwo等[19]系统回顾了基于智能体的敏捷开发方法。Russinovich等[14]讨论了智能体对软件工程职业形态的重塑作用。该类研究拓展了人机协同的组织视角，但主要回答了“人和智能体如何协作”的宏观问题，尚未深入到“任务如何分配、能力如何学习”的算法层面。第三类关注异构任务调度与在线学习算法。例如，Topcuoglu等[20]提出的异构最早完成时间算法（Heterogeneous Earliest Finish Time，HEFT）为异构有向无环图（Directed Acyclic Graph，DAG）任务调度的经典算法，在执行时间已知的前提下具有优异的性能—复杂度权衡。Tong等[21]将Q-learning与HEFT优先级结合，提出面向云环境的QL-HEFT调度方案。在不确定性决策方面，基于置信上界（Upper Confidence Bound，UCB）思想的UCB1[22]、Thompson Sampling[23]等多臂老虎机算法以及高斯过程贝叶斯优化算法[24]为探索—利用权衡提供了成熟工具。然而，经典DAG调度通常假设执行时间已知，缺乏从反馈中学习的能力；多臂老虎机与强化学习方法虽可进行在线学习，却普遍弱化任务依赖结构与多维能力语义，难以直接迁移到软件开发的任务级调度场景中。综上所述，框架类工作重在系统构建，组织类工作重在协同形态，调度类工作重在算法机制，但三者尚未在同一问题上闭合。在智能体能力未知、异构且带噪声的条件下，如何对具有DAG依赖的软件开发任务进行可在线改进的形式化调度仍需要进一步探索。

多智能体调度的根本难点在于智能体能力的不确定性与异构性。以SWE-bench这一真实GitHub问题修复基准为例，不同模型与版本的成功率差异巨大，失败常以编译中断、运行时崩溃、上下文溢出等离散形式出现，而非围绕正确结果的平滑退化，这使经典调度方法在多智能体软件开发场景中面临失配[7]。以HEFT为代表的异构DAG调度算法假设每个任务在每个处理单元上的执行时间已知且确定，据此一次性计算分配方案。然而，在多智能体

协同开发中，智能体真实能力向量事先未知，只能通过反复执行任务、观测带噪声且可能失败的结果逐步推断。静态调度算法无法利用这种跨项目积累的反馈信息，每次调度均重复依赖同一能力先验，因而难以随经验改善。换言之，多智能体调度本质上是能力部分可观测的在线优化问题，需要在探索（获取更准确的能力估计）与利用（基于当前估计做出较优调度）之间持续权衡。贝叶斯学习适合处理这一问题。首先，贝叶斯共轭后验能将带噪声观测递推转化为能力估计的更新，每次更新代价较低，适合在线调度；其次，后验方差可以自然刻画估计不确定性，并与UCB式探索机制结合，在能力未知初期鼓励必要探索，在后验收敛后逐步转向利用；再次，相较于无模型的多臂老虎机或强化学习方法，贝叶斯框架便于注入领域先验，在样本有限的工程迭代中更容易获得稳定估计。

基于上述分析，本文从软件工程视角构建多智能体协同开发的形式化模型，提出基于贝叶斯能力学习的迭代调度算法BayesSchedule，并通过5组系统化仿真实验验证其有效性。本文的主要贡献在于：①提出五元组模型 $S=<T, A, F, M, \Pi>$，将任务分解、能力映射、反馈学习与调度策略纳入统一框架；②构建面向实验设计的智能体与任务分类体系；③在不同规模、噪声和结构条件下验证反馈机制 F 与调度策略 Π 对调度质量的改善作用。

2　多智能体协同开发的形式化建模

2.1　系统模型定义

将多智能体协同开发系统定义为一个五元组：$S=<T, A, F, M, \Pi>$，其中任务集合 $T=\{t_1, t_2, \cdots, t_n\}$ 表示待完成的软件开发任务。每个任务 t_i 具有属性向量 $<d_i, r_i, c_i>$，分别表示任务难度、所需资源类型和预估执行成本。任务之间存在偏序依赖关系 $D \subseteq T \times T$，可建模为DAG，表示任务执行的先后约束。在实际软件工程中，这种偏序关系自然产生于模块间的接口依赖、数据流传递和集成测试的前置条件等场景中。智能体集合 $A=\{a_1, a_2, \cdots, a_m\}$ 表示参与协同开发的智能体。每个智能体 a_j 具有能力向量 $c_j=<c_{j1}, c_{j2}, \cdots, c_{jk}>$，所属能力维度数为 k。考虑到大语言模型智能体的实际表现并非平滑的对称波动，而是兼具连续抖动与离散失败两种模式，本文采用Bernoulli-Gaussian混合模型刻画其执行表现，即以概率 p_fail 发生灾难性失败（对应编译错误、运行时崩溃、上下文窗口溢出或超时等离散故障），此时任务产出无效并触发超时惩罚；以概率 $(1-p\_fail)$ 正常执行，此时观测能力为真实能力的高斯扰动 $c_j^{obs}=c_j+\varepsilon, \varepsilon \sim N(0, \sigma^2 I_k)$。其中失败概率 p_fail 随任务难度与智能体能力的差距单调上升，刻画了“任务越超出智能体能力、越容易彻底失败”的现实规律。

采用Bernoulli-Gaussian混合而非纯高斯噪声是基于近年来对LLM代码生成失败模式的实证研究。Tambon等[25]通过分析数百例LLM生成的代码缺陷发现，模型失败往往以离散、不可被编译器平滑捕获的形式出现，而非围绕正确解的对称扰动。在SWE-bench等端到端基准上，模型的整体失败率亦呈现明显的离散、长尾特征。因此，本文将失败概率 p_fail 参数化为任务难度与智能体能力差距的函数（基础失败率取0.05，并随能力缺口线性上升），失败时赋予执行成本以超时惩罚。这一设计使仿真更贴近真实多智能体软件开发中的故障特性，也对调度算法的鲁棒性提出了更高要求。

反馈机制 $F: H \times O \to \Delta$ 将历史执行记录和当前输出映射为改进策略，用于持续修正能力估计并优化后续调度。其中，H 为历史执行记录空间，每条记录包含（任务、被分配智能体、观测到的执行表现）三元组；O 为当前项目的输出观测空间，包含各任务的实际执行时间、成败状态与deadline满足情况；Δ 为系统状态的更新空间，在本文中具体为各智能体能力估计值 $\hat{c}_a$ 的修正量。后续将 F 具体实例化为贝叶斯能力学习机制，即历史记录与新观测通过共轭后验更新转化为能力估计的后验分布修正。

任务映射函数 $M: T \to 2^A$ 将每个任务分配给一个或多个智能体，满足能力匹配约束 $\forall t \in T, a \in \mathrm{M}(t): \mathrm{match}(d_t, \hat{c}_a) \geqslant \theta$，其中 θ 为匹配阈值，$\hat{c}_a$ 为智能体能力的当前估计值。匹配函数 $\mathrm{match}(\cdot,\cdot)$ 以任务难度向量与智能体能力向量的点积来衡量能力适配程度，反映了多维能力空间中任务需求与智能体特长的对齐关系。在工程实践中，能力匹配的质量直接决定了任务执行的效率与质量：当智能体的能力与任务需求高度匹配时，执行成本显著降低；而当匹配度不足时，不仅执行成本增加，而且可能导致输出质量下降和deadline违反。因此，能力估计的准确性是调度质量的基础，这正是引入贝叶斯能力学习机制的根本动机。调度策略 $\Pi: T \times A \times \mathrm{Time} \to \{0, 1\}$ 决定任务执行的时序安排，需满足DAG依赖约束。上述5个组件共同构成统一建模框架，并在后续算法与实验中得到验证。

