Jingfeng



智能体模型量化、压缩与部署

从数值误差到任务可靠性：权重、KV cache 与服务系统

Jingfeng · 2026 年 10 月 22 日修订

本书围绕权重、激活、KV cache 与 Agent 轨迹的共同预算展开。

摘要

大型语言模型（LLM）成为智能体（agent）的策略核心后，推理过程不再是一段独立文本生成，而是由工具调用、检索、代码执行、网页操作、记忆读写和多轮规划组成的长闭环。模型压缩因此出现了一个容易被忽略的事实：权重只是状态空间的一部分。激活值决定每一步的算子开销，KV cache 决定长轨迹的显存和带宽，工具 schema 与历史观察决定上下文长度，运行时的分页、调度、反量化和回滚决定最终延迟；任何一处局部失真都可能被后续步骤放大成任务失败。

本书从仿射量化的数值基础出发，解释舍入、截断、分组与元数据开销，按误差补偿机制组织权重与激活量化，并区分训练补偿、KV 数值量化和历史位置选择。部署部分进一步连接低比特算子、分页、前缀复用、调度和异构存储，把局部算法放回工具调用合法率、参数正确率、最终状态成功率、重复可靠性、时延和峰值内存的共同账本。

最后，本书给出轨迹感知校准、动作边界保持率、风险敏感混合精度、反事实重放和可复现实验协议。研究的起点是在固定总预算下检验语言指标与行为指标何时一致、何时失配，再比较敏感度估计、精度分配与系统实现能否改善这种失配；结论由配对轨迹与资源测量决定。

本章抓住这些

读完本书后，你应该能够回答五个问题：

1. 量化到底压缩了什么，哪些对象会在 Agent 中成为新的瓶颈？
2. 为什么同一个 4 bit checkpoint 在不同 kernel、上下文长度和工具轨迹上表现不同？
3. 如何从 PTQ、QAT、KV 压缩和系统 co-design 中选择可复现的 baseline？
4. 如何在明确的任务范围内检验工具选择、终止、记忆检索和拒绝决策的变化？
5. 怎样把只报告 PPL 的结果扩展为可复现、可证伪的 Agent 压缩研究？

如何使用这本书

本书面向具备基础数学与编程知识、希望读论文并开展实验的读者。每条阅读路线都围绕三个问题展开：系统在何处决定与执行动作，误差如何改变后续状态，以及什么证据支持资源收益与行为代价。

前置知识与起步自检

- **线性代数**。理解矩阵乘法、转置、范数和基本导数。看到 $X \in \mathbb{R}^{n \times d_{\text{in}}}$、$W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ 时，能写出 XW 的形状，并把 Frobenius 范数展开为元素平方和。二阶补偿需要的 Hessian 会在对应章节解释。
- **概率与统计**。区分条件概率、独立性、期望与样本比例；知道有限任务集上的成功率不是所有部署任务的保证。置信区间、配对比较和重复运行指标在评测部分展开。
- **Transformer 推理**。了解 token、因果注意力、Q/K/V 和自回归生成。若尚不能解释历史 KV 为什么可以复用，可先读“KV Cache：Agent 长轨迹的第二个模型”，再回到 KV 压缩算法。
- **Python 与结构化数据**。能运行脚本、查看数组的 shape/dtype、解析 JSON，并理解工具参数和环境返回。先用手算与合成实验核对协议，再接入模型；阅读本书不要求事先掌握 CUDA 编程。

起步时可做两个自检：给仿射量化写出编码与反量化的先后顺序；给工具调用画出“模型候选动作、参数检查、实际执行、状态验证”四个环节。前者定位数值误差，后者定位任务失败的责任边界。

按目标选择三条阅读路线

1. **从原理入门**。先读闭环、压缩对象与指标三章，再读“量化的数学基础与数值格式”“PTQ、QAT 与校准数据”。按输入缩放、二阶补偿、低秩修复的次序理解权重量化，再进入激活与 KV。阶段产出是一张能手算的误差和字节表，以及一个能解释每项元数据的格式说明；最后用附录实验检查推导与代码是否一致。
2. **以部署为入口**。先建立权重、激活和 KV 账本，读 prefill/decode、分页与前缀复用，再读低 bit serving、异构部署及部署配方。遇到数值格式与轴的问题时回查基础章节。阶段产出是固定负载下的硬件、软件、kernel、TTFT、TPOT 与峰值内存记录卡；随后回到任务配对评测，检查资源收益是否伴随行为退化。
3. **研究 Agent 可靠性**。先读指标与误差传播，再读轨迹校准、动作边界保护、安全测试、benchmark 与首次分叉分析。用一个冻结工具输出的小任务建立浮点/量化配对，按诊断结果选择权重、激活或 KV 方法。阶段产出是一条可重放的失败链、一项预先定义的指标和一组消融；再用“论文阅读地图”把假设与原文证据对应起来。

三条路线最终都汇合到“Benchmark 设计：从单步准确率到完整任务”、配对评测伪代码和最小量化与部署实验。“十二周学习与实验路线”给出更长的实践安排，读者可按设备条件缩减方法数量，保留基线、账本与验证闭环。

章节之间怎样交接

基础章节交付“码字如何还原为数值、误差如何计量、实际存储多少字节”；学完代表方法，应能说明它改动了哪一个优化自由度。KV 部分把静态权重扩展为随请求长度变化的运行状态，为后面的长轨迹问题建立资源尺度。

从数值方法进入 Agent-aware 部分时，先固定同一状态比较动作，再放开执行，观察首次分叉后的后果。从行为目标进入服务系统时，把候选精度配置交给真实 kernel 和请求负载检验；从服务系统进入评测部分时，再把排队、工具等待、超时和恢复纳入同一任务预算。每次交接都保留上一阶段的假设，避免把局部误差下降或单算子变快直接写成端到端结论。

本书与其他分册的边界

《LLM 智能体系统》讨论完整闭环架构、类型化状态、工具契约、记忆生命周期和上下文编排；本书解释这些对象被压缩后产生的资源与行为变化。通用硬件选型、网络与集合通信、集群调度可接读《大模型推理加速与 AI 基础设施》；线程层级、同步、GEMM 与算子优化可接读《Kernel 与 CUDA 编程》。这些主题均有对应分册，本书按主题转引，不依赖外书的页码或章号。

本书保留理解压缩所需的注意力、缓存、服务指标和工具执行边界解释；选择一种量化方案、建立账本和完成最小配对实验，不需要先通读其他分册。

怎样区分书中的证据

手算与合成数据用于检查机制，论文报告只支持其原有模型、任务与实现条件，本次运行记录只支持记录中的本地实验。理论 KV 字节数不等于实测显存，透明的合成工具选择器也不等于生产 LLM。尚未运行的 GPU 延迟或加速配置以实验方案给出；具体命令、规模和结果状态见最小实验附录。

阅读时始终追问的六个问题

1. 论文压缩的对象是权重、激活、KV、上下文，还是训练状态？
2. 压缩发生在训练前、校准时、运行时，还是请求之间？
3. 它优化的是张量误差、语言损失、单步动作，还是完整任务结果？
4. 需要什么校准数据？这些数据是否覆盖工具、代码、图像和长记忆？
5. 理论 bit 数是否被 scale、zero-point、索引、残差和反量化 kernel 抵消？
6. 失败时能否定位、重放、回滚，并且让用户知道系统不确定？

目录

摘要 ii

如何使用这本书 iii

符号与缩写 xiii

第一部分　问题定义：为什么 Agent 压缩不同于普通 LLM 压缩 1

第一章　Agent 是一个受资源约束的闭环系统 2

1.1 从一次生成到长轨迹 . . . 2
1.2 量化误差如何被闭环放大 . . . 2
1.3 四类资源预算 . . . 2
1.4 Agent 压缩的目标函数 . . . 3

第二章　压缩的对象：从参数到整个 Agent 状态 4

2.1 一张完整的对象地图 . . . 4
2.2 模型权重 . . . 4
2.3 激活与中间状态 . . . 4
2.4 KV cache . . . 4
2.5 上下文、记忆与工具描述 . . . 5
2.6 训练状态与草稿模型 . . . 5

第三章　从压缩率到任务效用：指标体系 6

3.1 不要把 PPL 当作 Agent 的充分统计量 . . . 6
3.2 五层指标 . . . 6
3.3 动作边界保持率 . . . 6
3.4 重复可靠性：pass^k 与 pass@k . . . 6

第四章　量化的数学基础与数值格式 8

4.1 统一仿射量化 . . . 8
4.2 粒度：per-tensor 到 per-token . . . 8
4.3 手算分组误差与真实字节数 . . . 8
4.4 误差不止是 MSE . . . 10
4.5 INT、FP 与 NF 格式 . . . 10

第五章　PTQ、QAT 与校准数据 11

5.1 Post-Training Quantization . . . 11

5.2 Quantization-Aware Training 11
5.3 校准集的四个层次 11
5.4 数据泄漏与校准集过拟合 11

第六章 常见压缩技术的全景 12
6.1 剪枝与稀疏化 12
6.2 低秩分解与适配器 12
6.3 知识蒸馏 12
6.4 架构级缩小、MoE 与动态路由 12
6.5 上下文压缩与语义记忆 12

第七章 W8A8：激活离群值与等价变换 14
7.1 LLM.int8：为什么八 bit 也不简单 14
7.2 SmoothQuant：把困难从激活迁移到权重 14
7.3 RPTQ 与通道重排 14

第八章 W4A16：GPTQ、AWQ 与 OmniQuant 15
8.1 GPTQ：用二阶信息做逐层舍入 15
8.2 AWQ：保护 activation-aware 的显著通道 15
8.3 OmniQuant：学习 clipping 与等价变换 15
8.4 三种方法的实验读法 16

第九章 2–3 bit 权重：极限压缩的路线与代价 17
9.1 SpQR 与稠密 – 稀疏混合 17
9.2 SqueezeLLM：非均匀码本与 dense/sparse 分解 17
9.3 QuIP 与 QuIP#：不可相干化和格码本 17
9.4 AQLM 与 additive quantization 17
9.5 QuaRot、SpinQuant 与离群值消除 17
9.6 BitNet 与训练时低比特 18

第十章 量化后补偿：QAT、QLoRA 与低秩修复 19
10.1 LLM-QAT 与数据自由训练 19
10.2 QLoRA：冻结量化底座，训练低秩适配器 19
10.3 LoftQ：让量化与 LoRA 初始化相互适配 19
10.4 蒸馏信号与训练投入 19

第十一章 激活量化：动态范围、校准与注意力路径 20
11.1 激活为什么比权重难 20
11.2 Attention 中的不同张量 20
11.3 混合精度分配 20

第十二章 KV Cache：Agent 长轨迹的第二个模型 **21**
12.1 从注意力公式理解 cache . . . 21
12.2 显存、带宽和并发的三重约束 . . . 21
12.3 KV 的错误传播 . . . 21

第十三章 KV Token 选择：H2O、Scissorhands、SnapKV 与 PyramidKV **22**
13.1 H2O 与重要 token . . . 22
13.2 Scissorhands：重要性持久性 . . . 22
13.3 SnapKV：生成前预测 head 的保留位置 . . . 22
13.4 PyramidKV：按层分配预算 . . . 22
13.5 保留、摘要与删除的组合 . . . 23

第十四章 KV Quantization：KIVI、KVQuant 与 GEAR **24**
14.1 为什么 K 与 V 需要不同轴 . . . 24
14.2 KVQuant：离群值、RoPE 与非均匀数据类型 . . . 24
14.3 GEAR：量化加低秩/残差修复 . . . 24
14.4 KV 量化的工程检查表 . . . 24

第十五章 语义压缩：上下文、RAG、Memory 与工具 Schema **25**
15.1 数值压缩与语义压缩的边界 . . . 25
15.2 RAG 与证据链 . . . 25
15.3 长期记忆 . . . 25
15.4 工具 Schema 与前缀缓存 . . . 25
15.5 Agent 专用前沿信号 . . . 25

第二部分 Agent-aware 压缩：从张量误差到行动可靠性 **27**

第十六章 量化误差在 Agent 轨迹中的传播 **28**
16.1 局部误差、离散动作与状态转移 . . . 28
16.2 四种常见失败链 . . . 28
16.3 错误放大模型 . . . 28
16.4 量化后的行为等价并非文本等价 . . . 28

第十七章 轨迹感知校准与混合精度分配 **29**
17.1 把 calibration sample 改写为 Agent event . . . 29
17.2 敏感度估计 . . . 29
17.3 离散分配问题 . . . 29
17.4 一个可复核的最小算法 . . . 30

第十八章 工具调用、记忆检索与停止边界的保护 **31**
18.1 工具调用是结构化输出问题 . . . 31

18.2 保护动作 token 的三种策略 31
18.3 记忆检索边界 31
18.4 停止、升级与人工接管 31
18.5 一组最小成对行为测试 32

第十九章 安全、隐私与风险敏感压缩 33
19.1 压缩改变攻击面 33
19.2 纵深防御 33
19.3 风险敏感目标 33
19.4 安全评测的分母与证据 33

第三部分 从低 bit checkpoint 到可用的 Agent 服务 34

第二十章 推理栈：prefill、decode 与 Agent 调度 35
20.1 两阶段的硬件性质 35
20.2 连续批处理与请求异质性 35
20.3 显存账本 35

第二十一章 PagedAttention、前缀复用与结构化运行时 36
21.1 PagedAttention 36
21.2 Prefix cache 的正确性条件 36
21.3 SGLang 与 RadixAttention 36
21.4 结构化输出与量化 36

第二十二章 低 bit Serving：Atom、QServe 与 kernel co-design 37
22.1 为什么 checkpoint 压缩不等于加速 37
22.2 Atom 37
22.3 QServe：W4A8KV4 37
22.4 量化 kernel 的检查表 37

第二十三章 异构、分布式与长上下文部署 38
23.1 FlexGen 与 offloading 38
23.2 Splitwise 与 DistServe 38
23.3 Tensor、pipeline 与 expert parallelism 38
23.4 长上下文与跨设备 KV 38

第二十四章 边缘与移动端：LLM in a Flash 与实际约束 39
24.1 手机不是一张小 GPU 39
24.2 边缘 Agent 的分层模型 39
24.3 能耗与热约束 39

第二十五章 Speculative Decoding、模型级联与早退 **40**
25.1 Speculative decoding 的基本机制 40
25.2 Agent 特有的接受条件 40
25.3 级联与 early exit 40

第四部分 评测、复现与研究路线 **41**

第二十六章 部署配方：从 baseline 到生产候选 **42**
26.1 配方 A：单卡研究 baseline 42
26.2 配方 B：云端高并发 42
26.3 配方 C：边缘/本地 42
26.4 配方 D：Agent-aware 研究 42

第二十七章 Benchmark 设计：从单步准确率到完整任务 **43**
27.1 环境与基准 43
27.2 任务分层 43
27.3 公平比较的固定项 43
27.4 匹配任务与配对比较 44
27.5 统计与置信区间 44

第二十八章 可观测性、重放与首次分叉分析 **45**
28.1 Trace schema 45
28.2 首次分叉 45
28.3 反事实重放 45

第二十九章 前沿研究主轴与可证伪问题 **46**
29.1 八条值得持续追踪的主轴 46
29.2 可证伪的研究问题模板 46
29.3 三个可执行项目 46

第三十章 十二周学习与实验路线 **47**

附录：公式、伪代码与参考文献 **49**

附录 A 量化与部署公式速查 **49**

附录 B 最小 Agent-aware 评测伪代码 **50**

附录 C 最小量化与部署实验 **51**
C.1 与正文公式一致的参考量化器 51
C.2 仿射手算与实际字节的边界 52

C.3 在本地虚拟环境运行 52
C.4 工具调用：JSON、参数与首次分叉 53
C.5 逐步增加上下文的 KV 预算 53
C.6 眞实模型的部署记录与复现接口 54

附录 D 论文阅读地图 **55**
D.1 先建立对象与成本地图 55
D.2 按优化自由度阅读权重量化 55
D.3 先分清 KV 的位置选择与数值表示 55
D.4 把服务论文的计时边界对齐 55
D.5 用行为证据收束阅读 56
D.6 前沿论文与证据边界 56
D.7 综合练习与解题提示 56

参考文献与原始技术资料 **57**

插图

2.1 Agent 推理中的压缩对象。虚线表示上下文和运行时会反过来改变模型输入与缓存布局。 4

20.1 Agent 请求的时间线。只优化 decode kernel 可能无法改善工具等待主导的端到端时延。 35

表格

2.1 不同压缩对象的主要收益、风险与首选指标 5

4.1 量化粒度的研究与系统折中 9
4.2 同一教学向量的误差与存储；含 scale、零点及组尾填充，不含文件头。 9

6.1 压缩技术的组合关系 13

7.1 W8A8 路线的核心差异 14

9.1 极低 bit 权重方法的研究定位 18

13.1 KV token 选择方法的适用条件 23

15.1 Agent 语义压缩的可逆性设计 26

25.1 部署策略的选择逻辑 40

27.1 量化 Agent 实验的最低复现记录 43

30.1 从量化基础到 Agent 部署研究的十二周计划 47

A.1 常用公式 49

C.1 同一张量的仿射编码；组与组之间用竖线分隔 52
C.2 重建误差与理想打包账本是两个不同口径 52
C.3 本次合成实验结果；每配置 4 任务 × 5 种子 53

符号与缩写

符号/缩写	含义
W, A, KV	配置中的权重、激活与 key/value cache 类别；后接数字表示位宽。局部公式中的同名字母按该处定义。
X, W, Y	线性层采用行样本约定：$Y = XW$，形状分别为 $n \times d_{\text{in}}$、$d_{\text{in}} \times d_{\text{out}}$、$n \times d_{\text{out}}$。
s, z, q	量化比例因子、整数零点与码字；$\hat{x} = s(q - z)$ 为重建值。
B_{batch}, T_i, L H_{KV}, d_h	请求数、第 i 条请求保留的位置数、层数、KV 头数与每头维度；勿将 KV 头数替换为查询头数。
PTQ / QAT	post-training quantization / quantization-aware training。
W4A16	4 bit 权重、16 bit 激活，通常称 weight-only quantization。
W8A8 / W4A8	权重与激活都低于 FP16 的整型或浮点量化。
group size	一个 scale 共享的连续权重元素数量。
PPL	perplexity，语言建模损失的指数形式；不能代替 Agent 成功率。
TTFT / TPOT	time to first token / time per output token。
SLO	service-level objective，服务延迟、吞吐或可靠性目标。
ABP	action-boundary preservation，动作边界保持率。
pass^k	同一任务的一组 k 次运行全部成功的概率；在任务集上估计其比例。独立同分布且单次成功率为 p 时为 p^k。
pass@k	同一任务的一组 k 次运行至少一次成功的概率；独立同分布且单次成功率为 p 时为 $1 - (1 - p)^k$。
GB / GiB	10^9 / 2^{30} 字节；MB / MiB 相应为 10^6 / 2^{20} 字节。

第一部分

问题定义：为什么 Agent 压缩不同于普通 LLM 压缩

第一章

Agent 是一个受资源约束的闭环系统

1.1 从一次生成到长轨迹

普通语言模型推理可以粗略写为

$$y_{1:T} \sim p_\theta(y_{1:T} \mid x),$$

其中输入 x 固定，输出 y 在生成结束后再被评价。Agent 则在每一步把模型输出送入执行器，获得环境的新观测：

$$a_t \sim \pi_\theta(\cdot \mid g, o_{\le t}, a_{<t}, m_t, \tau_t), \qquad (s_{t+1}, o_{t+1}, r_t) \sim P(\cdot \mid s_t, a_t).$$

这里 g 是目标与约束，o_t 是观测，m_t 是可跨步骤保存的记忆，τ_t 是工具目录、schema、权限与预算。动作可能是普通文本，也可能是 JSON 工具调用、代码、鼠标点击或拒答。

定义

一个可部署的 Agent 至少包含：策略模型、上下文构造器、动作解析器、受权限约束的执行器、环境反馈、验证器和终止/升级策略。若只有一次模型调用、没有新的环境观测，也没有下一步决策，那么它更接近一个模型调用或固定 workflow，而不是闭环 Agent。

1.2 量化误差如何被闭环放大

设原模型给出的动作分布为 $p_t(a)$，量化模型给出的分布为 $\hat{p}_t(a)$。即使局部差异较小，也要单独测量长轨迹成功率。令 F_t 表示第 t 步发生不可恢复的致命错误，并定义条件风险 $\epsilon_t = \Pr(F_t \mid F_1^c, \dots, F_{t-1}^c)$。由概率链式法则，在“成功恰好等于全程无致命错误”的简化模型下，

$$\Pr(\text{整条轨迹成功}) = \prod_{t=1}^{T} (1 - \epsilon_t).$$

这里不需要各步独立；若把 ϵ_t 换成各步的无条件错误率，再相乘就需要额外假设。例如恒定条件风险为 0.02 时，5 步与 50 步的存活概率分别约为 0.904 与 0.364，只是教学算例。真实系统允许恢复、步骤数也可能随动作改变，因而该模型不能由 token KL 直接推得，应保存完整轨迹并检验最终状态。

直觉

量化普通问答时，错误常常是“这句话不够准确”；量化 Agent 时，错误可能是“把退款 API 的金额字段填成了负数”，随后触发不可逆副作用。任务级评测因此对少量关键 token 极其敏感。

1.3 四类资源预算

Agent 的部署预算至少有四个维度：

1. **存储预算**：模型权重、量化元数据、词表、适配器和缓存副本；
2. **计算预算**：矩阵乘、注意力、反量化、采样、检索和工具执行；
3. **状态预算**：KV cache、会话历史、长期记忆、工具结果和前缀缓存；

4. **风险预算**：越权、数据泄漏、错误副作用和人工审核次数。

低 bit 只直接减少其中一部分。若反量化占据了 kernel 的主要时间，显存下降不一定带来 TPOT 下降；若上下文和 KV 占据显存，压缩权重也不一定支持更大的并发。

1.4 Agent 压缩的目标函数

将压缩配置记为 $\mathbf{c}$，包括位宽、粒度、保留位置、kernel 和路由策略。一个包含任务与系统条件的目标是约束优化：

$$\begin{aligned} \max_{\mathbf{c}} \quad & U_{\text{task}}(\mathbf{c}) - \lambda_r R(\mathbf{c}) - \lambda_c C(\mathbf{c}) \\ \text{s.t.} \quad & M(\mathbf{c}) \le M_{\max}, \quad L_{p95}(\mathbf{c}) \le L_{\max}, \quad E(\mathbf{c}) \le E_{\max}, \end{aligned}$$

其中 U_{task} 是任务效用，R 是风险，C 是成本，M, L, E 分别是峰值内存、尾延迟和能耗。加权项需先说明归一化或换算单位；涉及权限和不可接受副作用的要求还应作为硬约束，不能只靠较高平均效用抵消。只最小化权重 MSE 是这个问题的一个局部代理，而不是最终目标。

本章抓住这些

Agent 压缩研究的第一原则：先写清楚状态、动作、环境反馈、成功判定和资源约束，再选择量化算法。没有这五项，所谓“Agent-aware”往往只是把一个普通 LLM benchmark 换成了带工具的 prompt。

第二章

压缩的对象：从参数到整个 Agent 状态

2.1 一张完整的对象地图

图 2.1 将 Agent 推理时会占用的对象分成模型侧、上下文侧和系统侧。研究时必须说明压缩发生在哪一层，以及是否改变了接口契约。

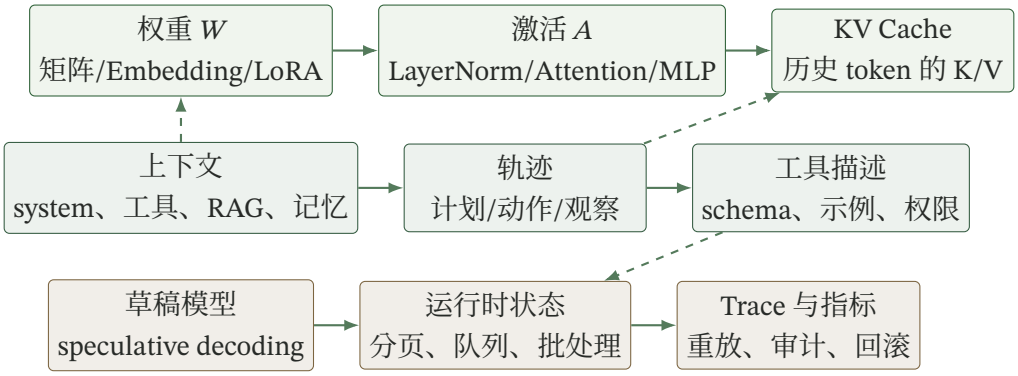


图 2.1 Agent 推理中的压缩对象。虚线表示上下文和运行时会反过来改变模型输入与缓存布局。

2.2 模型权重

权重通常是最容易离线压缩的对象。对一个线性层 $Y = XW$，权重从 FP16/BF16 变成 b bit 后，理论存储约从 $16|W|$ bit 降到 $b|W|$，但实际还要加上 scale、zero-point、码本、索引和异常值残差。权重量化通常不改变模型接口，因此适合先做 PTQ 基线；代价是它未必解决长上下文的 KV 瓶颈。

2.3 激活与中间状态

激活在每一层都产生，且随 batch 和序列长度变化。W8A8 可以降低矩阵乘输入的字节数；是否提高吞吐取决于设备的低精度算子、布局与反量化开销。激活离群值、动态范围和 scale 计算使它比仅量化权重更难。Agent 的工具结果、代码和图像 token 会改变激活分布，因此应检查通用文本校准集是否覆盖目标工作负载。

2.4 KV cache

自回归解码会保存每层每个历史 token 的 key 和 value。设 B_{batch} 条请求等长，每条有 T 个缓存位置，模型有 L 层、每层 H_{KV} 个 KV 头、每头维度 d_h。若 K/V 同用 b_{KV} bit，未共享、未分页时的数值载荷为

$$M_{\text{KV,payload}} = 2B_{\text{batch}} L H_{\text{KV}} T d_h \frac{b_{\text{KV}}}{8} \quad \text{字节.}$$

前面的 2 对应 K 和 V；分组查询注意力（GQA）与多查询注意力（MQA）共享 KV 头，不能把查询头数代入。实际占用还包括 scale、zero-point、高精度残留和 block 对齐，异长请求须逐请求求和，详见式 (12.1)。T 只统计实际进入相应注意力 cache 的位置；工具 schema、观察结果与多轮历史可

使它不断增加，多模态位置则取决于模型的编码和交叉注意力结构。

2.5 上下文、记忆与工具描述

把上下文压缩成摘要、把工具 schema 合并成模板、把检索文档做 chunk/重排或把历史观察转成结构化状态，都能减少 token 数；但这些操作不是数值量化，可能改变语义和权限。应把它们视为“语义压缩”，与数值量化分开测量：前者改变输入信息，后者近似保持算子功能。

2.6 训练状态与草稿模型

若目标是微调量化 Agent，还需要压缩梯度、optimizer states 和激活保存；QLoRA 通过冻结量化底座、训练低秩适配器降低显存。若目标是 speculative decoding，可用一个小的低 bit draft model 生成候选，再由高质量 target model 验证；此时压缩改变的不只是内存，还改变 draft acceptance rate 和工具调用边界。

表 2.1 不同压缩对象的主要收益、风险与首选指标

对象	主要收益	典型风险	首选指标
权重	常驻显存、带宽、模型装载时间	logits 漂移、低 bit 离群值、反量化开销	峰值显存、PPL、任务成功率、tokens/s
激活	GEMM 带宽与算力、能耗	动态范围、校准错位、算子不支持	TTFT/TPOT、kernel 利用率、激活误差
KV	长上下文显存、并发、decode 带宽	注意力失真、旧 token 被错误淘汰	上下文任务、KV bytes/token、pass^k
上下文/记忆	输入 token、prefill 时间、隐私暴露面	摘要遗漏、时序/权限错误	证据召回、记忆利用、引用正确率
草稿模型	speculative 接受率、平均解码步数	候选错误、工具边界不一致	acceptance rate、端到端 TPOT
运行时状态	碎片、重复计算、网络传输	状态错配、不可重放、租户泄漏	goodput、尾延迟、恢复成功率

第三章

从压缩率到任务效用：指标体系

3.1 不要把 PPL 当作 Agent 的充分统计量

PPL 测量的是在固定文本分布上预测下一个 token 的平均损失。它对常见文本很敏感，却可能忽略三个 Agent 关键点：工具名和参数是低频但高后果 token；终止/继续是一个离散边界；环境状态是否达到目标不能由字符串相似度判断。因此 PPL 应作为诊断指标，而不是唯一主指标。

3.2 五层指标

1. **张量层**：量化 MSE、最大误差、余弦相似度、激活范围、scale 分布；
2. **模型层**：PPL、困惑度差、校准损失、logit KL、校准误差；
3. **动作层**：工具选择准确率、函数名合法率、JSON 可解析率、参数 schema 通过率、停止/继续一致性；
4. **轨迹层**：步骤数、重试次数、循环率、恢复率、验证器触发率、证据链完整性；
5. **系统层**：峰值显存、TTFT、TPOT、端到端任务时延、吞吐、goodput、能耗、成本和安全违规。

3.3 动作边界保持率

在相同前置状态、工具目录和解码约束下，对每个决策位置，令原模型在动作集合 $\mathcal{A}$ 上的 top-1 选择为 a_t^*，量化模型选择为 $\hat{a}_t$。动作边界保持率定义为

$$\mathrm{ABP} = \frac{1}{N}\sum_{t=1}^{N} \mathbf{1}[a_t^* = \hat{a}_t],$$

但只在动作集合包含工具、参数、继续、完成和升级等离散决策时计算。若只比较所有 vocabulary token 的 top-1，指标会被普通文本 token 稀释。更细的做法是计算 margin：

$$\Delta_t = z_t(a_t^*) - \max_{a \neq a_t^*} z_t(a),$$

并优先保护 Δ_t 小、但后果严重的边界。这里的动作须先规范化：忽略 JSON 键顺序等无关差异，但保留工具名、参数值、权限和执行模式。若完整动作的概率无法枚举，应明确 margin 来自哪个工具名或参数决策头，不把局部 token margin 当成完整动作的置信度。ABP 衡量对参考模型的保持，不衡量动作本身是否正确；参考模型也可能犯错。真实 rollout 一旦前置状态分叉，后续动作不再处于同一条件下，不能继续作为同状态 ABP 的样本。

3.4 重复可靠性：pass^k 与 pass@k

对固定任务 i，令 $Y_{ij} \in \{0,1\}$ 为从同一初始快照开始、第 j 次完整运行是否成功，成功概率为 p_i。在给定任务、模型和环境版本后，各次运行独立且同分布时，

$$\mathrm{pass}_i^k = \Pr(Y_{i1} = \cdots = Y_{ik} = 1) = p_i^k, \qquad \mathrm{pass@}k_i = \Pr\Big(\max_{j \le k} Y_{ij} = 1\Big) = 1 - (1 - p_i)^k.$$

前者要求 k 次全部成功，反映重复可靠性；后者要求至少一次成功，反映允许多次尝试的覆盖能力。二者随 k 的变化方向相反，不能互换。对任务集作宏平均时，应先按任务计算再平均：$m^{-1}\sum_i p_i^k$ 通常不等于 $(m^{-1}\sum_i p_i)^k$。

若每任务共运行 $n_i \geq k$ 次、其中成功 c_i 次，在上述独立同分布假设下，无偏的组合估计量为

$$\widehat{\mathrm{pass}_i^k} = \frac{\binom{c_i}{k}}{\binom{n_i}{k}}, \qquad \widehat{\mathrm{pass@}k_i} = 1 - \frac{\binom{n_i - c_i}{k}}{\binom{n_i}{k}}.$$

约定 $a < k$ 时 $\binom{a}{k} = 0$。这相当于平均所有大小为 k 的运行子集，避免直接代入 $(c_i/n_i)^k$ 的有限样本偏差。若 $n_i = k$，估计量分别退化为“全部成功”和“至少一次成功”的单个指示量，信息很少；若 $n_i < k$，不能从这些运行估计该 k 的指标。

独立性必须来自实验设计：每次复位环境、清空会改变行为的跨运行记忆、独立抽取随机种子，固定工具版本与预算。重用同一结果、共享可变状态、服务漂移和相关采样都会破坏 p_i^k 的解释。此时可直接报告指定运行协议下观测到的成组可靠性，但不宣称它等于独立重复的概率。相同 seed 可用于 FP 与量化版本的配对；它不保证两种实现消耗相同的随机数，也不能替代多次运行。置信区间与按任务聚类的重采样见评测章节。

常见误区

不要把“量化模型回答与 FP16 相似”写成“量化模型与 FP16 等价”。等价至少要说明输入分布、随机种子、工具版本、环境状态、预算、停止规则和验证器。对不可逆工具动作，还应比较副作用而不只是文本。

第四章

量化的数学基础与数值格式

4.1 统一仿射量化

最常用的均匀仿射量化把实数 x 映射到整数 q：

$$q = \mathrm{clip}\left(\mathrm{round}\left(\frac{x}{s}\right) + z, q_{\min}, q_{\max}\right), \qquad \hat{x} = s(q - z),$$

其中 $s > 0$ 是比例因子（scale），整数 z 是零点（zero-point），表示实数零对应的编码。$q_{\min}, q_{\max}$ 是格式允许的整数范围。本文默认四舍六入、恰好在中点时取偶数（round-to-nearest, ties-to-even）；实际 kernel 若使用不同规则，必须单列。采用有符号编码的对称量化常令 $z = 0$，非对称仿射量化则允许零点偏移。

以无符号 b bit 编码 $q_{\min} = 0, q_{\max} = 2^b - 1$ 为例，先选含零的校准区间 $\alpha \le 0 \le \beta$、$\alpha < \beta$，再取

$$s = \frac{\beta - \alpha}{2^b - 1}, \qquad z = \mathrm{clip}\left(\mathrm{round}\left(-\frac{\alpha}{s}\right), 0, 2^b - 1\right).$$

一般编码范围的零点为 $\mathrm{round}(q_{\min} - \alpha/s)$，同样限制在编码范围内。零点取整后，实际可重建端点是 $s(q_{\min} - z)$ 与 $s(q_{\max} - z)$，未必恰等于 α, β。全正或全负组也要包含零，或明确采用另一种带偏置表示；全零组需显式规定 $s = 1, z = 0$ 等退化处理，不能除以零。区间越宽，普通值的步长通常越大；区间越窄，则更容易饱和，这构成截断校准的折中。