2.2　优化目标与复杂度边界

多智能体协同开发的优化目标为最小化加权综合成本，表示为 $\min \Sigma_{i=1}^{n}(\alpha C_i + \beta L_i - \gamma Q_i)$，其中 C_i 为任务 t_i 的执行成本，L_i 为延迟惩罚，Q_i 为质量收益；3个权重参数 α、β、γ 分别反映了成本效率、时间约束和质量要求在不同应用场景中的优先级差异。例如，在紧急修复场景中，β 权重较高；在长期迭代开发中，α 权重占主导。该优化目标具有多目标特性，不同应用场景可通过调整权重参数来匹配实际需求，无需改变算法本身的结构，这种灵活性是形式化模型相较于单一指标优化的重要优势。

3　面向实验设计的分类体系

3.1　按智能体类型（A维）分类

前文的五元组模型给出了系统的静态结构，本节进一

步沿 A（智能体）与 T（任务）两个维度对模型进行分类细化，这直接决定了后续能力空间的维度设计与工作流类型的选取。根据能力向量 c_j 的主成分差异，智能体可分为代码生成型（如 GitHub Copilot[6,9]）、测试与验证型（如 Amazon Security Agent[16]）、架构与设计型（如 MetaGPT 中的架构师智能体[10]）、调度与编排型（如 TheBotCompany 中的管理者智能体[12]）4 类。每个智能体的能力向量 $c_j \in \mathbb{R}^4$ 涵盖 coding、testing、design、review 4 个维度，对应上述分类。这一分类方式不仅具有理论完备性，而且与实际软件开发团队中常见的角色划分（开发者、测试工程师、架构师、项目经理）形成自然的对应关系，有助于将形式化模型的研究结论迁移至工程实践中。

3.2 按任务特征（T 维）分类

任务集合 T 的依赖关系 D 和粒度分布决定了系统的应用领域，例如智能编程助手任务粒度细、依赖简单；自主软件开发任务粒度粗、依赖复杂，需端到端自主完成[18]；智能运维任务具有持续性和实时性，需 7×24 h 自主响应[26]。不同应用领域对调度策略 Π 的要求存在本质差异：编程助手场景侧重单次任务的响应速度，自主开发场景关注全局调度的最优性，而智能运维场景则强调实时性和鲁棒性。值得注意的是，上述 3 类应用在任务图结构上也存在显著差异：编程助手的任务图通常呈链式结构（顺序依赖），自主开发的任务图呈复杂 DAG 结构（多条并行路径与汇聚点），智能运维的任务图则可能动态变化（运行时生成新的依赖关系）。

4 基于贝叶斯能力学习的调度算法

4.1 问题设定

考虑包含 m 个异构智能体的协同开发系统，需要串行执行 N 次软件工程项目，每次项目包含 n 个具有 DAG 依赖关系的任务。智能体 a_j 的真实能力向量 $c_j \in \mathbb{R}^k$ 未知，调度器仅能通过观测任务执行结果逐步推断。执行每次任务后，调度器以概率（$1-p\_fail$）获得带噪声的能力观测 $c_j^{obs} = c_j + \varepsilon, \varepsilon \sim N(0, \sigma^2 I_k)$，或以概率 p_fail 仅观测到一次失败事件，目标为在 N 次迭代中最小化总调度成本的累积值。该问题本质上是一个部分可观测的在线优化问题，需要在探索（获取更准确的能力估计）与利用（基于当前估计做出最优调度）之间取得平衡。

4.2 贝叶斯能力估计（反馈机制 F 的实例化）

采用高斯共轭先验模型，对每个智能体的各能力维度维护后验分布 $N(\mu_j^d, \sigma_j^{d2})$，初始先验设为 $N(0.5, 0.25^2)$。每次观测后按贝叶斯公式更新均值与方差，从而逐步逼近真实能力。

4.3 UCB 增强的 HEFT 调度（调度策略 Π 的实例化）

在任务分配阶段，算法采用 HEFT 框架，但将固定执行时间估计替换为 UCB 增强值 $\hat{c}_j^{ucb} = \mu_j + \lambda \cdot \sqrt{\sigma_j^2}$，其中 λ 为探索系数。该策略在能力不确定时鼓励探索，在后验收敛后转向利用。需要说明的是，UCB 探索与匹配阈值 θ 在所提算法中均定位为可选增强模块，算法的核心驱动力为贝叶斯能力学习与 HEFT 全局调度的结合，二者在 λ=0（即退化为纯后验均值）时依然完整可用。UCB 项主要面向弱依赖结构或智能体能力差异极大、需要额外探索的场景。

UCB 增强的 HEFT 调度步骤为：①基于 UCB 能力估算每个任务在每个智能体上的预计执行时间；②按照 HEFT 的 upward-rank 优先级对任务进行排序；③从高优先级任务开始，将每个任务分配给使其最早完成时间（Earliest Finish Time，EFT）最小的智能体；④项目执行完成后，收集观测数据并更新贝叶斯后验。步骤①-③构成了调度策略 Π 的单次执行，步骤④实现了反馈机制 F 的在线更新。4 个步骤循环迭代，构成了完整的在线学习调度框架。

4.4 算法复杂度分析

单次项目调度的时间复杂度为 $O(n \cdot m + n \cdot \log n)$，其中 $n \cdot m$ 为计算所有任务—智能体执行时间矩阵的开销，$n \cdot \log n$ 为 HEFT 排序的代价，贝叶斯更新的复杂度为 $O(m \cdot k)$，因此 N 次迭代的总复杂度为 $O(N \cdot (n \cdot m + m \cdot k))$，在实验规模（$n \leqslant 200$，$m \leqslant 20$，$k$=4，$N$=100）下具有良好效率。相较于基于模拟退火或遗传算法的全局优化方法，该算法每次迭代的计算成本仅为线性级别，适合在线实时调度场景。值得指出的是，BayesSchedule 算法的核心优势不仅在于单次调度的质量，更在于其跨迭代的自适应学习能力，这是 Random、Greedy 和 HEFT 等静态调度算法所不具备的特性。通过在同一框架内统一“学习”与“调度”两个过程，BayesSchedule 从数据驱动的角度实现了对调度策略的持续优化。

5 实验验证

5.1 实验设置

仿真环境基于 Python 实现，包含工作流加载器、异构智能体模型、Bernoulli-Gaussian 失败模型和离散事件调度引擎。为克服以往随机 DAG 仿真脱离真实场景的不足，本研究不再使用随机生成的任务图，而是采用 Pegasus 科学工作流标准基准作为任务依赖结构的来源。默认设置为 n=100、m=10、σ=0.15、N=100，并在每组参数下重复 20 次取均值。为支持工作的可复现性，全部实验代码（含 8 种调度算法实现、Pegasus 工作流加载器与实验脚本）已开源发布于 https://github.com/yanyw/schedule-multi-agent-dev，其中 Pegasus 工作流实例数据来源于 WfCommons 开放仓库[27]。如表 1 所示，仿真实验的默认参数覆盖任务规模、智能体数量、能力噪声、迭代次数及调度约束等关键因素，为后续不同场景的对比实验提供统一基准。