例如取 $b = 2, [\alpha, \beta] = [-2, 4]$，得到 $s = 2, z = 1$，重建格点为 $-2, 0, 2, 4$。输入 2.9 重建为 2，误差为 -0.9；输入 5 饱和到 4，误差为 -1。中点 $x = 3$ 按所写顺序先算 $\mathrm{round}(1.5) = 2$，再加零点，故重建为 4。若实现把零点放在取偶舍入内，$\mathrm{round}(1.5 + 1) = 2$，反会重建为 2：中点处不仅要固定舍入规则，还要固定算子顺序。

4.2 粒度：per-tensor 到 per-token

同一个 scale 作用于整个张量时是 per-tensor，作用于输出通道、输入通道、权重 group、attention head 或单个 token 时分别称 per-channel、per-group、per-head、per-token。粒度越细通常误差越小，但 metadata、访存和 kernel 分支越多。

4.3 手算分组误差与真实字节数

考虑八个教学权重 $w = (0, 1, 2, 3, 0, 1, 2, 9)$，按连续元素分组。为了让每个 scale 都可精确写出，这里比较 2 bit 与 3 bit 的窄范围对称量化：$q \in \{-Q, \dots, Q\}$、$Q = 2^{b-1} - 1$、$z = 0$、$s = \max|w|/Q$；该编码留一个码字不用。2 bit、组长 8 时只有一个 $s = 9$，重建为

$$\hat{w}_{2,8} = (0, 0, 0, 0, 0, 0, 0, 9).$$

改为组长 4 后，两组的 scale 分别为 3 与 9，重建为

$$\hat{w}_{2,4} = (0, 0, 3, 3, 0, 0, 0, 9).$$

平方误差和由 19 降为 7，但多存了一份元数据。3 bit、组长 4 时 scale 为 1 与 3，前四个数可精确重建，后四个数变为 $(0, 0, 3, 9)$，平方误差和为 2。

表 4.1 量化粒度的研究与系统折中

粒度	优点	代价与适用边界
per-tensor	metadata 少、kernel 简单	范围差异大时误差高，适合作为基线
per-channel	处理输出通道尺度差异	需要按通道广播，常用于权重和激活迁移
per-group	W4A16 常见，兼顾误差与存储	group size 影响 metadata 与 kernel 对齐
per-head	能利用 attention head 的异质性	head 级索引和调度更复杂
per-token	适合动态激活、KV value	在线求 scale，可能增加 decode 开销
混合粒度	在敏感位置使用更多自由度	需要记录布局，部署复现最困难

为明确计算存储，约定此教学文件的每组保存一个 FP16 scale 和一个 INT8 zero-point，载荷在组内紧密打包、每组尾部补到整字节。以上 scale 均为整数，FP16 可精确表示。若组数为 G、第 g 组有 n_g 个权重，则

$$S = \sum_{g=1}^{G} \left\lceil \frac{n_g b}{8} \right\rceil + 3G \quad \text{字节}, \qquad b_{\text{eff}} = \frac{8S}{N}, \quad N = 8.$$

表 4.2 同时计算重建 MSE 和字节数。

表 4.2 同一教学向量的误差与存储；含 scale、零点及组尾填充，不含文件头。

位宽	组长	MSE	载荷/B	元数据/B	总计/B	有效 bit/权重
2	8	19/8	2	3	5	5
2	4	7/8	2	6	8	8
3	8	1/2	3	3	6	6
3	4	1/4	4	6	10	10

3 bit、组长 4 的两组载荷各需 $\lceil 12/8 \rceil = 2$ 字节，不能把整段 24 bit 直接算成 3 字节。生产对称格式可省去恒为零的 zero-point，另一些格式则需要更大的对齐块、残差或码本；应按实际布局重算，不能套用本例头部。

再把 2 bit、组长 8 的范围人为截到 $[-3, 3]$，重建为 $(0, 0, 3, 3, 0, 0, 3, 3)$。单个离群值 $9 \to 3$ 产生截断平方误差 36，其余舍入平方误差为 4，总 MSE 为 $40/8 = 5$，反而差于不截断。一般有 $\hat{x} - x = (\hat{x} - x_{\text{clip}}) + (x_{\text{clip}} - x)$，平方后还含交叉项，本例恰为零。截断阈值需要校准，不能仅凭“保护多数普通值”断言总误差更低。

整模账本还要加上未量化的 embedding、归一化参数、输出头、适配器、配置和对齐；运行时则另加激活与 KV。标称权重位宽、含元数据的平均位宽与模型文件或进程峰值内存是三个不同量。

4.4 误差不止是 MSE

设校准输入 $X \in \mathbb{R}^{n\times d_{\text{in}}}$、权重 $W \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$，全书线性层采用 $Y = XW$ 的行样本约定。常见量化目标包括层输出均方误差与模型分布 KL：

$$\mathcal{L}_{\text{MSE}} = \frac{\|XW - X\hat{W}\|_F^2}{nd_{\text{out}}}, \qquad \mathcal{L}_{\text{KL}} = \mathrm{D}_{\text{KL}}\left(p_\theta(\cdot \mid x) \,\|\, p_{\hat{\theta}}(\cdot \mid x)\right),$$

还可以使用带曲率权重的近似目标。对于 Agent，可定义动作集合上的 KL、工具参数交叉熵和停止决策损失；JSON 可解析性或环境验证器通常是离散指标，若纳入梯度训练，需要说明可微代理、策略梯度或外层配置搜索，不能直接把其分数当作可导张量。一个候选多项目标为

$$\mathcal{L} = \lambda_w\mathcal{L}_w + \lambda_a\mathcal{L}_a + \lambda_{\text{logit}}\mathcal{L}_{\text{KL}} + \lambda_{\text{act}}\mathcal{L}_{\text{tool}} + \lambda_{\text{stop}}\mathcal{L}_{\text{stop}}.$$

这里 $\mathcal{L}_w, \mathcal{L}_a$ 分别表示归一化权重与激活重建误差，$\mathcal{L}_{\text{tool}}, \mathcal{L}_{\text{stop}}$ 是工具及停止决策监督损失；各项的样本、归一化和可导代理都须明示。权重系数既要考虑损失量级，也要反映任务后果，并通过留出验证选择。

4.5 INT、FP 与 NF 格式

INT8/INT4 使用整数格点与显式 scale；FP8/FP4 在指数与尾数之间分配有限码字，具体动态范围取决于格式、共享 scale 和特殊值约定，不能一概比缩放整数更宽。NF4 是针对近似正态分布设计的非均匀码本，并非 IEEE 浮点格式。格式不是算法的同义词：同样是 4 bit，group size、scale 类型、累加精度、布局和 kernel 可能导致不同的延迟与精度。

研究者视角

前沿研究者应把“bit-width”写成一个完整的配置元组：

$$\mathbf{c} = (b,\ \text{format},\ \text{granularity},\ \text{group size},\ \text{scale precision},\ \text{accumulator},\ \text{kernel}).$$

只报告“4 bit”无法复现实验。

第五章

PTQ、QAT 与校准数据

5.1 Post-Training Quantization

训练后量化（PTQ）从已经训练好的浮点 checkpoint 出发，在不重新进行全模型训练的条件下选择量化表示。它可以完全不读数据，也可以使用少量校准样本估计 scale、离群值和舍入决策。GPTQ、AWQ、SmoothQuant 以及学习少量校准参数的 OmniQuant 都属于这一范围。PTQ 不等于“没有优化”：二阶误差补偿、scale 搜索和可学习截断的计算成本不同，比较时要单列校准 token、优化步数、耗时和设备。

5.2 Quantization-Aware Training

量化感知训练（QAT）在训练前向中模拟部署量化，并更新允许训练的模型参数。伪量化通常先量化再反量化，矩阵乘仍可使用浮点 kernel；是否量化梯度是另一个选择，并非 QAT 的定义。对不可导舍入，直通估计器（STE）常用近似为

$$\frac{\partial \hat{x}}{\partial x} \approx \mathbf{1}_{x \in [\alpha, \beta]},$$

或在指定范围内把量化视作恒等映射。STE 是有偏的代理梯度，训练后的精度改善必须通过对照验证。训练可使用浮点伪量化，但最终部署必须与目标低比特 kernel 的舍入、截断、粒度和累加规则匹配；否则训练模拟的算子与实际算子不同。对于 Agent，训练样本还应覆盖工具结果和失败恢复轨迹。

5.3 校准集的四个层次

1. **语言校准**：WikiText、C4 或领域文本，用于基础分布；
2. **任务校准**：用户指令、代码、数学、长上下文；
3. **工具校准**：工具名、参数 JSON、schema、错误返回和重试；
4. **环境校准**：真实网页/桌面截图、RAG 证据、记忆读写和多轮 observation。

校准样本应记录 token role、位置、模态、工具风险和后置验证结果。否则混合精度算法会把最宝贵的 bit 分给“平均文本”，而不是工具决策边界。

5.4 数据泄漏与校准集过拟合

把测试答案、测试轨迹或专属后置状态用于选 scale、位宽或超参数会污染评测。公开工具 schema 若在真实部署时本来可见，可以同时提供给校准与测试；这属于同工具分布，不能宣称未见工具泛化。应按部署目标划分校准、验证和测试集，并分别报告同工具新任务、未见工具、未见用户措辞及布局变化。校准集的标签与筛选结果也属于训练信息，不能用最终测试结果反复选择量化配置。

第六章

常见压缩技术的全景

量化是本书主线，但前沿 AI 科学家还应理解与量化互补的压缩方法。它们改变的对象和错误形态不同，往往可以组合。

6.1 剪枝与稀疏化

非结构化剪枝删除单个权重，结构化剪枝删除 head、channel、layer 或 token。前者压缩率高但需要稀疏 kernel 才能加速，后者更容易获得眞实吞吐收益却可能损失容量。Agent 中可以剪枝低价值工具描述、重复 observation 或注意力 token，但必须用 goal checker 验证不能删掉关键证据。

6.2 低秩分解与适配器

对 $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$，用 $U \in \mathbb{R}^{d_{\text{in}} \times r}$、$V \in \mathbb{R}^{d_{\text{out}} \times r}$ 近似 $W \approx UV^{\top}$。仅当 $r(d_{\text{in}} + d_{\text{out}}) < d_{\text{in}} d_{\text{out}}$ 且保眞足够时，才减少参数和相应的理论乘法量。LoRA 冻结底座并增加低秩更新，本身不会删除底座参数。将低秩项用作量化残差补偿是一种组合方式；它能否修复敏感方向取决于残差结构和训练信号，还要支付额外矩阵乘与适配器管理成本。

6.3 知识蒸馏

教师模型提供 soft logits、隐藏状态、工具轨迹或验证反馈，学生模型学习

$$\mathcal{L}_{\text{KD}} = \tau_{\text{KD}}^2 \, \mathrm{D}_{\text{KL}} \big(\operatorname{softmax}(z_T/\tau_{\text{KD}}) \,\|\, \operatorname{softmax}(z_S/\tau_{\text{KD}}) \big).$$

其中 z_T, z_S 是教师和学生 logits，$\tau_{\text{KD}} > 0$ 是蒸馏温度，区别于轨迹长度 T；两者词表和事件语义需要对齐。对 Agent，蒸馏目标可以是完整的 Thought–Action–Observation 轨迹，也可以只蒸馏动作和验证结果，避免把不可靠的自由文本思维当作眞值。

6.4 架构级缩小、MoE 与动态路由

减少层数、隐藏维度或词表是直接改变模型容量的路线；稀疏专家模型（MoE）让每个 token 只激活部分专家，其活跃计算可小于使用全部专家的稠密方案，但总权重和通信仍需预算。Agent 也可先用小路由器筛选工具，再调用主模型。该方案可能减少 schema 上下文，却会新增一次路由开销和漏选风险，收益须在保持工具召回的条件下测量。

6.5 上下文压缩与语义记忆

摘要、检索、去重和结构化状态可以减少输入位置数，同时可能改变事实、时序和权限。建议把这些语义压缩视为有损编码器，保留来源与可恢复原文。精确前缀复用另属计算和存储复用：它跳过相同前缀的重复 prefill，并可共享物理 KV block，但不缩短每条请求的逻辑上下文，单请求 decode 仍须访问相关历史。

表 6.1 压缩技术的组合关系

技术	主要对象	可组合方式	典型新增问题
量化	权重/激活/KV	与剪枝、低秩、蒸馏、kernel 联合	离散误差、scale、反量化
剪枝	权重、head、token、工具	先剪枝再量化或动态淘汰 KV	结构损失、稀疏 kernel
低秩	权重残差/adapter	作为量化误差补偿或任务适配	额外算子、rank 选择
蒸馏	logits/轨迹/隐藏状态	训练小模型或校准量化模型	教师偏差、数据成本
上下文压缩	历史/文档/工具描述	与 KV 量化、prefix cache 联合	事实遗漏、权限错配
动态路由	专家/工具/模型级联	与 draft model、early exit 联合	路由错误、尾延迟

自检与练习

任选一个 7B–14B 指令模型，列出其推理过程中的所有可压缩对象，并为每一项写出：张量形状、生命周期、读写频率、失真度量、真实硬件收益和失败后果。若某项无法写出后置验证条件，说明你还没有定义清楚它的压缩问题。

第七章

W8A8：激活离群值与等价变换

7.1 LLM.int8：为什么八 bit 也不简单

LLM.int8[1] 观察到，大模型激活中少量大幅值特征会撑大量化范围，使普通值可用的格点减少、步长增大。它将矩阵乘拆成高精度 outlier 路径与 INT8 普通路径，只为少量异常维度付出额外代价。异常维度不一定对应异常 token；工具名、数字、代码标点和结构化参数的通道统计可能不同，应按 workload 重新估计 outlier 分布。

7.2 SmoothQuant：把困难从激活迁移到权重

对于 $X \in \mathbb{R}^{n\times d_{\text{in}}}$、$W \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$，正对角矩阵 $S = \operatorname{diag}(s_1, \ldots, s_{d_{\text{in}}})$ 给出恒等式 $Y = XW = (XS^{-1})(SW)$。缩小激活范围的尺度被吸收到权重中；SmoothQuant[2] 用 α 平衡两侧的量化困难：

$$s_j = \frac{\max |X_{:,j}|^{\alpha}}{\max |W_{j,:}|^{1-\alpha}}.$$

最大值分别沿校准样本和对应权重行计算，零范围通道须设正的数值下界。α 应在校准集上与硬件 kernel 联合选择，并非越大越好。误差来自后续量化和有限精度计算；若缩放跨越非线性层，还须满足网络的融合条件。

7.3 RPTQ 与通道重排

RPTQ[3] 按统计范围聚类、重排并量化通道，减少组内动态范围差异。重排必须在前后层等价抵消，否则会改变函数；实现时还须检查 fused attention、tensor parallel 切分和权重存储布局。

表 7.1 W8A8 路线的核心差异

方法	主要机制	Agent 部署时要额外检查
LLM.int8	outlier-aware 混合精度矩阵乘	异常比例、混合 kernel、工具/代码分布
SmoothQuant	等价缩放激活与权重	融合兼容性、校准覆盖
RPTQ	通道聚类与重排	切分与加载布局、跨卡通信

选择条件。设备有匹配 INT8 矩阵乘，且计算或输入带宽受限时，可考虑 W8A8。W4A16 主要减少权重载荷；W8A8 还降低矩阵乘输入精度，但增加激活 scale 与转换工作，须按 batch、输入长度和算子覆盖率实测。KV 或重复前缀受限时，另做 KV 压缩、前缀复用对照，不能仅凭位宽判断速度。

第八章

W4A16：GPTQ、AWQ 与 OmniQuant

8.1 GPTQ：用二阶信息做逐层舍入

GPTQ[4] 将一层量化视为在校准输入上最小化输出误差。本书使用 $X \in \mathbb{R}^{n\times d_{\text{in}}}$、$W \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$ 和 $Y = XW$，因此目标为

$$\min_{\hat{W}\in\mathcal{Q}} \frac{1}{2}\|XW - X\hat{W}\|_F^2.$$

$\mathcal{Q}$ 是指定码本、分组与位宽允许的候选权重集合。令 $e_r = W_{:,r} - \hat{W}_{:,r}$ 表示第 r 个输出通道的误差向量，则目标等于 $\frac{1}{2}\sum_r e_r^\top H e_r$，其中 $H = X^\top X \in \mathbb{R}^{d_{\text{in}}\times d_{\text{in}}}$。对这个层输出平方目标，$H$ 是精确 Hessian；它只是全模型语言损失曲率的代理。若目标不带 1/2，Hessian 为 $2X^\top X$，整体常数不改变最优解。

一次固定第 i 个输入方向的量化值 $Q(W_{i,:})$，把舍入残差记作 $e_i = W_{i,:} - Q(W_{i,:})$。令 F 为当前尚未固定的输入坐标集合，$\lambda > 0$，并记 $A = (H_{FF} + \lambda I)^{-1}$，则最小化局部阻尼二次误差的补偿形式为

$$W_{j,:} \leftarrow W_{j,:} - \frac{A_{ji}}{A_{ii}} e_i, \qquad j \text{ 尚未固定.}$$

在本书的矩阵方向下，量化与补偿沿 W 的输入**行**推进，各输出通道可并行；原论文常用转置后的权重布局，因而称逐列处理。阻尼避免校准样本不足时的奇异矩阵。实际 GPTQ 使用分块、延迟更新与逆 Hessian 的 Cholesky 因子降低开销；3–4 bit 的质量仍依赖模型、分组、校准与实现，并非该二次目标对最终任务的保证。

GPTQ 的研究价值不只在算法本身：它把“哪个权重更重要”从绝对数值改写成“在真实输入协方差下哪个误差更昂贵”。这正是把校准输入换成 Agent trace 的理论入口。

8.2 AWQ：保护 activation-aware 的显著通道

AWQ[5] 用激活统计识别对输出较敏感的权重输入通道。其核心做法是在 $XW = (XS^{-1})(SW)$ 中搜索逐通道 scale，再量化缩放后的权重；放大的显著通道在反向缩放后承受较小的有效舍入误差。它的硬件友好实现仍把权重放在统一低 bit 表示中，**并不默认将显著权重单独存成 FP16**。论文用保留少量高精度权重说明敏感性，但这不应与最终的缩放机制混为一谈。校准成本主要来自激活统计和 scale/截断搜索，需报告搜索范围与 group size。

8.3 OmniQuant：学习 clipping 与等价变换

OmniQuant[6] 在冻结原始模型权重的基础上，学习权重截断与等价变换参数，以分块重建误差进行校准。这不同于用下游目标更新全部模型权重的 QAT；“PTQ”标签也不表示零梯度或零优化成本。是否改善极低 bit 质量需绑定具体 W/A 配置，校准 token、优化轮数、耗时和显存必须一起报告。

三者都可减小层输出误差，但自由度不同：GPTQ 补偿舍入误差，AWQ 搜索缩放和截断，OmniQuant 优化可学习校准参数。统一核验浮点模型、目标格式与数据，再比较这些自由度，才能把数值收益与优化成本联系起来。

8.4 三种方法的实验读法

1. 固定同一个 FP checkpoint、tokenizer、校准集、group size 和 kernel；
2. 分别报告权重文件大小、加载时间和真正的端到端延迟；
3. 在普通文本、代码、JSON 工具调用和长上下文上分层测试；
4. 对 Agent 记录工具选择、参数 schema、停止决策与最终环境状态；
5. 以相同的重试和预算比较，而不是让某个量化模型拥有更多恢复机会。

研究者视角

一个常见但危险的结论是“AWQ 比 GPTQ 更适合 Agent”。严谨说法应是：在给定模型、工具轨迹、group size、kernel、硬件和预算下，AWQ 在某个任务集合上保持了更高的动作边界或更低的 TPOT。方法名本身不能替代实验条件。

第九章

2–3 bit 权重：极限压缩的路线与代价

9.1 SpQR 与稠密 – 稀疏混合

SpQR[7] 把大部分权重压到低 bit，并用稀疏高精度结构保存量化敏感的异常项，以减小指定模型与评测上的质量损失。它说明可以用额外元数据保护少数高代价误差；异常项、两级量化元数据和索引均应计入有效位宽。稀疏解码与随机访存还可能削弱 kernel 收益，“接近无损”不能脱离论文的模型和任务范围解释。

9.2 SqueezeLLM：非均匀码本与 dense/sparse 分解

SqueezeLLM[8] 将敏感权重与普通权重区分处理，并使用非均匀量化或稀疏分量减少误差。对于工具调用，敏感项可能不仅由权重数值决定，还与某些输出 logit、工具名 token 和 JSON 括号相关，因此可探索任务加权的异常值定义。

9.3 QuIP 与 QuIP#：不可相干化和格码本

QuIP[9] 对权重与曲率代理进行不可相干化预处理，使量化困难不集中于少数坐标，再使用曲率感知舍入。其理论保证针对满足相应条件的二次代理误差，不能推广为任意 2 bit 模型的任务成功保证。QuIP#[10] 使用随机 Hadamard 变换和格码本等设计改善低 bit 表示。预处理、码本查找与局部优化均有成本；部署时需确认它们如何实现或融合，并报告真实有效位宽。

9.4 AQLM 与 additive quantization

加性量化把一个向量表示为多个码本向量之和：

$$w \approx c_{1,i_1} + c_{2,i_2} + \cdots + c_{m,i_m}.$$

多个小索引可以表达比单一标量格点更丰富的向量。若每个 d_v 维分组使用 m 个、每个含 K 项的码本，索引载荷约为 $m \log_2 K/d_v$ bit/权重，还要加码本和 scale。AQLM[11] 将码本学习、索引搜索及重建优化结合；因此“2 bit”应先确认是索引载荷还是含全部开销的平均位宽。码本训练、查找和反量化也可能成为成本来源。

9.5 QuaRot、SpinQuant 与离群值消除

QuaRot[12] 通过固定正交变换分散通道离群值，为 W/A/KV 低精度计算创造更平衡的范围；SpinQuant[13] 则优化可学习旋转。若 R 为正交矩阵，$R^\top R = I$，在线性层输入轴上有

$$XW = X(RR^\top)W = (XR)(R^\top W).$$

实数算术中，这个恒等变换不改变输出或动作 margin；变化来自量化后的误差与有限精度实现。它也不能任意穿过 GELU 等非线性。实际网络必须按残差、归一化和注意力结构安排可融合与在线变换，而不是在任意相邻层插入旋转。

RoPE 是随位置变化的旋转，更需分清变换顺序。用列向量记位置 t 的查询、位置 j 的键为 q_t, k_j，RoPE 后点积为

$$(R_t q_t)^\top (R_j k_j) = q_t^\top R_t^\top R_j k_j.$$

对 RoPE 后的两者施加同一正交变换可保持点积；任意把一个公共旋转移到 RoPE 前，却要求它与不同的 R_t 交换，通常不成立。正交性保持二范数，不保持每通道最大值，因此前后两种量化统计不同。Agent 评测应检验完整变换再量化后的工具 margin 与任务可靠性，并核实 RoPE、GQA 和投影实现的兼容性。

9.6 BitNet 与训练时低比特

BitNet b1.58[14] 从训练阶段使用 $\{-1, 0, 1\}$ 三值权重与相应缩放；$\log_2 3 \approx 1.585$ 是三种状态的编码信息量，不等于任意文件或 kernel 恰好每权重占 1.58 bit。训练仍可维护高精度潜在权重和优化器状态，并非训练显存也自动降到该位宽。它与直接量化现有 checkpoint 的 PTQ 是不同获得路线，比较时要计入训练成本。将其用于 Agent，还需验证工具反馈、恢复和拒绝行为，不能由通用语言能力推断动作策略保真。

表 9.1 极低 bit 权重方法的研究定位

路线	主要思想	需要警惕的系统问题
稠密 – 稀疏	普通权重低 bit，异常值高精度稀疏保存	索引访存、稀疏 kernel、metadata
不可相干化	旋转/预处理让误差更均匀	旋转融合、位置编码、跨层布局
格/加性码本	用向量码本表达方向	查表、反量化、并行度、码本缓存
训练时低 bit	训练目标与硬件格式共同设计	重新训练成本、模型迁移和生态支持

第十章

量化后补偿：QAT、QLoRA 与低秩修复

10.1 LLM-QAT 与数据自由训练

LLM-QAT[15] 在缺少原始训练数据时，用教师模型生成或重构信号做量化感知训练；一般目标包括 teacher logits、隐藏状态或层输出的一致性。数据自由仍有成本，合成分布、工具调用覆盖与教师偏差决定 Agent 是否受益。

10.2 QLoRA：冻结量化底座，训练低秩适配器

QLoRA[16] 以 NF4 和双重量化等机制存储冻结底座，计算时反量化到计算 dtype，只更新低秩适配器。梯度经过冻结权重的计算路径，不更新量化码字，前向与反向也未必采用原生 INT4 kernel。分页优化器通过内存迁移缓解训练尖峰，效果依赖设备和负载。

沿用 XW 约定，令 $B \in \mathbb{R}^{d_{\text{in}} \times r}$、$A \in \mathbb{R}^{r \times d_{\text{out}}}$，更新为 $\Delta W = BA$。底座误差 $E_q = W - \hat{W}$ 提示了修复目标 $X(W - E_q + BA) \approx XW$，但 QLoRA 通常直接优化下游任务损失，并不默认求解浮点重建。低秩项不能补偿任意高秩误差，也不保证保留所有未训练任务。

10.3 LoftQ：让量化与 LoRA 初始化相互适配

通常 LoRA 将一个因子随机初始化、另一个置零，初始更新为零，不能主动修复底座量化误差。LoftQ[17] 无需校准输入，在权重目标上联合初始化量化底座与低秩因子：

$$\min_{\hat{W} \in \mathcal{Q},\, B,A} \|W - \hat{W} - BA\|_F^2, \qquad \operatorname{rank}(BA) \le r.$$

交替步骤先量化 $W - B^{(t-1)}A^{(t-1)}$，再对残差 $W - \hat{W}^{(t)}$ 做截断 SVD，更新秩不超过 r 的低秩项。$\mathcal{Q}$ 为指定格式允许的量化权重集合。该初始化不保证全局最优，也不直接优化 Agent 成功率。

初始化后可用工具轨迹微调，以环境 verifier 检查动作可执行性。将轨迹激活引入加权层输出重建，是待验证的扩展，不能当作 LoftQ 原算法的数据要求。

10.4 蒸馏信号与训练投入

量化补偿可蒸馏四类信号：

1. 全 vocabulary logits：保留语言分布，但计算和存储成本高；
2. 动作集合 logits：只保留工具、停止、升级等决策边界；
3. 隐藏状态或层输出：适合逐层校准，但不保证行为等价；
4. 环境反馈与 verifier：直接优化任务后置状态，最接近部署目标但数据昂贵。

可先用层输出稳定训练，再用动作与环境信号适配；须与直接任务微调、不微调基线比较，并检查未见任务的能力保留。W4A16 已满足任务和延迟时，QAT 收益可能不抵成本；W4A8、KV2、移动端整数 kernel 或特定工具域更值得尝试。评测须报告训练数据、步数、教师调用成本与未见任务泛化。

第十一章

激活量化：动态范围、校准与注意力路径

11.1 激活为什么比权重难

权重是静态的，可以离线估计 scale；激活依赖输入 token、batch、位置和前文状态。Agent 的输入分布还会随工具结果、网页、代码和图像变化。动态 per-token scale 需要在线统计，并可能增加张量并行中的同步；其误差和运行成本均须与静态 scale 对照。

11.2 Attention 中的不同张量

注意力路径至少包含 Q, K, V、缩放点积分数、softmax 概率和输出投影。Q/K 的量化误差先改变点积，再经 softmax 改变注意力分配；不能笼统断言 softmax 必然放大误差，其影响取决于输入尺度、分数间隔和数值实现。V 的误差直接改变加权内容。RoPE 保持向量二范数，但会混合通道、改变逐通道范围，所以前后统计不可混用。各张量还可能使用不同的计算、存储和累加精度，应逐项记录。

11.3 混合精度分配

设第 l 层第 h 个 head 使用 bit $b_{l,h}$，在总预算 B 下可写成

$$\min_{b_{l,h}} \sum_{l,h} D_{l,h}(b_{l,h}) \quad \text{s.t.} \quad \sum_{l,h} \mathrm{bytes}_{l,h}(b_{l,h}) \le B.$$

$D_{l,h}$ 可来自校准输出误差、动作 KL 或 attention 权重变化；相加是假定各块失真可近似分离，联合低精度时可能存在交互。分配器可以使用动态规划、拉格朗日松弛或贪心敏感度，但候选位宽必须受 kernel 支持约束，字节数也应含元数据和对齐。最终配置必须联合重放验证，不能把逐块测得的收益直接相加。

自检与练习

对一个工具调用样本，分别计算普通文本 token、工具名 token、JSON 数字、工具返回错误和最终 stop token 的 activation max/percentile。比较用全局 scale、per-channel scale 和按角色 scale 时的饱和率，并把结果与工具参数合法率联系起来。

第十二章

KV Cache：Agent 长轨迹的第二个模型

12.1 从注意力公式理解 cache

对每头宽度为 d_h 的一个查询头，在第 t 步，因果注意力可写成

$$\mathrm{Attn}(q_t, K_{\le t}, V_{\le t}) = \mathrm{softmax}\left(\frac{q_t K_{\le t}^{\top}}{\sqrt{d_h}}\right) V_{\le t}.$$

因果注意力中，历史 K, V 不依赖未来 token，可在相同模型、前缀和位置规则下复用。Prefill 生成前缀 KV，decode 计算新 token 的 Q/K/V 并读取历史。工具结果可新增一段 prefill；保留会话 cache 时位置数持续增长，重建 prompt 时则可能重算或复用前缀，不必永久沿用同一个 cache。

12.2 显存、带宽和并发的三重约束

设 B_{batch} 条请求各保留 T_i 个位置，模型有 L 层、每层 H_{KV} 个 KV 头、每头宽 d_h。各层同形状、K/V 同位宽且不共享时，

$$\begin{aligned} M_{\text{KV,payload}} &= \frac{2LH_{\text{KV}}d_h b_{\text{KV}}}{8} \sum_{i=1}^{B_{\text{batch}}} T_i, \\ M_{\text{KV,allocated}} &= M_{\text{KV,payload}} + M_{\text{meta}} + M_{\text{residual}} + M_{\text{padding}}. \end{aligned} \tag{12.1}$$

各项以字节计，b_{KV} 以 bit/元素计。高精度区的 M_{residual} 只计相对低 bit 载荷的增量；K/V 异位宽时用 $b_K + b_V$ 替代 $2b_{\text{KV}}$，层形状或滑窗长度不同时逐层求和。GQA 的查询头数可远大于 KV 头数，MQA 每层只有一个 KV 头；这是架构属性，部署时直接删头不保持原函数。

例如 $L = 32, H_{\text{KV}} = 8, d_h = 128, B_{\text{batch}} = 1, T = 8192$：FP16 载荷为 1 GiB $= 2^{30}$ 字节，四条同长请求为 4 GiB；INT4 单请求载荷为 256 MiB。每 64 个元素另存 FP16 scale 与 8 bit 零点需 24 MiB，共 280 MiB。这是未计高精度增量和分页的形状算例，并非显存实测。

分页按物理 block 向上取整，共享前缀的相同物理块只计一次，offload 分列主机与设备字节；预留池和临时缓冲另计，因此理论载荷不能代替实测峰值。

全注意力 decode 每步访问历史 KV，带宽可能限制 TPOT。请求共享权重，通常各持后缀 cache；量化减小每位置字节，淘汰减少位置，分页改善分配与共享，offload 改变存放位置，收益须分别记账。

12.3 KV 的错误传播

若 q 和 K 的扰动分别为 $\delta q, \delta K$，点积的精确差为

$$\delta(qK^{\top}) = q(\delta K)^{\top} + \delta q K^{\top} + \delta q(\delta K)^{\top}.$$

仅量化 K 时 $\delta q = 0$，只剩首项，分数再除以 $\sqrt{d_h}$。若 $p = \mathrm{softmax}(z)$，一阶变化为 $\delta p_i \approx p_i(\delta z_i - \sum_j p_j \delta z_j)$，故 K 误差的影响取决于查询与原分布；V 扰动改变 pV 的内容。指令、工具证据和填充的敏感性须以固定轨迹验证，角色标签不能保证某位置可删或可低精度保存。

第十三章

KV Token 选择：H2O、Scissorhands、SnapKV 与 PyramidKV

KV token 选择删除部分可访问的历史位置；KV 量化则近似保存保留下来的 K/V 数值。二者可以组合，但必须分别记录保留位置数与每位置实际字节。它们也不同于将原文改写为摘要：摘要改变 token 序列，通常需要重新 prefill，而不是把原 KV 任意求和后当作摘要。

13.1 H2O 与重要 token

H_2O[18] 联合保留近期 token 与累计 attention 贡献较大的 heavy hitters，在预算不足时淘汰其他位置。它把“保存所有历史”改成“保存可能影响未来注意力的部分历史”。对 Agent，累计 attention 不等于任务价值：一条当前权重低、但后续动作需要引用的工具错误信息可能仍须保留。

13.2 Scissorhands：重要性持久性

Scissorhands[19] 假设在早期步骤重要的 token 往往在后续仍有价值，因此可用历史重要性预测未来保留集合。它减少了每一步重新估计所有 token 的成本，但在上下文主题突变、工具切换和长时间间隔下需要重新校准。

13.3 SnapKV：生成前预测 head 的保留位置

SnapKV[20] 利用 prompt 末端 observation window 的 attention 模式，在生成前预测每个 head 应保留的历史位置。它适合问题 – 文档式长上下文，但 Agent 的末端可能是工具返回、错误堆栈或用户追问，未必代表所有未来动作的查询模式。

13.4 PyramidKV：按层分配预算

PyramidKV[21] 根据层间注意力聚合的差异，为不同层配置不同 cache 容量。它支持“各层不必相同预算”的设计，但不直接证明按 Agent 角色分配位宽有效。将层/头的保留数与 token 角色位宽联合分配，是需要单独消融和重放检验的扩展问题。

表 13.1 KV token 选择方法的适用条件

方法	核心信号	Agent 场景中的失效模式
H_2O	近期位置与累计 heavy hitter	低 attention 但高后果的工具证据被删
Scissorhands	重要性持久性	任务切换或主题突变时历史先验失效
SnapKV	末端窗口预测未来 head 需求	observation 末端不代表未来多工具查询
PyramidKV	层间漏斗与动态预算	层预算与量化 bit 的联合搜索复杂