5.1.1 仿真场景设计

实验模拟的核心场景为多智能体软件开发中的任务

Table 1　Default parameter settings for simulation experiments

表 1　实验默认参数设置

参数	符号	默认值	说明
任务数	n	100	DAG 中的节点数
智能体数	m	10	参与调度的智能体数量
能力噪声	σ	0.15	观测噪声标准差
迭代次数	N	100	串行执行工程数
能力维度	k	4	coding/testing/design/review
UCB 系数	λ	0.5	探索–利用平衡系数
匹配阈值	θ	0.3	任务–智能体最低匹配度
松弛系数	γ	1.5	deadline 松弛倍数
基础消耗	η	0.3	最低执行开销系数

调度。每次工程对应一个完整的软件开发项目，包含一组具有偏序依赖关系的任务，这些任务构成一个 DAG。调度器的职责是将每个任务分配给最合适的智能体，并确定执行顺序，使得总调度成本最小化。系统串行执行 N=100 次工程：在执行每次工程后，调度算法基于观测到的数据更新对智能体能力的估计，然后在下一次工程中使用更新后的估计进行任务分配。这一“执行—观测—更新—再执行”的迭代过程正是形式化模型中反馈机制 F 的具体表现。

5.1.2　数据生成机制

每个智能体的真实能力向量从 Uniform(0.2, 1.0)采样并保持不变；任务难度向量则依据 Pegasus 工作流中的节点类型、真实运行时和 I/O 规模等属性映射到 coding、testing、design、review 四维能力空间，用于模拟能力异构与真实任务负载。

异构 DAG 调度领域长期以 Pegasus 工作流库（经 WfCommons 项目整理与开放）作为标准评测基准，其工作流来自天文图像合成、地震模拟、引力波检测等真实应用，具有明确的任务类型语义与真实采集的运行时、I/O 等执行属性[27]。本文从中选取 5 类具有代表性且任务规模覆盖 50~200 个区间的工作流家族，分别为 Montage（天文图像镶嵌）、CyberShake（地震危险性分析）、Inspiral（引力波数据处理）、SIPHT（生物信息学 sRNA 搜索）与 Epigenomics（基因组测序）。每个工作流实例的节点对应一个真实任务，边对应真实的数据依赖关系。任务的难度向量依据其在数据集中的任务类型（如 Montage 的 mProject、mDiffFit、mBackground 等）、真实运行时与 I/O 字节量映射到 coding、testing、design、review 4 个能力维度。这一设计使实验任务图在拓扑结构（关键路径、并行宽度、汇聚模式）和任务异构性上均贴近真实软件工程项目。

需要指出的是，软件工程领域目前尚缺乏带有真实任务依赖结构与执行属性，且公开可复现的大规模任务图基准。在缺乏软件工程专用基准的情况下，本文采用异构 DAG 调度研究社区公认的 Pegasus 标准基准，主要基于两点考虑：①可复现性与客观性。使用公认基准而非自行随机生成的任务图可使实验结果被独立复现并进行横向比较；②结构同构性。本文关注的是调度算法在任务依赖拓扑上的表现，而 Pegasus 工作流所具有的关键路径长度、并行分支宽度、扇入扇出汇聚等拓扑特征，与真实软件项目中由模块接口依赖、数据流传递、集成测试前置条件所形成的任务图在结构上高度同构。在此基础上，本研究进一步将工作流的任务类型按其计算特征映射到软件工程的 coding/testing/design/review 四维能力空间。因此，本文实验应理解为在真实任务依赖结构上对调度机制有效性的验证，而非对特定软件产品交付收益的直接度量。

每个任务 t_i 的难度向量 $d_i \in \mathbb{R}^4$ 从 Uniform(0.3, 1.0)中采样，基础执行成本 $c_i^{base} = 2 \cdot \| d_i \|_2$ 与难度的 L2 范数成正比。任务的 deadline 基于其在 DAG 中的拓扑深度和基础执行成本推算，并乘以松弛系数（默认 1.5）以预留合理的时间裕度。执行成本的计算公式为 $C_i = c_i^{base} \cdot \max(0, \| d_i \| - \| c_a^{obs} \|) + c_i^{base} \cdot \eta$，其中 η=0.3，为基础消耗系数。该公式的含义为当智能体能力越接近或超过任务难度时，额外开销越小；即使能力完全匹配，也存在不可消除的基础执行开销。

5.2　对照算法与评价指标

为全面评估 BayesSchedule 的性能，设计 7 种对照算法，覆盖从随机基线到理论最优、从静态到在线学习的完整范围。表 2 为各算法的名称与核心思路。

Table 2　Baseline algorithms for comparison

表 2　对照算法

算法	说明
Random	在满足 DAG 依赖约束下随机分配智能体，作为性能下界
Greedy	每个就绪任务贪心选择当前估计能力最匹配的空闲智能体
HEFT	Heterogeneous Earliest Finish Time，经典异构 DAG 调度算法
UCB1	频率主义置信上界，在线学习无先验[22]
Thompson	Beta–Bernoulli 后验采样在线学习[23]
QL–HEFT	Q–learning 结合 HEFT 优先级[21]
Oracle	已知真实能力向量（σ=0）时的 HEFT 调度，作为性能上界
BayesSchedule	贝叶斯能力学习 + UCB 探索 + HEFT 调度框架

对照算法的选择遵循 3 个维度：①纵向覆盖。Random 提供下界，Oracle 提供理论上界，BayesSchedule 应落于两者之间并随学习逼近 Oracle；②静态横向比较。Greedy 代表局部贪心，HEFT 代表全局 DAG 调度，但二者均不具备在线学习能力，用于检验反馈机制 F 的贡献；③在线横向比较（本文新增的关键对照）。针对“会学习的算法对比不会学习的算法是否公平”这一问题，引入 3 类同样具备跨迭代自适应能力的在线学习基线，分别为 UCB1（频率主义置信上界）[22]、Thompson Sampling（Beta–Bernoulli 后验采样）[23] 与 QL–HEFT（表格型 Q–learning 结合 HEFT 优先级）[21]。三者与 BayesSchedule 在数学上构成阶梯式对比：UCB1 缺少贝叶斯先验与任务上下文，Thompson Sampling 采用离散 Beta–Bernoulli 后验而非连续高斯能力模型，QL–HEFT 则以值函数替代显式后验。通过与这 3 类在线学习算法进行比较，可验证“高斯共轭后验 + UCB 探索”这一组合在能力未知场景下的不可替代性。

Random、Greedy 和 HEFT 3 种算法均使用固定的能力

先验(初始估计值0.5)进行调度,不具备根据观测数据更新估计的能力。因此,即使串行执行100次工程,这3种算法的行为在每次迭代中基本保持不变,这一特性使其成为检验反馈机制F有效性的理想对照基线。

后续实验主要采用4类评价指标:①总成本。计算公式为$Cost=\Sigma_i C_i$,用于衡量完成全部任务的资源消耗;②总工期。计算公式为$makespan=max_i FT_i$,用于衡量项目整体完成时间;③违反率。计算公式为$VR=|\{i:FT_i>deadline_i\}|/n$,用于衡量deadline约束满足情况;④估计误差。计算公式为$Err=(1/mk)\Sigma_j\Sigma_d|\mu_j^d-c_j^d|$,用于刻画智能体能力后验均值与真实能力之间的平均偏差。上述指标共同覆盖成本、时间、约束满足和学习精度4个维度。