13.5 保留、摘要与删除的组合

可以把“高风险原样保留、中风险量化、低风险摘要或删除”作为候选策略，但风险分组及其收益必须通过成对任务验证。淘汰后应保留原 position ID 和可访问位置的映射，不能把剩余 token 从零重新编号而继续沿用旧 RoPE 键。保留原始文本也不等于可以直接恢复某个旧 KV：恢复可能需要原前缀、模型版本、位置规则和压缩配置重新计算。因此应把冷存储重建的延迟和恢复后的成功率纳入预算，而非仅声明“可撤销”。

第十四章

KV Quantization：KIVI、KVQuant 与 GEAR

14.1 为什么 K 与 V 需要不同轴

KIVI[22] 对 K/V 使用不同量化轴。对单层单头的 $T \times d_h$ cache，key 固定通道、跨一组 token 共享 scale，value 固定 token、沿一组特征共享 scale。前者处理通道离群值，后者避免绑定不同 token 的范围；asymmetric 主要指轴不同，不等于零点非零。

流式分组不能使用未来 token 的统计，因而保留近期高精度 residual cache。残留长度、组长和元数据共同决定实际位宽，不能把“2 bit”套给所有位置。轴选择仍须在目标模型与工具轨迹上核验。

14.2 KVQuant：离群值、RoPE 与非均匀数据类型

KVQuant[23] 结合 per-channel key、RoPE 前量化、按层敏感度加权的非均匀码本与逐向量稠密 – 稀疏离群值表示。RoPE 混合通道后会改变逐通道范围，故保存 pre-RoPE key 利用原通道统计；这不是随机旋转方法。

将缓存的 pre-RoPE 量化 key 反量化为 $\hat{k}_j$ 后，位置 t 对历史位置 j 的分数为

$$(R_t q_t)^\top (R_j \hat{k}_j)/\sqrt{d_h}.$$

历史位置的 R_j 不能省略，量化与旋转不能任意交换。论文融合反量化与在线 RoPE，复现还须核对 kernel 和稀疏离群值路径。

14.3 GEAR：量化加低秩/残差修复

GEAR[24] 组合低精度分量、低秩修复与稀疏离群值：

$$C \approx \hat{C}_{\text{low}} + UV^\top + C_{\text{sparse}}, \qquad C \in \{K, V\},$$

$\hat{C}_{\text{low}}$ 是码字反量化后的实数张量，不能直接把整数码字加到实数残差。低秩因子、稀疏值与索引计入内存，修复的访存、矩阵乘和分支计入时延；指定任务上的“接近无损”仍须在 Agent 上验证。

14.4 KV 量化的工程检查表

上线前核对六项：scale 的离线、prefill 或 decode 统计时机；K/V 的轴、位宽与零点；RoPE、滑窗、GQA/MQA 与 attention kernel 的兼容性；元数据对分页及前缀共享的影响；淘汰与量化的先后顺序及误差信号；工具结果、代码、图像和指令的角色保护。

实验建议：固定权重、工具轨迹和 cache 字节预算，比较 FP16、INT4、INT2 与 token eviction 的组合，报告任务成功率、pass^k、KV bytes/token 和 TPOT 的 Pareto 前沿。

第十五章

语义压缩：上下文、RAG、Memory 与工具 Schema

15.1 数值压缩与语义压缩的边界

数值量化试图让

$$f_\theta(x) \approx f_{\hat{\theta}}(x),$$

而语义压缩通常把输入 x 变为 $c(x)$，只希望目标任务保持：

$$\Pr(y \mid x) \approx \Pr(y \mid c(x)).$$

后者更依赖任务和验证器，不能用单一张量误差描述。Agent 系统应保留原始证据、摘要版本、压缩策略和 provenance，以便发生争议时重放。

15.2 RAG 与证据链

RAG[25] 把外部文档引入生成；DPR[26] 和 REALM[27] 分别代表 dense retrieval 与可学习检索路线。部署时可以压缩 embedding、索引、文档 chunk 和上下文，但每一层都会改变召回。建议分开报告：召回率、重排质量、打包后的证据覆盖、引用正确率和最终动作正确率。

15.3 长期记忆

Agent memory 不是简单向量库。它包含事件、摘要、事实、用户偏好、程序技能、时间戳、权限和删除请求。压缩记忆时可以做去重、层级摘要、实体合并和低精度 embedding，但必须保留冲突版本和原始来源。一个过度压缩的“用户偏好”可能变成永久错误信念。

15.4 工具 Schema 与前缀缓存

工具 schema 往往在每一轮重复出现。模板、短 ID 或结构化语言可以缩短描述，但改变编码后要验证模型是否仍理解参数类型、权限和副作用。精确前缀缓存则复用相同 token 前缀的计算，通常不改变逻辑 token 数；decode 仍访问对应 KV。SGLang[28] 的 RadixAttention 管理共享前缀；Agent 还应把复用身份与模型、tokenizer、位置及精度配置、工具版本和租户权限绑定，避免把文本相似误当作状态相同。

15.5 Agent 专用前沿信号

本次核验截至 2026 年 10 月 22 日。所收录的 TriAxialKV[29] 预印本按时间、模态和语义角色组合标签，为各标签选择 INT2 或 INT4。其校准时一次只量化一个标签、其余保持浮点，重放 attention 并记录输出 MSE $D_k(2)$ 与 $D_k(4)$，再在字节预算下按近似可加的 $\sum_k D_k(b_k)$ 分配位宽。这是张量误差代理，既未直接优化任务成功率，也不能忽略多组同时量化时的交互；最终配置仍须在完整轨迹上验证。

ReCache[30] 预印本先采用 resource-wise attention，取消资源之间的交互，并使用 resource-local position，使单个资源的编码不随资源组合或顺序改变；在这套注意力与位置规则下，才可独立预计算并组合资源 KV。随后再按贡献选择 layer 与 KV-head-group 路由，结合结构和语义剪枝。该方法改变了资源编码的依赖关系，不能解释为原有全注意力模型可以任意拼接独立计算的缓存。

这些预印本可作为研究与复现线索。评估具体算法时，应固定版本、任务轨迹、实际缓存字节与实现，核验它们是否在相同可靠性门槛下改善资源成本。

表 15.1 Agent 语义压缩的可逆性设计

对象	压缩操作	必须保留的可审计信息
RAG 文档	chunk 去重、摘要、重排	原文指针、文档版本、检索分数、引用 span
对话历史	摘要、状态化、删除 filler	时间顺序、说话者、未解决承诺
工具 schema	模板化、短 ID、prefix cache	参数类型、ACL、版本、不可逆副作用
工具结果	结构化字段、异常过滤、摘要	原始返回、错误码、时间戳、请求 ID
长期记忆	合并、降权、过期	provenance、冲突版本、用户删除记录

第二部分

Agent-aware 压缩：从张量误差到行动可靠性

第十六章

量化误差在 Agent 轨迹中的传播

16.1 局部误差、离散动作与状态转移

把一次 Agent 决策抽象为 logits z_t、动作解析器 $\mathcal{P}$ 和执行器 $\mathcal{E}$：

$$a_t = \mathcal{P}(z_t), \qquad s_{t+1} = \mathcal{E}(s_t, a_t).$$

在贪心解码且 parser 确定时，量化带来的 δz_t 跨过决策边界才会改变动作；在采样解码下，即使 top-1 不变，概率质量变化也会改变动作分布。一旦改变动作，环境状态可能发生不可逆跃迁。因此平均 logit MSE 对风险的描述很弱。应估计每个决策的 margin、动作集合 KL 和后置状态差异。

16.2 四种常见失败链

1. **格式链**：量化扰动括号、引号或枚举 token，JSON 解析失败；
2. **选择链**：工具名称的 logit margin 变小，选错 API；
3. **参数链**：函数名正确但数字、日期、实体 ID 错误；
4. **时序链**：停止/继续边界改变，过早结束或无限重试。

这些失败可分别用 parser、schema validator、业务规则和 goal checker 诊断，不应统称为“模型质量下降”。

16.3 错误放大模型

设第 t 步错误类型为 e_t，环境转移将其放大系数记为 G_t。一个可用于分析的近似是

$$\mathbb{E}[\text{loss}_{1:T}] \approx \sum_{t=1}^{T} G_t \, \Pr(e_t).$$

该式是忽略交互和重复计损的启发式，不是一般成立的误差传播定理。G_t 需由预先定义的后果等级或受控干预估计；同一早期错误导致的多个后续错误不能无条件重复相加。高风险工具、不可逆动作和缺少验证器的步骤可据此优先验证更高精度、约束或人工批准，但不能事先断言增加 bit 必然降低风险。

16.4 量化后的行为等价并非文本等价

如果原模型和量化模型在大多数普通 token 上一致，却在工具选择 token 上不一致，任务成功率仍会显著变化。相反，一些自然语言表达的差异并不改变环境状态。故应同时保存 FP 与量化模型的 logits、解析动作、工具返回、验证器结果和最终状态，进行反事实重放。

第十七章

轨迹感知校准与混合精度分配

17.1 把 calibration sample 改写为 Agent event

传统校准样本是 token 序列；Agent 校准样本应是带事件标签的轨迹：

$$\tau = (e_1, \dots, e_T), \quad e_t = (\text{role}, \text{span}, \text{tool}, \text{risk}, \text{outcome}).$$

role 可取 system、user、tool-name、tool-argument、tool-result、error、final、stop；若运行时公开可用的推理片段，也可单独标注，但校准不能依赖部署时不可观测的内部思维。risk 可由副作用、权限和可逆性标记。保存工具 Schema、来源、位置编码和环境快照；outcome 是离线标签，不能泄漏到模型输入。对每类 event 估计 activation range、logit margin、注意力贡献和失败率。按任务族划分校准、开发与测试集，避免同一轨迹的相邻片段跨分区。

17.2 敏感度估计

对层、通道或 KV head/事件组 i，固定其他单元为参考精度，只替换该单元，得到候选 b 的局部干预。所有模型重放同一 FP 前缀，不让前一步分叉混入当前敏感度。可用以下信号估计敏感度：

$$S_i^{\text{out}} = \mathbb{E}_{x\sim\mathcal{D}} \| f_i(x) - \hat{f}_i(x) \|^2, \tag{17.1}$$

$$S_i^{\text{act}} = \mathbb{E}_{\tau}\, \mathrm{D}_{\mathrm{KL}}(p_{\mathrm{FP}}(a \mid \tau) \| p_{\mathrm{q}}(a \mid \tau)), \tag{17.2}$$

$$S_i^{\text{risk}} = \mathbb{E}_{\tau}[G_t \mathbf{1}(\text{wrong action})]. \tag{17.3}$$

组合成

$$S_i = \lambda_o S_i^{\text{out}} + \lambda_a S_i^{\text{act}} + \lambda_r S_i^{\text{risk}}.$$

三个量的单位和尺度不同，应先按参考能量、事件数或预先固定的尺度归一化，再确定权重；否则大张量会支配总分。动作 KL 只在相同合法动作集合、相同 grammar 和相同前置状态下比较。若只能计算工具名或参数 token 的条件分布，应标为代理损失。G_t 是声明过的风险权重，不自动等于真实因果放大系数。敏感度统计和量化参数拟合只使用训练/校准分区；开发集用于选择配置，测试集只用于最终评估。

17.3 离散分配问题

令候选 bit 集合为 $\mathcal{B} = \{2, 3, 4, 8, 16\}$，每个单元 i 从可执行子集 $\mathcal{B}_i \subseteq \mathcal{B}$ 选择 b_i：

$$\min_{b_i \in \mathcal{B}_i} \sum_i D_i(b_i) \quad \text{s.t.} \quad C_{\text{fixed}} + \sum_i C_i(b_i) \le B_{\text{bytes}}.$$

其中 C_i 以 byte 计，包含 $n_i b_i/8$、scale、zero-point、索引、对齐和必要缓冲；C_{fixed} 是其余固定开销。动态 KV 应按约定的最大上下文、并发和保留策略估计峰值，而不是用一条短轨迹的平均 bytes。候选集 $\mathcal{B}_i$ 只允许目标 kernel 支持的布局；例如不支持角色切换精度的权重 kernel，不能把“按 token 角色改变权重 bit”当作免费操作。D_i 是局部干预损失，求和忽略了单元交互，只适合作为搜索代理。可用 greedy、dynamic programming 或整数规划产生候选，再在开发轨迹上整体验证。

17.4 一个可复核的最小算法

下面的贪心升级算法给出明确输入、成本和停止条件。它用于生成可执行候选，不保证达到全局最优。相同成本下的损失改进先于按 byte 排序的升级；成本相等且损失相同的配置按预先固定顺序打破平局。成本模型必须与后续实测显存对照；延迟不是 bit 数的线性函数，须另设 SLO 门槛。

Listing 17.1 按轨迹局部干预估计并分配精度

```
# Cal/Dev/Test are disjoint by task family, not by token.
# K[i] contains only layouts supported by the target kernel.
fp = record_reference(Cal, frozen_tools, reset_each_run=True)
for i in units:
    for cfg in K[i]:
        q[i, cfg] = fit_quantizer(i, cfg, Cal)  # scales use Cal only
        r = replay_same_prefix(fp, replace_only=(i, q[i, cfg]))
        D[i, cfg] = normalized_event_loss(r, fp, fixed_weights)
        C[i, cfg] = resident_bytes(i, cfg, max_load)  # metadata too
choice = cheapest_supported(K, C)
if fixed_bytes + sum_cost(choice, C) > budget_bytes:
    raise InfeasibleBudget
while True:
    upgrades = feasible_upgrades(choice, K, C, budget_bytes,
                                 fixed_bytes)
    # Positive marginal loss reduction per extra resident byte.
    u = best_positive_gain_per_byte(upgrades, D, C)
    if u is None:
        break
    choice = apply_upgrade(choice, u)
model = materialize_joint_model(choice, q)
# Local additive scores can miss interactions: validate jointly.
report = paired_rollout(model, FP, Dev, frozen_protocol)
if not passes_quality_and_latency_gates(report):
    return candidate_rejected(report)  # revise on Dev, then rerun
# Freeze all choices before the held-out test; never tune on Test.
return frozen_evaluate(model, Test, frozen_protocol)
```

固定权重、KV 量化、token 淘汰三者应分别做消融，再做联合候选。报告包括校准事件分布、候选列表、每单元损失与成本表、搜索顺序、最终布局、开发门槛和测试结果。若联合配置失败，不应只增加一个分数后宣称解决；需沿首次分叉定位交互，重新拟合或恢复相关单元精度，并再次在开发集验证。

研究者视角

检验“轨迹感知量化”假设的一组最低证据应包括：统一 FP baseline；至少两种非 Agent 校准集与一种 Agent 校准集；W4/W8 或 KV4/KV2 的 bit budget 对照；动作边界保持率；最终任务成功率与 pass^k；以及首次分叉的失败归因。

第十八章

工具调用、记忆检索与停止边界的保护

18.1 工具调用是结构化输出问题

一个工具动作可表示为

$$a_t = (name, arguments, \text{mode}),$$

其中 name 必须属于当前工具目录，arguments 必须通过 JSON Schema，mode 可能是 dry-run、execute 或 ask-human。量化实验要分别测工具名准确率、JSON 可解析率、Schema 通过率、字段准确率、业务约束通过率和环境目标成功率。JSON 合法不代表参数符合被选工具，Schema 合法也不代表收款人、日期或金额正确。对有副作用的工具，还要核验状态变化恰好发生一次、未触及非目标资源。

18.2 保护动作 token 的三种策略

1. **logit 蒸馏**：只在工具名、参数和 stop token 上加权 KL；
2. **高精度岛**：保留输出头、关键词表行或最后若干层的高精度；
3. **约束解码**：让 parser/grammar 屏蔽非法 token，把量化误差限制在合法动作集合内。

Schema 约束通常只保证语法或字段类型，不能保证选择了正确工具、实体或权限；若 grammar 同时限制工具集合，还应单独记录这种策略收益。高精度岛增加存储、分派和 kernel 分支，应测真实收益。

18.3 记忆检索边界

量化 embedding 或 query encoder 可能改变 top-k 检索排序。应记录原模型与量化模型的 recall@k、MRR、证据覆盖和最终动作差异；对于长期记忆，增加时间新鲜度、权限和冲突版本测试。若一条记忆来自不可信工具结果，压缩系统不能把它变成更“确定”的永久信念。

18.4 停止、升级与人工接管

停止、继续与升级共同形成动作分类边界。设完成、继续、澄清、升级的 logits 为 $z_t(a)$，应报告每一类的 calibration curve、selective risk 和在不同风险阈值下的 coverage。量化后如果置信度变得过度尖锐，系统可能更少请求人工却更常产生不可逆错误。

18.5 一组最小成对行为测试

为每条任务构建相同初始快照下的 FP/量化模型对；再给两者施加同一故障或扰动，形成“模型版本 × 条件”的四格设计。固定工具返回、权限、预算和目标状态，每次执行前复位环境；只比较模型版本时，不能同时替换检索器、grammar 或重试策略。

1. **工具误调用**：构造名称相近的只读/写入工具、邻近实体 ID 和单位换算，分别记录名称、参数、业务规则、状态副作用。合法 JSON 但错误写入属于失败。
2. **证据丢失**：同一长轨迹包含早期关键证据、后续干扰和版本冲突；在完整上下文与既定压缩策略下，检查证据覆盖、来源指针、最终动作和遗漏位置。答案猜对不能代替证据保留。
3. **拒绝与升级**：匹配主题和长度相近的授权请求与越权请求，分别测合法请求的误拒率、违规请求的执行率和正确升级率；不能把两个分母合成一个“拒答率”。
4. **恢复能力**：向同一步注入超时、空返回、可重试错误或冲突状态，验证是否在预算内恢复、是否重复执行非幂等写入、最终状态是否正确。报告仅在注入故障任务上的恢复率及额外时延。

所有工具副作用测试先在模拟器或隔离快照中执行。逐对保存首次分叉、失败类型和状态差异；既报告总体变化，也报告高风险子集。这样可以区分数值压缩、KV 淘汰、摘要和外部验证器各自的贡献。

第十九章

安全、隐私与风险敏感压缩

19.1 压缩改变攻击面

下面是需要测试的潜在失败机制，而非量化普遍损坏安全性的既定结论：

- 低精度导致拒答边界漂移，安全分类器漏检；
- 上下文摘要删除了权限或不可信来源标记；
- KV 复用跨租户共享，造成提示或工具结果泄漏；
- 模型文件或量化/稀疏 layout 的加载器存在实现缺陷；这属于软件可信计算基，需与模型行为误差分开审计；
- 压缩后重试/反思不能纠正错误，或在代理奖励下反复触发错误行为；是否加剧需用成对轨迹验证。

19.2 纵深防御

安全不应由模型 logits 单独承担。建议分层：模型层做拒答和风险识别；parser 层做结构化约束；policy engine 做身份、ACL 和数据流标签；sandbox 做资源与网络隔离；执行器做幂等、dry-run 和回滚；审计层保存 trace。量化只能在这个外壳内优化，不能因为模型变小而移除外壳。

19.3 风险敏感目标

把任务损失分布记为 $L(q,\tau)$，可用 CVaR 关注最坏尾部：

$$\mathrm{CVaR}_\alpha(L) = \min_\eta \left[\eta + \frac{1}{1-\alpha}\mathbb{E}(L-\eta)_+\right].$$

其中 $0<\alpha<1$，$(u)_+ = \max(u,0)$，η 是优化的分位数阈值。对离散损失，该表达式仍定义上尾 CVaR，不能简单用“超过某分位数的样本平均”替代而忽略分位点的质量。在固定显存预算下可把 CVaR 作为搜索目标，但它不会自动保证把 bit 分给正确单元；需用真实风险指标验证。报告 α、风险标签、损失尺度和尾部样本数。小样本下高 α 的估计会很不稳定。

常见误区

不要用“量化模型拒答率更低”直接宣称用户体验更好。低拒答率可能只是安全边界损坏；必须同时测攻击成功率、误拒答、可逆性、人工升级率和任务效用。

19.4 安全评测的分母与证据

对授权任务报告误拒率与成功率，对违规请求报告违规执行率，对故障任务报告恢复率；同时给出每类样本数和配对差值的区间。测试零次违规不等于风险为零。即使独立同分布的二项模型适用，$n>0$ 次零失败的单侧 95% 上界仍为 $1-0.05^{1/n}$；同一攻击模板的多个改写通常相关，应按攻击族聚类，而非当作独立证据。风险门槛应在开发集预先冻结；测试后不断改阈值会使区间失去原本含义。

第三部分

从低 bit checkpoint 到可用的 Agent 服务

第二十章

推理栈：prefill、decode 与 Agent 调度

20.1 两阶段的硬件性质

Prefill 处理整段输入，矩阵乘并行度高，通常更偏 compute-bound；decode 每次只生成一个或少量 token，需要反复读取权重和 KV，通常更偏 memory-bandwidth-bound。Agent 请求还会在两次模型调用之间插入检索、工具执行、人工批准和网络等待，因此端到端延迟不能只看模型 tokens/s。

定义

$$\mathrm{TTFT} = t_{\text{first token}} - t_{\text{request}}, \qquad \mathrm{TPOT} = \frac{t_{\text{last}} - t_{\text{first}}}{N_{\text{out}} - 1}.$$

TTFT 的起点必须声明是否包括排队、网络和预处理。TPOT 仅在 $N_{\text{out}} > 1$ 时定义；单 token 输出记为不适用，不填零。流式响应有分块时须统一时间戳粒度，不能把 chunk 间隔混作 token 间隔。对 Agent 还应记录 tool latency、等待用户时间、重试时间以及从请求到最终环境状态的 wall-clock latency。系统吞吐以单位时间完成的请求/任务或输出 token 报告，goodput 则只计满足预定质量与时延 SLO 的完成量；两者的分母和时间窗口必须一致。

20.2 连续批处理与请求异质性

多轮 Agent 请求的输入长度、工具等待时间和输出长度差异很大。continuous batching 可以动态加入/退出序列，但长上下文会占用更多 KV block，短请求可能被拖尾。调度器需要同时考虑剩余 token、cache bytes、优先级、风险级别和工具依赖。

20.3 显存账本

对整个服务进程的峰值，建议显式计算

$$M_{\text{peak}} = M_W + M_{\text{act}} + M_{\text{KV}} + M_{\text{runtime}} + M_{\text{temp}}.$$

权重通常由请求共享，KV/激活随活跃请求累加，不能把共享权重对每个请求重复求和。量化只会直接改变其中的某些项。实际部署时要分别报告设备总占用、进程占用、框架 allocated/reserved 口径及 allocator 碎片、CUDA graph workspace、通信 buffer、scale/zero-point、稀疏索引和临时反量化 buffer。各分项须在同一峰值时刻对账；把各自不同时间的最大值相加只是保守上界。

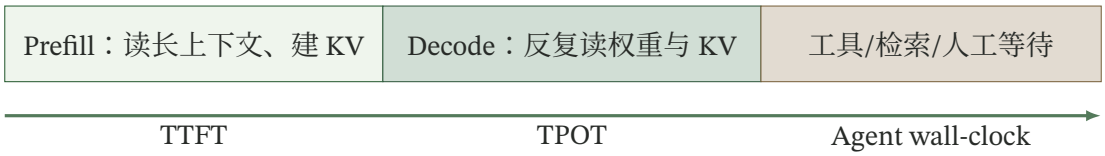


图 20.1 Agent 请求的时间线。只优化 decode kernel 可能无法改善工具等待主导的端到端时延。

第二十一章

PagedAttention、前缀复用与结构化运行时

21.1 PagedAttention

PagedAttention[31] 把 KV cache 划分成固定大小 block，以非连续物理块支撑逻辑连续序列，减少大块预留和外部碎片；尾块仍有内部碎片，并支持带写时复制管理的共享前缀。对 Agent，system prompt、工具 schema 和稳定的 memory preamble 可能跨请求共享；用户私有 observation 不能因相似文本而被错误共享。

21.2 Prefix cache 的正确性条件

数值正确性要求前缀 token ID、位置/RoPE 配置、模型权重及 adapter、attention mask、KV 格式和计算语义相容。语义相似的文本不能直接复用 KV。工具版本和权限若进入 prompt，也会改变 token 前缀；若 ACL 在模型外执行，即使数值 KV 相同，复用也不能授予新权限，执行器仍须独立鉴权。跨租户共享还涉及缓存命中侧信道和数据生命周期，应由明确隔离策略决定。任何压缩或量化 cache 都应保存模型/adapter 版本、格式、scale 和有效来源域，不能只按可见字符串做键。

21.3 SGLang 与 RadixAttention

SGLang[28] 用结构化语言程序和 RadixAttention 管理共享前缀、约束解码和复杂 LM 程序。它对 Agent 的意义是：工具调用、JSON grammar、few-shot 示例和反思模板可以视为程序节点，而不是一串无结构文本。量化 kernel 必须与这种树/基数结构的 block 生命周期兼容。

21.4 结构化输出与量化

结构化解码可以把非法 token 屏蔽掉，降低 JSON 失败，但会改变候选集合。应比较：

1. 无约束 FP16；
2. 无约束量化；
3. 约束 FP16；
4. 约束量化。

这样才能分离数值误差和 parser 约束带来的收益。

第二十二章

低 bit Serving：Atom、QServe 与 kernel co-design

22.1 为什么 checkpoint 压缩不等于加速

一个 W4 checkpoint 在执行前可能需要：解包 4 bit、读取 scale、反量化到寄存器/共享内存、执行矩阵乘，再把结果转成高精度累加。相对 FP16 的纯权重 payload，W4 名义上减少到四分之一；加入元数据和未量化层后，整模节省小于这个值。若解包和访存不规则，存储下降可能只换来很小的吞吐提升，甚至变慢。

22.2 Atom

Atom[32] 研究低 bit 权重与服务吞吐的协同，强调 kernel、batching、权重布局和量化格式要共同选择。评测时应把单请求、连续批处理和长上下文分开，因为低 bit 的收益随算子强度和 batch 改变。

22.3 QServe：W4A8KV4

QServe[33] 将 QoQ（W4A8KV4）算法与服务系统共同设计。progressive group quantization 先用逐通道尺度得到受限范围的 INT8 权重，再量化为分组 INT4，目的是降低 INT4 到 INT8 的反量化开销并利用 INT8 矩阵乘。SmoothAttention 通过等价缩放把 key 的量化困难迁移到未量化的 query，缓解 KV4 误差；它不是对所有 attention 张量统一套用同一种量化。系统再配合面向计算顺序的权重重排、寄存器并行和 KV4 attention kernel。

本节描述的是所引 v3 的 GPU 实现机制，不是任意 W4 文件都能获得的收益。其关键启示是：若激活和 KV 仍是 16 bit，权重压缩可能受其他瓶颈限制；同时降低三者时，又须控制误差叠加、反量化和布局成本。迁移到新模型、GQA 配置或硬件后，需要重新验证精度和性能。

22.4 量化 kernel 的检查表

- 累加精度是 FP16、BF16、FP32 还是整数？
- scale/zero-point 的布局是否与 tensor parallel 对齐？
- group size 是否匹配 warp、tile 和 memory transaction？
- 反量化发生在 global memory、shared memory 还是寄存器？
- KV block 的量化是否支持 flash-attention、GQA/MQA 和 paged layout？
- 多租户请求是否可能复用带有不同精度/权限的 prefix？

报告加速比时固定 GPU、驱动、模型、上下文、并发、请求到达过程和 SLO，分别说明 warm-up、编译/CUDA graph、prefix cache 冷热状态。公开 baseline 与候选 kernel 的版本和调用路径；若回退到高精度算子，应记为实测实现的一部分。只比较一个高度优化的量化栈和未调优 FP 栈，无法单独归因于位宽。

第二十三章

异构、分布式与长上下文部署

23.1 FlexGen 与 offloading

FlexGen[34] 将 GPU、CPU 和磁盘之间的权重、激活与 KV 调度写成优化问题，使单 GPU 也能运行大模型。对本地 Agent，offload 可以换取可运行性；但工具交互对延迟敏感，磁盘 I/O、PCIe 传输和 cache 迁移必须计入 wall-clock。

23.2 Splitwise 与 DistServe

Splitwise[35] 和 DistServe[36] 将 prefill 与 decode 分离到不同资源，分别优化 compute-bound 和 memory-bound 阶段。Agent 的多轮调用带来大量短 prefill、长 decode 或反之的混合负载，因此应按请求类型路由，而不是仅按模型大小分片。

23.3 Tensor、pipeline 与 expert parallelism

Tensor parallel 将矩阵切分到多卡，可能缩短计算时间却增加集合通信；是否降低单请求延迟取决于互联和张量规模。pipeline parallel 将层分段，需足够微批量才能摊薄流水线气泡；expert parallel 适合 MoE，却引入 token dispatch 和负载不均。量化布局必须与切分边界一致，避免每次通信都先反量化成高精度。

23.4 长上下文与跨设备 KV

当 KV 超过 GPU 显存，可分层存储：最近 token 在 GPU，中期 token 在 CPU 或压缩内存，冷记忆在外部存储。迁移策略应基于注意力/风险，而不是简单按时间截断。Agent 可在工具调用暂停时释放或压缩状态，但必须保存 token/位置、模型和 adapter 版本、量化元数据、逻辑块映射及权限上下文。恢复测试要比较最终状态和恢复延迟，防止把重新 prefill 的成本排除在报告之外。

第二十四章

边缘与移动端：LLM in a Flash 与实际约束

24.1 手机不是一张小 GPU

移动端受限于 DRAM 容量、内存带宽、闪存吞吐、热设计、功耗和后台调度。LLM in a flash[37] 研究把超出 DRAM 容量的部分权重放在闪存：利用 FFN 激活稀疏性和预测器，只装载需要的神经元权重；windowing 复用近期活跃神经元，减少跨 token 的重复传输；row-column bundling 把相关的上/下投影权重打包，使读取更适合闪存的大块传输。

这些机制依赖足够的激活稀疏、预测准确度和设备 I/O 特性，不能直接假定所有采用 SiLU 等激活的模型同样受益，也不等同于一般的 KV offload。随机小块访存、预测遗漏和 DRAM 缓存迁移仍有成本。书中边缘架构是部署方案；论文在其模型和设备上的报告不构成本书对手机或任意平台的实测。

24.2 边缘 Agent 的分层模型

一个实用架构可分成：设备上的小量化 router/拒答器；本地 W4/W8 主模型；必要时云端高精度模型；以及本地工具和隐私过滤器。路由条件不应只看置信度，还应看网络状态、数据敏感度、任务可逆性和用户是否允许上传。

24.3 能耗与热约束

报告移动部署时，应测每任务能量而非只测峰值功率：

$$E_{\text{task}} = \int_{t_0}^{t_1} P(t)\,dt, \qquad \text{J/success} = \frac{E_{\text{all runs}}}{N_{\text{success}}}.$$

一个更小但需要更多重试的模型，可能比稍大的稳定模型消耗更多能量。测量需声明墙端/电池/芯片功率口径、采样频率、是否扣除空闲基线、冷启动与稳态时段；升级到云端的网络与服务成本应另列。若没有成功任务，J/success 不定义，应报告失败和总能量，不能填零。

第二十五章

Speculative Decoding、模型级联与早退

25.1 Speculative decoding 的基本机制

小 draft model 从分布 q 提议 k 个 token，target model 用分布 p 验证。经典精确采样算法 [38] 对提议 x 以 $\min\{1, p(x)/q(x)\}$ 接受；首次拒绝时从归一化的 $(p-q)_+$ 残差分布重采样并丢弃其后的草稿。全部接受时还可从 target 生成一个额外 token。适当实现可保持 target 的采样分布，而不是只看 top-1 是否相同。量化 draft 主要改变提议质量和接受率；量化 target 则改变需要保持的目标分布。

25.2 Agent 特有的接受条件

精确性要求两模型使用相容的词表、前缀和目标解码约束，并正确实现接受/拒绝校正。在相同 grammar 下重新归一化分布时，概率比也必须与该约束一致。额外按工具或风险拒绝一部分已被接受的 token 会改变采样协议，不能未经证明继续宣称分布等价。

任何草稿中的工具动作都不能在 target 验证前执行；非幂等写入更不能因推测或重试重复提交。可在动作边界结束草稿块、由 target 直接生成高风险动作，但需报告这项路由规则和开销。加速收益取决于 draft 成本、接受长度、target 验证 kernel、并发和上下文；接受率高本身不证明端到端更快。

25.3 级联与 early exit

低成本模型先回答简单问题或选择工具，高成本模型处理不确定/高风险步骤。early exit 可以在中间层提前输出，但退出信号必须校准。一个安全路由器应满足：在给定 coverage 下，风险错误低于阈值；否则升级而不是静默接受。

表 25.1 部署策略的选择逻辑

场景	首选策略	主要验证
单卡低并发、需低门槛	W4A16 + paged serving	峰值显存、单请求 TTFT/TPOT、工具合法率
高并发云端	W4A8/KV4 + continuous batching	goodput、p95/p99、batch scaling、能耗
超长上下文	KV 量化 + token 选择 + 分层存储	长上下文任务、恢复正确性、bytes/token
移动/边缘	小模型路由 + W4/W8 + offload	J/success、热 throttling、隐私与离线能力
高风险工具	高精度动作头/target + draft 普通文本	误调用、拒答校准、人工升级与回滚

第四部分

评测、复现与研究路线

第二十六章

部署配方：从 baseline 到生产候选

26.1 配方 A：单卡研究 baseline

1. 选择 7B–14B 开源指令模型，保存 FP16/BF16 checkpoint 与 tokenizer；
2. 用 W4A16 GPTQ 和 AWQ 建立两个权重量化 baseline，固定 group size 128；
3. 用同一批 Agent trace 做离线 logit、动作解析和最终状态 replay；
4. 使用 paged serving，分别测短文本、工具 JSON、代码和长上下文；
5. 再加入 KV4/KV2，报告权重、KV 和运行时的显存分解。

这套配方的目标是定位瓶颈，而不是追求最低 bit。先知道是权重、KV、prefill 还是 tool latency 限制，后续优化才有意义。

26.2 配方 B：云端高并发

在固定 SLO 下比较 W8A8、W4A8KV4 与 FP16。请求生成器应模拟 Agent 的 burst：一个用户问题触发多次短模型调用、不同长度工具结果和偶发人工等待。记录吞吐、p95/p99、队列等待、cache 命中、GPU 利用率和每成功任务成本。

26.3 配方 C：边缘/本地

先以 W4A16 作为可运行候选，再测试 W3/W2 或低秩补偿。把模型装载、首 token、每 token、工具执行、温度降频、后台抢占和断网重试都纳入能耗。若任务失败可以升级到云端，报告本地成功率、升级率、隐私暴露和总能耗。