5.3 实验结果与分析

5.3.1 实验A:智能体规模效应

实验A的核心设计思路为固定任务规模和DAG结构不变,仅改变参与协同的智能体数量m,观测总调度成本随m的变化趋势。同时,考察BayesSchedule在不同团队规模下的调度质量,即更多的智能体是否总能带来更好的调度效果,还是会因调度开销而导致边际效益递减。固定n=100、ρ=0.15、σ=0.05,变化智能体数$m\in\{5,10,20\}$。表3为各算法在不同m值下的性能指标。可以看出,随着智能体数量m从5增加至20,BayesSchedule的总成本由6 673.5降至1 747.8,总工期由175.31降至113.88,说明合理编排能够释放并行协作收益。在所有智能体规模下,BayesSchedule的总成本均与Oracle基本持平(如m=20时为1 747.8对1 747.6),能力估计误差保持在0.01量级。以m=10为例,BayesSchedule的总成本(3 548.4)相较UCB1(5 734.0)、Thompson Sampling(4 996.3)和QL-HEFT(11 795.9)分别降低约38.1%、29.0%与69.9%,表明"高斯共轭后验+UCB探索+全局DAG调度"的组合在能力未知场景下更容易收敛并维持调度质量。BayesSchedule在m=10和m=20时的违反率分别为0.618和0.352,均低于HEFT及3类在线学习基线,说明其优势不仅体现在成本和工期上,亦体现在deadline约束满足能力上。

5.3.2 实验B:学习曲线与反馈机制验证

实验B聚焦于反馈机制F,考察其在线学习能力与收敛特性。在100次工程迭代中,BayesSchedule的贝叶斯后验不断接收新的观测数据并更新对智能体能力的估计,因此其调度质量应随迭代次数单调改善,最终逼近已知真实能力的Oracle上界。相比之下,不具备学习能力的对照算法(Random、Greedy、HEFT)的性能应在100次迭代中保持平稳。这一"学习曲线"与"平坦基线"的对比是验证反馈机制F有效性的最直接证据。固定n=100、m=10、ρ=0.15、σ=0.15,图1为代表性算法在100次串行工程中的调度成本、能力估计误差与违反率收敛曲线。可以看出,BayesSchedule在前20次迭代内将能力估计误差从0.49快速降至0.05,至第100次迭代时进一步降至0.01,呈现单调收敛特性;总成本逐渐下降,并最终与Oracle基本持平,总工期也显著降低。违反率曲线显示,BayesSchedule在学习过程中逐步接近Oracle水平,并明显优于不具备反馈学习能力的静态基线,说明贝叶斯后验更新不仅改善了成本估计,而且提升了deadline约束下的调度稳定性。作为对照,Random、Greedy和HEFT的能力估计误差始终停留在初始的0.49附近,无法随经验改善。这一鲜明对比直接证实了性能提升来自于反馈机制F的在线学习能力,而非调度框架本身。

Table 3 Performance indicators of various algorithms under different m values

表3 各算法在不同m值下的性能指标

m	算法	总成本	总工期	违反率	估计误差
5	Random	9 051.0	216.50	0.870	0.508
5	Greedy	7 831.1	205.30	0.873	0.508
5	HEFT	7 989.6	199.19	0.872	0.508
5	UCB1	8 991.6	212.75	0.893	0.398
5	Thompson	7 921.8	195.93	**0.868**	0.639
5	QL-HEFT	13 484.1	248.03	0.894	0.739
5	BayesSchedule	**6 673.5**	**175.31**	0.872	**0.009**
5	Oracle	6 625.9	174.53	0.872	0.000
10	Random	5 719.7	186.00	0.726	0.493
10	Greedy	4 189.7	173.46	0.651	0.493
10	HEFT	4 312.3	162.01	0.672	0.493
10	UCB1	5 734.0	200.58	0.817	0.385
10	Thompson	4 996.3	168.74	0.743	0.664
10	QL-HEFT	11 795.9	234.84	0.853	0.778
10	BayesSchedule	**3 548.4**	**142.68**	**0.618**	**0.011**
10	Oracle	3 520.4	142.16	0.621	0.000
20	Random	3 651.1	161.09	0.546	0.486
20	Greedy	2 284.1	141.87	0.390	0.486
20	HEFT	2 280.5	131.69	0.403	0.486
20	UCB1	2 964.8	212.06	0.640	0.383
20	Thompson	3 053.1	144.42	0.559	0.657
20	QL-HEFT	10 200.5	220.11	0.825	0.763
20	BayesSchedule	**1 747.8**	**113.88**	**0.352**	**0.014**
20	Oracle	1 747.6	113.42	0.351	0.000

5.3.3 实验C:噪声鲁棒性

实验B在固定噪声水平下验证了反馈机制的学习能力。然而,在实际多智能体系统中,智能体的表现可能因模型版本更新、输入分布变化等因素而显著不同。为考察BayesSchedule在不同噪声强度下的适应能力,即当智能体表现的随机性增大时,贝叶斯学习是否仍能有效收敛,本研究设计单因子变化方案:固定n=100、m=10、ρ=0.15,仅变化能力噪声$\sigma\in\{0.05,0.15,0.30\}$,其中$\sigma$=0.05代表较为稳定的智能体(如经过微调的专用模型),σ=0.30代表高度波动的智能体(如通用LLM在不熟悉领域的表现),通过观测不同σ值下智能体的性能退化程度来评估反馈机制F的噪声缓冲能力。值得注意的是,噪声参数σ在实际应用中具有明确的物理意义:较小的σ对应经过专门微调和优化的专用智能体,其在特定领域的输出质量相对稳定;较大的σ对应通用型大语言模型,其在面对超出其训练分布

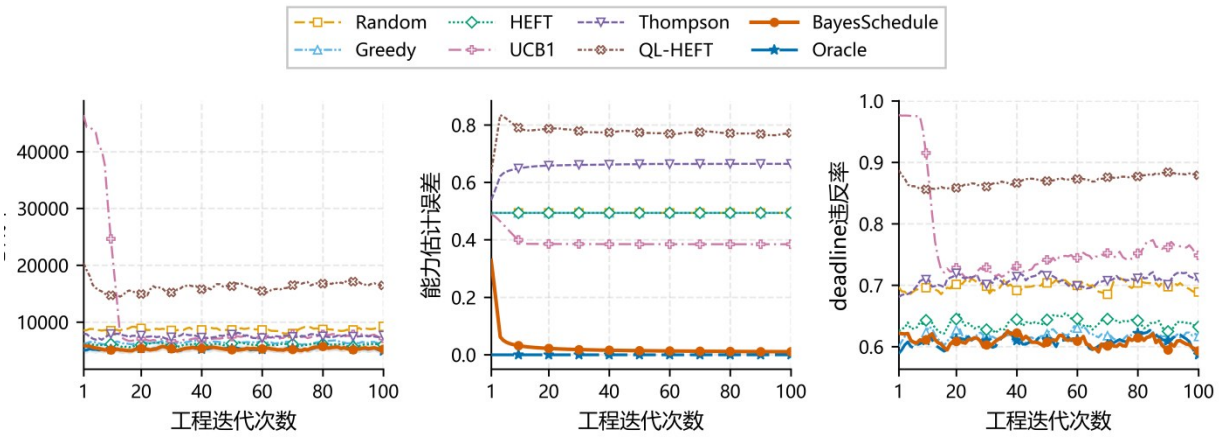

Fig. 1　Convergence curves of scheduling cost, capacity estimation error, and violation rate for representative algorithms