26.4 配方 D：Agent-aware 研究

建立四个版本：

$$\text{FP16-reference}, \quad \text{Q-text-calib}, \quad \text{Q-agent-calib}, \quad \text{Q-agent-mixed}.$$

FP16 参考通常不需要 PTQ 校准，不应称为 text-calib。第二版与参考版区分量化影响；第三版只改变校准分布；第四版在相同总资源预算下检验混合精度。若第四版增加了 bytes 或使用不同 kernel，需作为额外变量报告。每个版本都运行相同任务、工具、配对 seed、预算和 goal checker，并按任务族划分校准/开发/测试。以上配方是实验方案，不是本书已经运行过的模型结果。

第二十七章

Benchmark 设计：从单步准确率到完整任务

27.1 环境与基准

工具/API 可参考 ToolLLM 与 API-Bank[39, 40]；跨环境 Agent 可参考 AgentBench[41]；多轮工具 – 用户一致性可参考 τ-bench[42]。GAIA[43] 侧重需要推理、网页或工具协作的通用助手问题；WebArena[44] 提供可重复的网站任务环境；OSWorld[45] 面向真实计算机应用中的多模态操作；SWE-bench[46] 以 GitHub issue 驱动的代码修改为任务。它们各有环境、执行接口和成功判定，不能把分数直接当成跨环境能力排名，也不能把后续 Verified/Lite 等变体当作原版的同一任务集。

自动程序修复量化研究 [47] 的所用 v2 比较六个 LLM、十三种配置，并在 HumanEval-Java 和 Defects4J 上同时分析修复、显存、推理时间和能耗。论文报告表明，在这些实现条件下，内存节省未必伴随时间或能耗下降；相近的修复总数还可能掩盖解出问题集合的变化。这支持同时测行为和成本，但不能推广为所有量化模型都会更慢。本书未复现其性能数字。量化评测不应只挑有利的静态 QA 子集。

27.2 任务分层

1. **动作微基准**：工具名、参数、schema、停止和拒答；
2. **短轨迹**：一到三次调用，有明确验证器；
3. **长轨迹**：检索、工具、代码和多轮记忆交错；
4. **扰动轨迹**：工具版本、网页布局、返回字段、用户措辞变化；
5. **风险轨迹**：间接注入、越权、不可逆操作和错误恢复。

27.3 公平比较的固定项

表 27.1 量化 Agent 实验的最低复现记录

类别	必须记录
模型	checkpoint hash、架构、tokenizer、RoPE/position 配置、adapter
量化	bit、format、group size、axis、scale 精度、校准样本、异常值策略
运行时	kernel、框架 commit、CUDA/Triton、GPU、驱动、线程/并行配置
负载	任务集合、工具版本、schema、上下文长度、batch、seed、温度、重试
成功判定	goal checker、最终状态、人工接管、失败类型、超时与回滚
系统指标	TTFT、TPOT、端到端时延、吞吐、峰值显存、能耗、成本

27.4 匹配任务与配对比较

每个任务 i、重复 j 从相同环境快照运行 FP 与量化版本，固定工具返回、提示、预算、成功判定和随机种子表。版本运行顺序随机化或交错，减少机器温度、共享负载和时间漂移影响。模拟工具可采用冻结返回；真实工具不可冻结时，应记录版本和状态并分块运行。相同 seed 是配对索引，不是数值路径一致性的保证。

令成功指标为 Y_{ij}^{q} 和 Y_{ij}^{FP}，先计算每任务配对差值

$$d_i = \frac{1}{n_i}\sum_{j=1}^{n_i}(Y_{ij}^{\mathrm{q}} - Y_{ij}^{\mathrm{FP}}), \qquad \widehat{\Delta} = \frac{1}{m}\sum_{i=1}^{m} d_i.$$

这样每任务权重相同，不会让重试次数多的任务支配结论。若任务分层，应预先声明各层权重。分别报告 FP 独有成功、量化独有成功、两者均成功和两者均失败的数量；单次成对 Bernoulli 数据可用 McNemar 检验作为补充，但多种 seed 或同任务改写不能假装成独立任务。

27.5 统计与置信区间

任务成功是 Bernoulli 变量，但同一任务的重复、同一用户或同一环境派生任务可能相关。Wilson 区间适合独立 Bernoulli 的一个成功比例，不应直接套在所有相关 rollout 或两个比例的差上。对跨任务总体比较，可用**按任务聚类的配对 bootstrap**：有放回抽取任务 ID，每次保留该任务全部 seed、条件和模型版本，再计算宏平均差值；重复例如 2000 次，取预先指定的 2.5%/97.5% 分位点形成近似 95% 区间。若多个任务属于同一共享场景/用户，应在更高一级重采样。样本很少时区间本身也不稳定，应公开任务数和重采样 seed。

此区间以任务总体为抽样目标，条件于当前每任务运行表；若还需估计重复采样噪声，可在独立性成立时，外层抽任务、内层共同抽取 FP/量化的 seed 对，形成层次 bootstrap。不要只打散行重采样，破坏任务关联和版本配对。比较 pass^k 与 pass@k 时，在每次重采样内按各任务成功数重新计算组合估计量，并只使用 $n_i \geq k$ 的同一任务集合；报告不同 k 对应的有效样本数。

延迟至少报告中位数、p95、p99、超时率和原始样本数；尾分位点需要足够样本，不应由几十次运行给出虚假的小数精度。超时是结果，不能先删去再报告“成功请求延迟”。质量与时延约束应事前冻结，例如成功率下降不超过容许界且 p95 满足 SLO。多方法、多精度、多指标的探索应与预先指定的主要比较区分；测试集不能反复用于选择最有利配置。

第二十八章

可观测性、重放与首次分叉分析

28.1 Trace schema

每个 Agent step 至少记录：

Listing 28.1 最小可重放的 Agent trace 事件

```
{
  "task_id": "...", "run_id": "...", "seed": 3,
  "step": 7, "role": "tool_call",
  "prompt_hash": "...", "model_variant": "w4a16_awq",
  "candidate": {"name": "search", "arguments": {"q": "..."}},
  "logit_margin": 1.37, "kv_bytes": 524288,
  "tool_version": "v3", "result_hash": "...",
  "state_before": "...", "state_after": "...",
  "goal_state": "pending", "risk": "medium"
}
```

不要把完整敏感内容无条件写入日志；可用 hash、脱敏字段和访问控制保存可验证性。

28.2 首次分叉

定义动作/解析器分叉时刻为 $t^* = \min\{t : a_t^{\mathrm{FP}} \neq a_t^{\mathrm{q}}$ 或 parser 状态不同$\}$。某一轨迹提前结束也属于分叉；完全一致才记为无分叉。事件按逻辑调用 ID 对齐，不按墙钟时间或原始 token 下标盲目 zip。输入前缀已不同时，后续 logit 差异不再能直接归因于当前量化。给定 FP 和量化 trace，从首次不同动作、parser 状态或状态输入开始比较：

1. logits 是否已经分叉？
2. 若 logits 相同，采样/grammar 是否分叉？
3. 若动作相同，工具返回或环境状态是否不同？
4. 若环境相同，后续上下文 packing 或 memory retrieval 是否不同？

这个流程能区分数值量化、随机采样、工具非确定性和语义压缩的责任。

28.3 反事实重放

将量化模型的某一步替换成 FP logits，或将 FP 的工具结果注入量化 trace，观察是否恢复成功。受控替换若改变结果，可提供该干预下的因果证据；不能仅凭一次恢复就断言该组件是唯一原因。干预必须从一致快照重启，固定其他条件，重复配对运行，并检查替换后的状态/工具返回是否仍相容。对不可逆工具，反事实执行必须在 sandbox 或模拟器中进行。分别报告同前缀的 teacher-forced replay 和自由 rollout：前者隔离局部数值误差，后者才检验累计行为与真实成本。

第二十九章

前沿研究主轴与可证伪问题

29.1 八条值得持续追踪的主轴

1. **轨迹感知量化**：校准分布从文本转向带角色、风险和后置状态的 Agent event；
2. **权重 –KV 联合预算**：在固定显存下共同选择 W bit、KV bit、token 保留率和 prefix cache；
3. **动作边界保持**：保护工具、停止、拒答和升级 logits，而不是平均 vocabulary；
4. **量化 –kernel 协同**：让低 bit 真正转化为 TPOT/goodput，而不是只减少文件大小；
5. **长记忆与缓存复用**：结合语义 provenance、租户隔离和可撤销压缩；
6. **风险敏感搜索**：根据动作后果动态决定精度、验证和人工批准；
7. **多模态 Agent 压缩**：文本、截图、音频和工具结果的不同敏感度；
8. **持续学习与回滚**：把线上轨迹作为经验，不让压缩模型自我强化错误。

29.2 可证伪的研究问题模板

一个好的问题应写成“在固定条件下，方法是否改善明确指标”：

> 在相同模型、工具、总 KV bytes、kernel 和 rollout budget 下，按 token role 分配 INT2/INT4 是否比统一 INT4 提高 pass[8]，且 p95 延迟不增加超过 5%？

这个问题可以被反驳；“Agent-aware quantization 更好”则不能。这里的 5% 是应事先选定的示例容许界，不是已测结果。还需冻结任务分布、重复次数、主要指标和非劣界，给出任务配对差值的置信区间；若试验规模不足以区分小差异，应报告不确定，而不是把“未显著”写成“等价”。

29.3 三个可执行项目

1. **Trajectory-aware mixed precision**：以工具/观察/代码/stop 角色构建校准集，联合优化 W4A16 与 KV2/4；
2. **Action-boundary compensation**：用输出头低秩补偿和 grammar 约束保护工具、拒答和停止边界；
3. **Budgeted agent serving**：把显存、能耗、风险和人工升级纳入统一路由器，比较单模型、多模型级联和 speculative decoding。

第三十章

十二周学习与实验路线

表 30.1 从量化基础到 Agent 部署研究的十二周计划

周	主题	实践任务	产出
1	Agent 闭环	实现 parser–tool–verifier 最小循环	trace schema
2	量化基础	手写 affine quantizer 与误差分析	notebook
3	PTQ	跑 GPTQ/AWQ W4A16 baseline	checkpoint 报告
4	激活	跑 SmoothQuant/LLM.int8 对照	W8A8 指标
5	QAT/LoRA	复现 QLoRA 或 LoftQ 小实验	适配器
6	KV 基础	测完整 KV 的显存、带宽和长度曲线	memory profile
7	KV 压缩	实现 KIVI 或 token eviction baseline	KV4/KV2
8	serving	接入 paged/continuous batching	TTFT/TPOT
9	Agent 校准	标注 tool/argument/observation/stop 角色	trajectory set
10	混合精度	做 layer/head/role budget search	bit allocator
11	可靠性	运行 pass^k、重放和首次分叉	failure taxonomy
12	写作	做消融、统计、局限和开放代码	workshop-style paper

本章抓住这些

最小成功标准不是“模型能跑起来”，而是你能回答：在同一任务和系统预算下，压缩让什么变小、什么变快、什么变坏、第一次坏在哪里，以及如何通过更高精度、验证器或回滚把风险控制住。

附录：公式、伪代码与参考文献

附录 A

量化与部署公式速查

表 A.1 常用公式

对象	公式
仿射量化	$q = \mathrm{clip}(\mathrm{round}(x/s) + z)$，$\hat{x} = s(q - z)$。
线性层输出误差	$\|XW - X\hat{W}\|_F^2$。
动作 KL	$\mathrm{D}_{\mathrm{KL}}(p_{\mathrm{FP}}(a) \| p_{\mathrm{q}}(a))$，$a$ 只取合法动作集合。
KV 显存	$M_{\mathrm{KV}} \approx 2B_{\mathrm{batch}} L H_{\mathrm{KV}} T d_h \, b/8 + M_{\mathrm{meta}}$。
峰值显存	$M_{\mathrm{peak}} = M_W + M_A + M_{\mathrm{KV}} + M_{\mathrm{runtime}} + M_{\mathrm{temp}}$。
动作边界	$\mathrm{ABP} = N^{-1} \sum_t \mathbf{1}[a_t^{\mathrm{FP}} = a_t^{\mathrm{q}}]$。
风险尾部	$\mathrm{CVaR}_\alpha(L) = \min_\eta [\eta + (1-\alpha)^{-1} \mathbb{E}(L - \eta)_+]$。
成本/成功	$\mathrm{Cost/Success} = \text{total cost}/N_{\mathrm{success}}$。

附录 B

最小 Agent-aware 评测伪代码

以下骨架显式建立模型对与环境快照；运行记录包含任务 ID，置信区间按任务聚类。FP 与量化均使用冻结协议，动作解析和环境目标分别计分。

Listing B.1 固定预算与快照的配对评测骨架

```
def evaluate_pair(fp, quant, tasks, tools, protocol, seeds):
    rows = []
    for task in tasks:
        snapshot = make_initial_snapshot(task, tools)
        for seed in seeds:
            pair = {}
            # Randomize version order with a separate recorded RNG.
            for name, model in randomized_order(fp=fp, quant=quant):
                env = restore(snapshot)
                pair[name] = run_agent(model, task, env, seed=seed,
                    max_steps=protocol.steps,
                    max_tokens=protocol.tokens,
                    timeout=protocol.timeout,
                    record_events=True)
            rows.append({
                "task_id": task.id, "seed": seed,
                "fp": score(pair["fp"], task.goal_checker),
                "quant": score(pair["quant"], task.goal_checker),
                "first_divergence": first_divergence(
                    pair["fp"], pair["quant"]),
            })
    # Keep all seeds and both model versions together per task.
    return paired_cluster_bootstrap(rows, cluster="task_id"), rows

def first_divergence(fp, quant):
    xs, ys = align_by_logical_call(fp.events, quant.events)
    for i in range(max(len(xs), len(ys))):
        if i >= len(xs) or i >= len(ys):
            return i, "early_end"
        x, y = xs[i], ys[i]
        if x.state_before != y.state_before:
            return i, "state_or_context"
        if x.parser_state != y.parser_state:
            return i, "parser"
        if normalize(x.action) != normalize(y.action):
            return i, "action"
        if x.state_after != y.state_after:
            return i, "tool_or_environment"
    return None
```

`score` 必须保存 JSON、Schema、参数、业务状态、成功、超时、拒绝/升级、恢复、时延和显存的原始分子分母；无适用事件的比率记为不适用。首次分叉是诊断标签，不替代 goal checker。不同工具造成事件数不同需先对齐，并保留原始 trace；不能因为 `zip` 在较短序列处停止，就把“提前终止”误记为没有分叉。此骨架定义接口与检查顺序，具体运行时需实现快照恢复、计时同步和持久化。

附录 C

最小量化与部署实验

C.1 与正文公式一致的参考量化器

本附录把数值公式、行为指标和资源账本连到可运行的程序。代码使用 Python 标准库，不下载模型，也不执行有外部副作用的工具。低比特权重通过整数编码后反量化模拟；它能检查数值与指标的含义，不能测量低比特 kernel 的速度。

`scripts/quantization_reference.py` 实现无符号分组仿射量化。每组的校准区间先包含零，即

$$\ell = \min(0, \min_i x_i), \quad u = \max(0, \max_i x_i), \qquad s = \frac{u-\ell}{2^b-1}, \quad z = \operatorname{clip}\big(\operatorname{round}(-\ell/s), 0, 2^b-1\big).$$

全零组约定 $s=1, z=0$。这样正、负常数组和单元素组也有明确表示，而不是用一个任意小的 scale 导致饱和。程序拒绝空输入、NaN、Inf 以及非法位宽、组长，不把异常数据静默替换成零。

Listing C.1 标量仿射参考：先舍入再加整数零点

```python
import math


def quantize_groupwise(values, bits=4, group_size=128):
    if not isinstance(bits, int) or isinstance(bits, bool) or not 2 <= bits <= 8:
        raise ValueError("bits must be an integer from 2 to 8")
    if not isinstance(group_size, int) or isinstance(group_size, bool) or group_size < 1:
        raise ValueError("group_size must be a positive integer")
    values = [float(v) for v in values]
    if not values or not all(math.isfinite(v) for v in values):
        raise ValueError("input must be nonempty and finite")
    qmax = (1 << bits) - 1
    codes, scales, zero_points = [], [], []
    for start in range(0, len(values), group_size):
        group = values[start:start + group_size]
        lo, hi = min(0.0, min(group)), max(0.0, max(group))
        scale = (hi - lo) / qmax if hi != lo else 1.0
        zero = max(0, min(qmax, round(-lo / scale)))
        codes.extend(max(0, min(qmax, round(v / scale) + zero)) for v in group)
        scales.append(scale)
        zero_points.append(zero)
    return {"codes": codes, "scales": scales, "zero_points": zero_points,
            "bits": bits, "group_size": group_size, "count": len(values)}


def dequantize_groupwise(quantized):
    group_size = quantized["group_size"]
    return [(q - quantized["zero_points"][i // group_size]) *
            quantized["scales"][i // group_size]
            for i, q in enumerate(quantized["codes"])]
```

这里使用 ties-to-even 舍入。务必保持 round$(x/s)+z$ 的操作次序：当 z 为奇数时，它与 round$(x/s+z)$ 在半整数处并不相同。例如 $x=3, s=2, z=1$ 时，前者得到编码 3、重建值 4，后者得到编码 2、重建值 2。部署时应检查真实 kernel 的舍入规则、scale 精度、饱和边界和打包布局，而不是仅比较一个函数名称。