图 1　代表性算法的调度成本、能力估计误差与违反率收敛曲线

的任务时表现出高度不确定性。表 4 为各算法在不同噪声水平下的性能比较。可以看出，随着 σ 从 0.05 增至 0.30，BayesSchedule 的能力估计误差从 0.003 平缓上升至 0.032，但始终远低于所有不具学习能力或无显式能力建模的基线（Random/Greedy/HEFT 稳定在 0.493，QL-HEFT 高达 0.764），表明贝叶斯后验在高噪声下仍能有效收敛，具备良好的噪声鲁棒性。在总成本上，BayesSchedule 三档均紧贴 Oracle（差距分别约为 0.3%、4.6%、1.9%），并显著优于 UCB1、Thompson、QL-HEFT 3 类在线学习基线（如 σ=0.15 时分别降低 27.5%、25.1%、64.8%）。值得说明的是，本实验中各算法的总工期整体偏大，这是由于采用了关键路径更长的工作流子集。在此条件下，Greedy 凭借其纯局部贪心在总成本上偶有微弱领先，但其能力估计误差始终停留在 0.493，无法支撑跨项目的持续优化，这正凸显了 BayesSchedule“既学得准、又调度优”的综合优势。

5.3.4　实验 D：消融实验

前述实验验证了 BayesSchedule 整体的有效性，但未区分各组件的独立贡献。消融实验是评估复杂算法中各组件独立贡献的黄金标准方法，其关键在于每次仅移除一个组件，保持其余组件不变，从而将性能变化的原因精确归因于被移除的组件。消融实验结果见表 5，通过依次移除 BayesSchedule 的核心组件，包括贝叶斯学习、UCB 探索策略和匹配阈值约束，来量化各组件对最终性能的贡献，为算法设计决策提供实证依据。具体而言，设计 3 个消融变体：①w/o Learning。该变体移除贝叶斯后验更新机制，智能体能力估计始终保持初始先验值（0.5），算法退化为基于固定先验的标准 HEFT 调度；②w/o UCB。该变体保留贝叶斯学习但移除 UCB 探索策略，调度时仅使用后验均值 μ_j 而非 $\mu_j+\lambda\cdot\sqrt{\sigma_j^2}$，算法退化为纯贪心式利用已有估计；③w/o Threshold。该变体移除能力匹配阈值 θ 的约束，允许将任务分配给任意智能体而不检查能力匹配度。通过逐一比较每个变体与完整模型的性能差距，可以分离出各组件的独立贡献。如表 5 所示，贝叶斯学习组件的贡献最为显著，移除后算法退化为固定先验 HEFT，总成本由 7 321.2 升至 8 227.3，增幅为 12.4%，总工期由 247.91 升至 275.94，估计误差停留在 0.493，说明能力后验更新是性能提升的核心来源。相比之下，移除 UCB 探索后总成本变化为 −0.4%，移除匹配阈值 θ 后结果与完整模型完全一致，表明在默认 1 000 genome 工作流和 σ=0.15 设定下，HEFT 关键路径优先级已承担主要筛选作用，UCB 额外探索与阈值约束的边际贡献较小。

为进一步检验 UCB 探索与匹配阈值在不同工作流结构与噪声水平下的边际贡献，本研究在实验 D 的基础上补充设计了 4 组消融场景：以 1 000 genome（相对扁平、并行较多的弱依赖工作流）和 Montage（具有显著扇入扇出的强

Table 4　Performance comparison of various algorithms under different noise levels

表 4　各算法在不同噪声水平下的性能比较

σ	算法	总成本	总工期	违反率	估计误差
0.05	Random	10 004.4	284.52	0.645	0.493
0.05	Greedy	7 087.4	255.48	**0.548**	0.493
0.05	HEFT	6 799.9	238.28	0.568	0.493
0.05	UCB1	8 791.2	282.92	0.726	0.384
0.05	Thompson	8 164.1	260.20	0.609	0.843
0.05	QL-HEFT	17 729.5	351.36	0.878	0.764
0.05	BayesSchedule	**5 980.3**	**222.60**	0.551	**0.003**
0.05	Oracle	5 960.7	224.21	0.554	0.000
0.15	Random	10 063.9	285.35	0.650	0.493
0.15	Greedy	7 127.0	256.16	**0.552**	0.493
0.15	HEFT	6 846.0	238.94	0.572	0.493
0.15	UCB1	8 674.8	284.85	0.724	0.383
0.15	Thompson	8 392.1	261.83	0.627	0.665
0.15	QL-HEFT	17 865.8	352.86	0.880	0.764
0.15	BayesSchedule	**6 285.1**	**221.52**	0.555	**0.010**
0.15	Oracle	6 008.5	224.80	0.558	0.000
0.30	Random	10 239.4	287.71	0.665	0.493
0.30	Greedy	7 250.5	258.13	**0.563**	0.493
0.30	HEFT	6 980.4	240.82	0.584	0.493
0.30	UCB1	8 671.2	286.51	0.722	0.391
0.30	Thompson	8 135.8	264.53	0.623	0.461
0.30	QL-HEFT	18 245.5	356.90	0.886	0.764
0.30	BayesSchedule	**6 272.2**	**226.44**	0.567	**0.032**
0.30	Oracle	6 158.3	226.69	0.570	0.000

Table 5 Ablation study results

表 5 消融实验结果

变体	总成本	总工期	估计误差	成本增量/%
完整模型	7 321.2	**247.91**	0.011	—
w/o Learning	8 227.3	275.94	0.493	+12.4
w/o UCB	**7 290.8**	248.98	**0.010**	**−0.4**
w/o Threshold	7 321.2	**247.91**	0.011	0.0

依赖工作流）为两类对照拓扑，分别在 σ=0.15 和 σ=0.30 两个噪声水平下进行四变体消融，共 16 组对照（每组 20 次重复，独立随机种子）。结果见表 6。补充消融实验揭示了两个关键现象。其一，在所有 4 个测试场景下，移除贝叶斯学习均导致成本显著上升，尤其是在强依赖结构（Montage）上，贝叶斯学习的边际贡献从 1 000 genome 上的 9.36% 跃升至 23.13%，表明任务依赖结构越强、跨任务能力推断的价值越大，这是由于强依赖工作流中下游任务严重依赖上游执行结果，准确的能力估计能避免连锁性资源错配；其二，移除 UCB 探索与移除匹配阈值 θ 在 4 个场景下的成本变化均在±1.4% 以内，其中匹配阈值在全部场景下精确为 0%，UCB 探索仅在两个场景下略呈正向贡献而在另外两个场景下略呈负向。综合两次消融实验（默认场景与 4 个补充场景）可以得出结论：贝叶斯学习是 BayesSchedule 的核心驱动力，而 UCB 探索与匹配阈值作为可选增强模块，在本研究测试的 Pegasus 工作流中贡献均较小。因此在实际部署中，可将算法简化为“贝叶斯学习 + HEFT 调度”的最小配置以减少超参调试开销，而在跨任务能力差异极大或弱结构化任务图的场景下保留 UCB 作为可选项，以应对潜在的探索需求。

Table 6 Supplementary ablation study results

表 6 补充消融实验结果

场景	变体	总成本	总工期	估计误差	成本增量/%
1000genome, σ=0.15	Full	**7 368.6**	**245.63**	0.011	—
1000genome, σ=0.15	w/o Learning	8 058.7	270.87	0.493	+9.36
1000genome, σ=0.15	w/o UCB	7 454.9	249.08	**0.010**	+1.17
1000genome, σ=0.15	w/o Threshold	**7 368.6**	**245.63**	0.011	**0.00**
1000genome, σ=0.30	Full	7 630.6	247.81	0.033	—
1000genome, σ=0.30	w/o Learning	8 204.8	273.03	0.493	+7.53
1000genome, σ=0.30	w/o UCB	**7 525.6**	**245.82**	**0.032**	**−1.38**
1000genome, σ=0.30	w/o Threshold	7 630.6	247.81	0.033	0.00
Montage, σ=0.15	Full	**3 554.0**	144.19	0.011	—
Montage, σ=0.15	w/o Learning	4 376.2	159.59	0.493	+23.13
Montage, σ=0.15	w/o UCB	3 561.0	**136.84**	**0.010**	+0.19
Montage, σ=0.15	w/o Threshold	**3 554.0**	144.19	0.011	**0.00**
Montage, σ=0.30	Full	3 746.9	**144.64**	0.033	—
Montage, σ=0.30	w/o Learning	4 522.8	161.85	0.493	+20.71
Montage, σ=0.30	w/o UCB	**3 729.8**	146.56	**0.032**	**−0.46**
Montage, σ=0.30	w/o Threshold	3 746.9	**144.64**	0.033	0.00

5.3.5 *实验 E：任务规模可扩展性*

实验 A–D 均在固定或小范围变化的任务规模下展开，实验 E 则通过大幅度改变任务数量来考察 BayesSchedule 算法在不同工程规模下的适应能力和调度质量退化情况。该实验对于评估算法在实际软件开发中的实用性至关重要，因为工业级软件项目的任务数可能从数十个（小型功能迭代）到数百个（大型版本发布）不等。将任务规模从 n=50 变化至 n=200，同时固定智能体数 m=10、噪声水平 σ=0.15，观测各算法总成本、总工期和违反率的变化趋势。这一设计使得智能体与任务的比例从 1∶5 变化到 1∶20，可以测试算法在高负载条件下的表现。表 7 为各算法在不同任务规模下的性能比较。可以看出，BayesSchedule 在不同任务规模下均保持接近 Oracle 的性能：n=50 时总成本为 1 081.2，略优于 Oracle 的 1 101.9；n=100 和 n=200 时与 Oracle 的成本差距分别约为 1.0% 和 0.2%，能力估计误差保持在 0.01 量级。随着 n 从 50 增加至 200，BayesSchedule 相较 HEFT 的总成本优势保持稳定，n=200 时为 46 928.5 对 54 053.9，降低约 13.2%，总工期也由 676.24 降至 613.65，说明任务规模扩大 4 倍后算法未出现明显质量退化。在违反率方面，BayesSchedule 在 n=50、100、200 时分别为 0.328、0.582 和 0.801，均低于 HEFT 及 3 类在线学习基线，并与 Oracle 基本持平。需要说明的是，deadline 按真实关键路径保守设定，绝对违反率随规模增大整体上升，但 BayesSchedule 始终维持在接近理论上界 Oracle 的水平，说明其约束满足能力在大规模任务下仍具有稳定性。从工程实践角度看，n=200 对应包含前端界面、后端服务、数据库迁移、API 接口、单元测试和集成测试等数百个具体任务的中等规模版本发布，BayesSchedule 仍能保持接近 Oracle 的调度质量，表明该算法具备处理中等工业级项目的基本能力。

Table 7 Performance comparison of various algorithms under different task scales

表 7 各算法在不同任务规模下的性能比较

n	算法	总成本	总工期	违反率	估计误差
50	Random	2 377.9	147.01	0.507	0.493
50	Greedy	1 475.5	129.45	0.403	0.493
50	HEFT	1 369.6	121.14	0.381	0.493
50	UCB1	1 653.1	138.46	0.512	0.385
50	Thompson	1 807.8	128.16	0.531	0.662
50	QL-HEFT	3 694.4	166.51	0.767	0.766
50	BayesSchedule	**1 081.2**	**102.16**	**0.328**	**0.015**
50	Oracle	1 101.9	101.72	0.326	0.000
100	Random	12 557.5	332.79	0.691	0.493
100	Greedy	9 077.8	292.23	0.586	0.493
100	HEFT	8 227.3	275.94	0.614	0.493
100	UCB1	10 135.5	320.94	0.752	0.384
100	Thompson	10 293.7	284.76	0.717	0.663
100	QL-HEFT	20 721.6	396.47	0.880	0.761
100	BayesSchedule	**7 321.2**	**247.91**	**0.582**	**0.011**
100	Oracle	7 249.2	245.48	0.580	0.000
200	Random	62 889.1	748.90	0.845	0.493
200	Greedy	49 062.5	649.35	**0.798**	0.493
200	HEFT	54 053.9	676.24	0.822	0.493
200	UCB1	54 739.7	757.02	0.898	0.384
200	Thompson	54 180.5	642.31	0.866	0.664
200	QL-HEFT	106 941.4	935.56	0.930	0.764
200	BayesSchedule	**46 928.5**	**613.65**	0.801	**0.009**
200	Oracle	46 824.4	606.79	0.799	0.000

5.3.6　综合性能评价

为了从全局视角评价各算法的综合表现，以热力图形式展示 5 种代表性算法在总成本、总工期、违反率与能力估计误差 4 个核心指标上的归一化性能（数值越低越好），为选择适合不同场景的调度策略提供直观参考。结果如图 2 所示。可以看出，BayesSchedule 在总成本、总工期和能力估计误差 3 个维度上均接近 Oracle 的最优水平，其违反率亦与 Greedy、Oracle 持平（均为最低水平）。热力图的颜色分布还揭示了一个有趣的模式：Random 和 Greedy 在不同指标上的表现参差不齐，HEFT 虽然在静态调度维度上表现出色，但在需要动态学习能力的指标上明显不足。只有 BayesSchedule 在所有维度上实现了均匀的低值分布，体现了其作为综合性调度框架的优越性。总体而言，BayesSchedule 是唯一在所有指标上接近理论最优的实用算法。

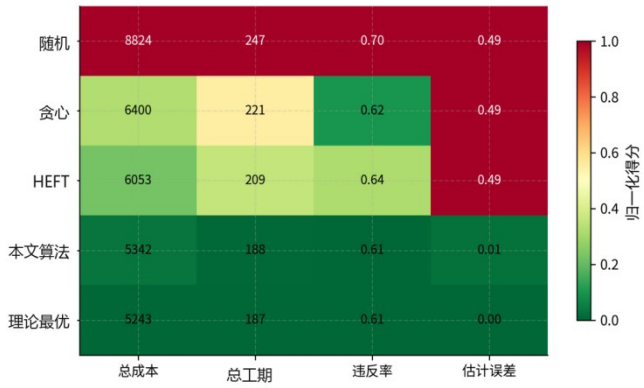

Fig. 2　Summary performance heatmap

图 2　综合性能热力图

6　从实验到范式：协同开发的深层影响

形式化模型与实验验证不仅具有技术描述功能，而且揭示了多智能体协同开发对软件工程实践的深层影响。需要说明的是，以下范式讨论建立在前述仿真实验的理想化前提之下，即智能体能力静态可学习、任务图结构已知、沟通与知识传递开销未纳入模型，因此相关推论应理解为基于受控仿真证据的趋势性洞察，其在真实工程环境中的成立程度有待实证检验。基于前述实验数据，分析模型中任务映射函数 M 和调度策略 Π 两个核心概念如何映射为软件工程组织管理中的范式转变。

6.1　*M* 函数的工程实践意义

任务映射函数 *M* 负责将任务分配给能力匹配的智能体。实验 B 的结果表明，BayesSchedule 通过反馈机制 *F* 持续优化 *M* 函数的映射质量，100 次迭代后的分配质量与 Oracle 基本持平（成本差距在统计噪声内），这意味着传统软件开发中需要项目经理人工评估开发者能力、协调任务分配的工作量可以通过算法自动完成。Jwo 等[19]研究亦表明，基于智能体的敏捷开发可将团队协调的管理开销降低约 40%，这一转变的实质是软件工程的管理重心从“管人”转向“编排能力”，项目管理者的核心职责从“人员考核与激励”转向“智能体能力配置与协同策略设计”。

6.2　*Π* 函数的工程实践意义

调度策略 *Π* 决定了任务执行的时序与资源分配。实验 A 和 B 的结果表明，在本研究提供的仿真条件下（智能体能力向量静态且可通过观测学习，任务依赖结构已知），BayesSchedule 的自适应调度使 m=20 时的总工期（113.88）低于 m=5 时（175.31），说明在智能体组织得当的前提下，扩大团队规模确能提升效率，这与 Brooks 定律指出的“向已延迟的项目增加人手只会使其更加延迟”的经典结论形成了有趣对照[28]。需要强调的是，该对照仅在本研究的理想化仿真假设下成立，真实工程中的沟通开销、知识传递成本与需求变更等因素并未建模，因此这一结果应理解为多智能体协同在合理组织架构和调度策略支持下有可能突破传统人力协作规模瓶颈的初步证据，而非确定性结论。人类管理者的角色随之从“指挥者”转变为“规则制定者”和“审核者”[18]。

6.3　组织形态的深层变革

从更宏观的视角审视，多智能体协同开发引发的变革不仅停留在 *M* 函数和 *Π* 策略的技术层面，更深刻地影响着软件开发组织的形态与运作模式。传统软件开发组织遵循科层制结构，从项目经理到团队负责人再到一线开发者，信息沿层级链路传递，决策权集中于管理层。然而，多智能体系统的引入打破了这种固定的层级结构，使组织形态向更加扁平化、网络化的方向演进。在多智能体协同框架下，任务分配不再依赖管理者的主观判断，而是由算法根据客观的能力数据自动完成；进度追踪不再依赖定期会议汇报，而是通过反馈机制 *F* 实时获取执行状态并动态调整。这种从“经验驱动”到“数据驱动”的范式转变不仅提高了资源配置效率，而且降低了人为偏见对项目决策的影响。实验 B 和 C 的结果从定量角度支持了这一观点，BayesSchedule 的自适应学习能力在 100 次迭代中实现了从初始的高不确定性到近最优决策的平稳过渡，其决策质量的提升轨迹远比人工学习过程更加可预测。

综合比较来看，BayesSchedule 的优势并非集中在单一指标的极端最优，而是在成本、时延、估计误差与约束满足之间取得了更稳定的平衡。这一点对于软件工程场景尤为重要，因为真实项目通常同时受交付周期、资源预算和质量要求约束，单纯优化某一指标往往会造成其他维度恶化，综合性能的均衡性比局部最优更具工程意义。此外，传统团队通常以岗位边界和人力投入作为组织基础，而在多智能体协同环境中，更关键的是定义任务输入输出、权限边界和反馈闭环，使不同角色的智能体能够在统一协议下稳定协作。因此，组织治理的重点将逐步从层级汇报和人工协调转向流程可观测性、能力可评估性以及调度规则可复用性。对于软件工程而言，这意味着项目管理方法、

开发平台建设和质量控制机制均需随之调整。

7 结语

7.1 总体结论

本文从软件工程视角出发，对多智能体协同开发进行了系统研究，提出包含任务分解、能力映射与反馈优化的形式化模型$S=<T, A, F, M, \Pi>$，构建了按智能体类型和任务特征的二维分类体系，并基于该模型提出了以贝叶斯能力学习为核心驱动力的调度算法BayesSchedule（UCB探索与匹配阈值作为可选增强模块）。通过在Pegasus真实科学工作流基准上的5组系统化仿真实验对模型核心组件进行了全面验证，实验结果表明：①BayesSchedule在100次迭代后能力估计误差降至0.01，总成本与理论最优基准Oracle基本持平，总工期较HEFT降低10.0%，且总成本较UCB1、Thompson Sampling、QL-HEFT三类在线学习基线分别降低30.2%、28.3%、67.9%，直接验证了反馈机制F与调度策略Π的有效性；②算法在噪声强度σ从0.05增至0.30时仍保持与Oracle的极小差距，展现了优异的鲁棒性；③消融实验证实贝叶斯学习组件是算法性能的核心驱动力，移除后总成本上升12.4%。

7.2 研究存在的不足

本研究仍存在一些局限：一是反馈机制F的设计尚不完善。实验D的消融分析表明，UCB探索策略在不同工作流结构下的边际贡献存在差异，高噪声场景的自适应探索策略亟需优化；二是成本控制机制缺失。Liu等[5]研究表明，当前生态系统在测试基础设施和文档质量方面仍存在明显短板；三是人机权责划分模糊。在关键决策环节如何界定人类开发者与智能体之间的责任边界尚无明确规范[14]；四是可解释性不足。当前动态调度算法虽然能够产生高质量的任务分配方案，但其决策过程对于人类管理者而言仍是黑箱，缺乏直观的可解释性支持，不利于在安全关键领域的推广应用。具体而言，本文尚未进一步分析不同任务类型之间的迁移效应，例如某个智能体完成代码生成任务后，是否会在相关测试或重构任务中表现出持续收益；亦未建模不同能力维度之间的相关性，现实中设计能力、实现能力和审查能力往往并非独立变化。此外，实验默认任务图与能力分布在单次仿真期间保持静态，而真实研发过程中的需求变更、任务插入和优先级调整会持续改变调度空间。上述因素均可能影响算法在真实场景中的收敛速度与最优性边界，需要在后续研究中结合真实项目数据进一步验证。

实验仍基于仿真环境，尚未覆盖任务迁移效应、相关噪声、超大规模场景和更高维能力空间，因此其外部有效性仍需进一步验证。此外，需要对各实验中绝对违反率整体偏高的现象（如实验C中Oracle的违反率亦达0.55、实验E大规模场景下达0.80）作出说明。本文任务deadline的计算方式为：以任务在DAG中所处拓扑深度对应的关键路径累积基础执行成本为基准，乘以松弛系数γ=1.5预留时间裕度。该设定之所以仍导致大量违反，原因有三：其一，基准按照基础执行成本计算，未计入能力不完全匹配带来的实际执行时间膨胀（执行时间为基础成本除以能力匹配度，匹配度通常小于1）；其二，Bernoulli-Gaussian失败模型中约5%~7%的任务会发生灾难性失败并占用2倍deadline的阻塞窗口，失败任务本身计为违反且其下游任务被连锁推迟；其三，关键路径上任意一个任务的延迟会沿依赖链传播，使深层任务的实际完成时间系统性超出按静态关键路径估算的deadline。在此设定下，即使是已知真实能力的Oracle也无法避免大量违反，这恰恰说明违反主要源于deadline设定的紧约束性与失败模型的随机性，而非调度算法本身的缺陷。因此，本文的违反率指标仅用于算法之间的相对比较（衡量在同等紧约束下哪种算法的约束满足能力更强），不代表实际部署中的服务水平，实际部署时可根据业务容忍度调大松弛系数以获得期望的违反率水平。本文采用的Bernoulli-Gaussian混合模型是对LLM执行表现的一种简化近似，其较好地刻画了编译中断、运行时崩溃、上下文溢出等可被自动检测的显性失败，但现实中LLM还存在生成功能正确但低效的代码、输出与需求语义偏离（答非所问）、引入隐蔽缺陷等隐性失败模式。这类失败在任务完成时刻难以被二元事件捕获，往往要等到集成测试甚至生产阶段才暴露。隐性失败的存在意味着真实场景中的能力观测噪声可能呈现有偏、重尾甚至延迟反馈的特性，这会减缓贝叶斯后验的收敛速度并可能引入系统性估计偏差。本文即时、无偏的观测假设因而构成结论泛化性的一个边界条件，将隐性失败建模为延迟揭示的多级质量信号（如结合代码评审与测试覆盖信息的连续质量分数），并研究相应的延迟反馈贝叶斯更新机制是贴近LLM真实行为的重要方向。

此外，本文默认智能体能力在整个实验期间保持稳定，尚未考虑模型版本更新、提示工程调整或外部工具可用性变化带来的能力漂移，任务成本函数也未显式纳入通信延迟、上下文切换、多轮审查等真实协作开销。对于以代码仓库、缺陷单和测试流水线为载体的工程环境，这些因素都可能改变调度收益的绝对数值。因此，当前实验结果更适合作为机制有效性的初步证据，而非对生产环境收益的直接量化。

7.3 未来研究方向

基于实验发现和局限性分析，未来研究应聚焦以下方向：一是构建软件工程专用的任务依赖基准。本研究受限于软件工程领域缺乏公开的带依赖结构任务图数据集这一事实，转而采用DAG调度社区的Pegasus标准基准，未来可从真实软件项目的构建系统依赖图（如Bazel/Make的目标依赖）、持续集成流水线与缺陷修复任务链中抽取带真实执行属性的任务图，建立软件工程专用的调度评测基

准;二是研究面向非平稳环境的自适应调度策略,将贝叶斯模型扩展为在线变点检测框架以应对智能体能力漂移。实验C的结果表明现有模型在静态噪声下表现良好,但智能体能力随时间变化的动态场景尚未涉及;三是探索人机协同的最优分工模式,从工程实践角度界定人在回路中的最佳角色[16];四是开展更大规模(n>1 000, m>100)的可扩展性验证,评估分层协同架构的实际性能极限;五是将形式化模型从离散任务调度扩展到持续集成/持续部署(CI/CD)管道的优化,探索多智能体系统在DevOps全流程中的应用潜力。

多智能体协同开发可将软件工程中的任务分配与调度过程转化为可学习、可优化的问题,为从"人员管理"转变为"能力编排"提供了方法基础。本文不仅给出了一套具体算法,更在于为多智能体协同软件开发提供了可量化、可比较、可迭代优化的分析框架。借助能力估计、任务映射和反馈更新等显式变量,原本依赖经验判断的组织协调过程得以转化为可验证的工程问题,这为后续在真实研发流程中开展数据驱动的调度优化奠定了基础。

参考文献:

[1] STANDISH GROUP. CHAOS 2020: beyond infinity [R]. Boston: The Standish Group International, 2021.

[2] BLOCH M, BLUMBERG S, LAARTZ J. Delivering large-scale IT projects on time, on budget, and on value [R]. New York: McKinsey & Company, 2012.

[3] HE J, TREUDE C, LO D. LLM-based multi-agent systems for software engineering: literature review, vision and the road ahead [J]. ACM Transactions on Software Engineering and Methodology, 2025, 34(5): 1-30.

[4] LI Y, XU F, XIE G Q, et al. Survey of development and application of multi-agent technology [J]. Computer Engineering and Applications, 2018, 54(9): 13-21.
李杨, 徐峰, 谢光强, 等. 多智能体技术发展及其应用综述 [J]. 计算机工程与应用, 2018, 54(9): 13-21.

[5] LIU D, FENG S, ZHANG Y, et al. A large-scale study on the development and issues of multi-agent AI systems [DB/OL]. https://arxiv.org/abs/2601.07136。

[6] PARADIS E, GREY K, MADISON Q, et al. How much does AI impact development speed? an enterprise-based randomized controlled trial[C]// Proceedings of the IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice, 2025: 618-629.

[7] JIMENEZ C E, YANG J, WETTIG A, et al. SWE-bench: can language models resolve real-world GitHub issues? [C]//Vienna: Proceedings of the 12th International Conference on Learning Representations, 2024.

[8] Quanta Computer. Quanta adopts Microsoft generative AI coding tools with 33% efficiency improvement [EB/OL]. https://ieknet.iek.org.tw/.
广达电脑. 广达导入微软生成式AI编码工具,开发效率提升33% [EB/OL]. https://ieknet.iek.org.tw/.

[9] SONG F, AGARWAL A, WEN M. The impact of generative AI on collaborative open-source software development: evidence from GitHub Copilot [DB/OL]. https://arxiv.org/abs/2410.02091.

[10] HONG S, ZHENG X, CHEN J, et al. MetaGPT: meta programming for a multi-agent collaborative framework [C]//Proceedings of the 12th International Conference on Learning Representations, 2024: 1-21.

[11] GRAVITAS S. AutoGPT: autonomous AI agent platform [EB/OL]. https://github.com/Significant-Gravitas/AutoGPT.

[12] LYU W, XIAO Y, ZHANG Y, et al. Self-organizing multi-agent systems for continuous software development [DB/OL]. https://arxiv.org/abs/2603.25928.

[13] IDC. Developers aren't just using AI agents, they're building them[EB/OL]. https://dev.idc.com.

[14] RUSSINOVICH M, HANSELMAN S. Redefining the software engineering profession for AI [J]. Communications of the ACM, 2026, 69(3): 34-36.

[15] JENNINGS N R. On agent-based software engineering [J]. Artificial Intelligence, 2000, 117(2): 277-296.

[16] TANG Y, RUNKLER T. LLM-based agentic systems for software engineering: challenges and opportunities[C]//Proceedings of Software Engineering, 2026: 45-58.

[17] ZHANG W. A survey on multi-agent workflow automation: methods, taxonomies, and future directions [D]. Hong Kong: Hong Kong University of Science and Technology, 2025.

[18] BENKOVICH N, VALKOV V. Agyn: a multi-agent system for team-based autonomous software engineering [DB/OL]. https://arxiv.org/abs/2602.01465.

[19] JWO J S, LIU C H. Agent-based agile software development: a systematic review [J]. Journal of Systems and Software, 2025, 213: 112034.

[20] TOPCUOGLU H, HARIRI S, WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274.

[21] TONG Z, DENG X, CHEN H, et al. QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment [J]. Neural Computing and Applications, 2020, 32(10): 5553-5570.

[22] AUER P, CESA-BIANCHI N, FISCHER P. Finite-time analysis of the multiarmed bandit problem [J]. Machine Learning, 2002, 47(2-3): 235-256.

[23] AGRAWAL S, GOYAL N. Further optimal regret bounds for Thompson sampling [C]//Proceedings of the 16th International Conference on Artificial Intelligence and Statistics, 2013: 99-107.

[24] SRINIVAS N, KRAUSE A, KAKADE S, et al. Gaussian process optimization in the bandit setting: no regret and experimental design[C]//Proceedings of the 27th International Conference on Machine Learning, 2010: 1015-1022.

[25] TAMBON F, MORADI D A, NIKANJAM A, et al. Bugs in large language models generated code: an empirical study [J]. Empirical Software Engineering, 2025, 30: 65.

[26] Amazon Web Services. Amazon frontier agents now available, enterprise IT operations and security enter fully autonomous era [EB/OL]. https://www.doit.com.cn/p/556028.html.
亚马逊云科技. 亚马逊云科技前沿Agent正式可用,企业IT运维与安全迈入全自主时代 [EB/OL]. https://www.doit.com.cn/p/556028.html.

[27] COLEMAN T, CASANOVA H, POTTIER L, et al. WfCommons: a framework for enabling scientific workflow research and development [J]. Future Generation Computer Systems, 2022, 128: 16-27.

[28] BROOKS F P. The mythical man-month: essays on software engineering [M]. Boston: Addison-Wesley, 1995.

(责任编辑:尹晨茹)