C.2 仿射手算与实际字节的边界

正文第四章已经比较 2/3 bit、组长 4/8 和 clipping。本节再用非对称零点检查另一种常见配置。取

$$x = (-1.00, -0.20, 0.30, 0.90 \mid 0.05, 0.10, -0.15, 0.20), \qquad g = 4.$$

两组都跨过零，故不需要扩展校准区间。对 2 bit，第一组 $s = 1.9/3$，$z = 2$，第二组 $s = 0.35/3$，$z = 1$。编码与误差如下；表中 scale 仅为显示值，计算使用未截短的数值。

表 C.1 同一张量的仿射编码；组与组之间用竖线分隔

位宽	两组 scale	两组零点	编码
2	(0.633333, 0.116667)	(2, 1)	(0, 2, 2, 3 | 1, 2, 0, 3)
4	(0.126667, 0.023333)	(8, 6)	(0, 6, 10, 15 | 8, 10, 0, 15)

表 C.2 重建误差与理想打包账本是两个不同口径

位宽	MSE	MAE	含元数据字节	有效位宽
2	0.03465278	0.14583333	8	8 bit/元素
4	0.00070417	0.01958333	10	10 bit/元素

字节账本假设 codes 连续打包、每组一个 FP16 scale 和一个 8 bit zero-point，总计 $8b + 2(16 + 8)$ bit；本例正好整除字节。它尚未计入张量头、对齐、索引和运行时副本。程序的 `storage_cost` 按整个缓冲区一次向上取整，正文第四章另有按组补齐的例子，二者不能在组尾有余位时混用。

表中误差使用理想 scale（脚本为 binary64），并没有把 scale 先舍入到 FP16。若实际保存 FP16 scale，重建误差也应据保存后的值重新计算。因此“按 FP16 元数据计费”不等于“已经模拟 FP16 元数据的数值误差”。2 bit 第一组重建 -1 为 -1.266667，也说明原范围与可重建端点不总能同时对齐；超出原范围的重建值本身不能被误称为 clipping。

自检与练习

用上表的 2 bit 编码重建第一组，先求四个平方误差，再与表中全张量 MSE 比较。提示：第一组重建为 $(-1.266667, 0, 0, 0.633333)$，全张量 MSE 必须把第二组也计入，不能直接取第一组均值。再将两个 scale 保存为 FP16，区分编码变化与仅重建值变化这两种实验。

C.3 在本地虚拟环境运行

在本主题根目录执行以下命令；所有实验仅依赖 Python 3.9 或以上版本的标准库：

Listing C.2 可直接执行的数值和工具调用实验

```
python3 manuscript/scripts/run_minimal_experiments.py
```

入口使用调用它的 Python 创建独立虚拟环境，依次运行：

- `scripts/verify_experiments.py`：独立有理数参考与指标检查；
- `scripts/agent_tool_call_experiment.py`：工具调用重放；
- `scripts/kv_budget_experiment.py`：KV 长度与位宽扫描；
- `evidence/numeric_principles_check.py`：正文原理核验。

以上路径均相对 `manuscript/`。运行清单保存在 `output/experiments/run_manifest.json`，包含实际命令、解释器、平台、源码散列、结果散列与日志位置。独立有理数参考核对编码和重建，并检查常数组、尾组、异常输入、JSON/schema 错例和提前终止；原理核验检查正文手算、GPTQ 补偿、RoPE 次序和 KV 账本。2026 年 10 月 22 日的运行分别通过 216 项检查和 5 组原理核验。

C.4 工具调用：JSON、参数与首次分叉

实验固定四个合成任务、五个种子、同一组工具返回和两事件预算。第一事件由一个 3×4 线性选择器决定调用 weather、refund 或 search；参数槽位从任务提示中提取，第二事件为停止。FP32 基线对权重、乘法和累加均显式作 FP32 舍入。INT4/group12 和 INT2/group12 只量化这 12 个选择器权重，反量化后仍以相同浮点路径计算。它们不是预训练 LLM，也没有运行整数 kernel。

每次调用先用 `json.loads` 检查语法，再独立检查工具 Schema 的字段、类型和取值约束；例如布尔值不能冒充整数金额。参数正确率在本实验中要求工具名称和参数同时与任务标准答案一致。任务成功在本例中等同于该判定，停止状态也据此固定生成，并未独立评测学习到的停止策略。保存的逐条 trace 包含原始 JSON、解析动作、工具观察、种子和首次分叉的位置与类型。

表 C.3 本次合成实验结果；每配置 4 任务 × 5 种子

配置	JSON 有效	Schema 有效	参数正确	任务成功
FP32-reference	1.00	1.00	1.00	1.00
INT4-group12	1.00	0.75	0.75	0.75
INT2-group12	1.00	0.75	0.75	0.75

两种量化配置都在窄 margin 的 `borderline_weather` 上选错工具，五个种子均在第 0 步发生 `tool_or_arguments` 分叉，其余三个任务成功。这显示量化扰动可以跨过动作边界，但不能据此推断所有模型中 INT2 与 INT4 表现相同。JSON 有效率始终为 1.00，是固定序列化器保证的结果；它不是模型生成 JSON 的能力证据。

结果中的 `pass_5_by_task` 只表示本次该任务五次运行是否全部成功，不是从五条教学样本估计出的总体可靠性。种子仅控制很小的 logit 扰动，任务集也经过人为构造；不据此计算泛化显著性或安全结论。复现真实模型时，应换成实际生成的工具调用文本，使用独立测试任务，并按正文第三章与第二十七章定义配对重复、任务簇和置信区间。

C.5 逐步增加上下文的 KV 预算

预算脚本固定 $(B_{\text{batch}}, L, H_{\text{KV}}, d_h) = (1, 32, 8, 128)$，扫描 $T \in \{512, 2048, 8192, 16384\}$ 与 $b \in \{16, 8, 4, 2\}$。对于 K、V 等位宽、无残差区的稠密缓存，纯数据字节为

$$M_{\text{payload}} = \left\lceil \frac{2B_{\text{batch}} L H_{\text{KV}} T d_h b}{8} \right\rceil, \qquad M_{\text{cache}} = M_{\text{payload}} + M_{\text{meta}} + M_{\text{residual}}.$$

上述 M_{cache} 仅是逻辑编码账本，实际分配还可能增加对齐、页内空闲和内存池保留量。这里的 H_{KV} 是实际 KV 头数，而非 query 头数。$T = 8192$ 时，FP16/INT8/INT4/INT2 的纯数据部分分别为 1,024/512/256/128 MiB。元数据估计、分页、对齐和 scratch 均不包含在 `theoretical_cache_bytes` 中；`metadata_estimate_bytes` 明确为 null。

本机未配置目标 checkpoint、CUDA 设备和兼容量化 KV kernel，故 `measured_gpu_bytes` 与 `decode_latency_ms` 同样为 null。JSON 内保存六步实测协议：固定 token IDs、batch 和生成长度；记录模型/runtime/kernel/驱动与量化轴；预热、同步并重复；区分 allocated/reserved/peak 以及静态

模型占用；同时采集 TTFT、TPOT、分位时延、吞吐和 OOM。接入兼容后端后才可填入实测栏位。理论 payload 下降不能替代显存实测，更不能直接换算为服务加速。

C.6 真实模型的部署记录与复现接口

本工程直接可执行的范围是上述四个脚本。真实模型的 PTQ、回放和服务接口随框架版本变化，应按下列输入输出契约对接选定后端，并使用该版本的官方接口。先冻结 FP16/BF16 checkpoint、tokenizer、chat template、工具 Schema 与任务集；把校准、开发和测试任务分开。候选量化产物必须记录权重格式、位宽、组长、scale 精度、量化轴及后端支持；测试阶段用同一提示、种子、工具返回和预算与浮点基线配对。

协议与接口

每个候选保存一张不可变记录卡：模型及数据 SHA；硬件、驱动、框架与 kernel 版本；W/A/KV/runtime/scratch 的存储分解；JSON、Schema、参数、任务成功、ABP、首次分叉、pass^k 与 pass@k；TTFT、TPOT、p95/p99、吞吐、峰值显存和测量条件。文件缩小、显存下降和速度改善分别报告。对超出已测上下文、工具类型或并发范围的请求，应记录回退行为，不能沿用未验证结论。

真实模型实验仍待读者在目标硬件补齐。本附录的结果只证明参考公式、指标与文件记录可以复核；性能、长轨迹可靠性和安全性需要对应任务上的独立证据。

附录 D

论文阅读地图

阅读的基本单位是一条可核验主张：它改变了哪个对象，在什么假设下优化什么量，又用什么实验说明收益。建议按下面的次序各完成一张阅读记录卡，再根据自己的瓶颈扩展方法。卡片至少保存论文版本、压缩对象、优化目标、校准或训练成本、实现条件，以及尚未得到支持的外推。

D.1 先建立对象与成本地图

以压缩和低比特综述 [48, 49] 建立“权重、激活、KV、训练状态”四类对象；再用高效模型、推理与服务综述 [50, 51, 52] 检查它们如何进入计算、访存、通信和调度。综述帮助定位原论文，具体机制、配置与性能结论仍应回到原文、代码和版本记录核对。

读后检查：给同一个 4 bit 权重配置写出三个量：码字载荷、包含元数据的权重文件、运行时峰值内存。逐项说明哪些开销会随请求数或上下文长度增加。若无法把三者分开，应先回读“压缩的对象”和“量化的数学基础与数值格式”。

D.2 按优化自由度阅读权重量化

先对照 SmoothQuant[2] 与 GPTQ[4]：前者通过等价缩放调整量化前的分布，后者在校准输入定义的二次目标下补偿舍入。随后读 LoftQ[17]，区分低秩初始化与后续任务微调。若研究目标需要 2–3 bit，再读 QuIP# 与 AQLM[10, 11]，追踪它们增加的变换、码本或搜索自由度及其存储成本。

读后检查：把每篇目标函数写成同一 $Y = XW$ 方向，标出优化变量与冻结变量；逐一回答是否需要输入校准、是否反向传播、是否增加推理项。等价变换只在相应代数条件下保持原函数，代理目标变小也不自动说明 Agent 成功率提高。先复算一层或一个小矩阵，再比较整模型任务结果。

D.3 先分清 KV 的位置选择与数值表示

以 H_2O[18] 和 KIVI[22] 作为一组对照：一篇改变可访问的历史位置，另一篇改变保留元素的数值表示。再读 KVQuant[23]，检查量化轴、RoPE 次序、异常值和实际读取路径。此时应能把新方法放进式 (12.1)，说明它改变位置数、位宽、元数据、高精度残留区还是物理分配。

读后检查：写出一个头的 $T \times d_h$ 张量，并在图上圈出每个 scale 覆盖的元素。指出最近 token、历史证据和位置 ID 如何保留；若删除或摘要后还要恢复原文，列出重建 cache 所需的前缀、模型与位置条件。用同一轨迹分别测试“只选位置”“只量化”和二者组合，才能辨认收益与失效来源。

D.4 把服务论文的计时边界对齐

先读 PagedAttention[31] 的物理缓存管理，再读 SGLang[28] 的共享前缀复用；随后根据瓶颈选择 QServe[33] 的低比特计算路径，或 DistServe[36] 的 prefill/decode 分离。把每项设计画进同一条请求时间线：排队、prefill、decode、工具等待、恢复与最终完成。

读后检查：论文数字测的是单算子、模型生成还是完整任务？设备、batch、输入输出长度、并发和 SLO 是否相同？码字读取、反量化、临时缓冲和跨设备传输是否计入？原文的吞吐提升只支持

原文条件；迁移到另一种 kernel 或 Agent 负载时，需重新记录 TTFT、TPOT、尾延迟和任务成功率。

D.5 用行为证据收束阅读

用 ReAct[53] 标出动作与观察之间的闭环，再读 τ-bench[42] 的交互规则与后置状态评测；AgentBench[41] 可帮助扩展环境类型。阅读这些工作是为了定义任务和判定协议，它们本身不构成某种量化方案保持可靠性的证据。

读后检查：模型对是否从相同快照开始？JSON 可解析、参数合规和最终目标是否分别计分？重复运行是否共享任务难度或环境状态？首次分叉是模型动作改变、工具随机性，还是提前结束？将答案落实为“最小 Agent-aware 评测伪代码”中的字段，再运行“最小量化与部署实验”；先验证计分管线，再扩大任务与模型。

D.6 前沿论文与证据边界

截至 2026 年 10 月 22 日，书中收录的 TriAxialKV、ReCache 与自动程序修复量化研究 [29, 30, 47] 按固定版本的研究报告阅读。应分别追问其精度分配代理、资源编码条件和任务协议；不要把不同版本、模型或预算下的结果拼成统一排名。

一张合格的阅读记录卡应把证据分成四栏：**原文机制**记录算法和适用假设；**原文实验**记录模型、数据、硬件、实现与分母；**本地验证**记录实际执行的命令与结果；**待验证外推**记录新任务、新 kernel 或新风险范围。定理只在其假设内适用，论文实验只覆盖其测试条件，自己的研究设想须写成可证伪问题。

复用手算、合成工具任务和理论 KV 扫描时，应保留“已计算、已实测、尚未运行”的状态，并按最小实验附录核对命令与适用范围。

D.7 综合练习与解题提示

1. **元数据与填充**。复算表 4.2 中 3 bit、组长 4 的配置；若格式省去恒为零的 INT8 zero-point，总字节和有效位宽怎样变化？
   提示：先给每组的 12 bit 载荷分别补到整字节，再加两份 FP16 scale。省去零点只减少两字节，不会消除组尾填充。最后用 $8S/N$ 求有效位宽。
2. **更低位宽与更长上下文**。沿用 KV 预算算例的模型形状，比较长度 8192 的 INT4 与长度 16384 的 INT2。两者均每 64 个元素保存一个 FP16 scale 和一个 8 bit 零点，忽略残留区与分页：纯载荷相同是否意味着总字节相同？
   提示：把“元素个数乘位宽”和“组数乘每组元数据”分开计算；上下文翻倍会增加组数。所得是理论账本，解码时延仍需单独测量。
3. **动作边界**。设两个候选动作的浮点 logit 差为 $m > 0$，量化后每个 logit 的绝对扰动不超过 ϵ。给出保证二者顺序不翻转的充分条件，并说明为何平均 MSE 不能直接代替该条件。
   提示：最坏情况同时压低领先动作、抬高另一动作，差值最多减少 2ϵ。检查严格不等式；有更多动作时还要覆盖所有竞争者。这是给定扰动界下的局部结论，不保证后续整条轨迹成功。
4. **重复运行的分母**。同一任务的单次成功率为 p，先在独立同分布假设下推导 pass^k 与 $\text{pass@}k$，再说明同一任务多种子结果为何不能被当成同样数量的独立新任务。
   提示：分别使用“全部成功”和“全部失败的补事件”；从任务总体推断时保留任务 ID，对浮点/量化结果做配对，并以任务为单位聚类。独立性是计算公式的假设，不能从重复运行次数本身推出。

参考文献与原始技术资料

[1] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. *LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale*. Version v2 dated 2022-11-10. Aug. 15, 2022. arXiv: 2208.07339v2. URL: https://arxiv.org/abs/2208.07339v2 (visited on 10/22/2026).

[2] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. Version v7 dated 2024-03-29. Nov. 18, 2022. arXiv: 2211.10438v7. URL: https://arxiv.org/abs/2211.10438v7 (visited on 10/22/2026).

[3] Z. Yuan et al. *RPTQ: Reorder-based Post-training Quantization for Large Language Models*. Version v4 dated 2023-05-17. Apr. 3, 2023. arXiv: 2304.01089v4. URL: https://arxiv.org/abs/2304.01089v4 (visited on 10/22/2026).

[4] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. Version v2 dated 2023-03-22. Oct. 31, 2022. arXiv: 2210.17323v2. URL: https://arxiv.org/abs/2210.17323v2 (visited on 10/22/2026).

[5] J. Lin et al. *AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration*. Version v6 dated 2026-04-25. June 1, 2023. arXiv: 2306.00978v6. URL: https://arxiv.org/abs/2306.00978v6 (visited on 10/22/2026).

[6] W. Shao et al. *OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models*. Version v3 dated 2024-03-18. Aug. 25, 2023. arXiv: 2308.13137v3. URL: https://arxiv.org/abs/2308.13137v3 (visited on 10/22/2026).

[7] T. Dettmers et al. *SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression*. Version v1 dated 2023-06-05. June 5, 2023. arXiv: 2306.03078v1. URL: https://arxiv.org/abs/2306.03078v1 (visited on 10/22/2026).

[8] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer. *SqueezeLLM: Dense-and-Sparse Quantization*. Version v4 dated 2024-06-05. June 13, 2023. arXiv: 2306.07629v4. URL: https://arxiv.org/abs/2306.07629v4 (visited on 10/22/2026).

[9] J. Chee, Y. Cai, V. Kuleshov, and C. De Sa. *QuIP: 2-Bit Quantization of Large Language Models With Guarantees*. Version v2 dated 2024-01-15. July 25, 2023. arXiv: 2307.13304v2. URL: https://arxiv.org/abs/2307.13304v2 (visited on 10/22/2026).

[10] A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. De Sa. *QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks*. Version v2 dated 2024-06-04. Feb. 6, 2024. arXiv: 2402.04396v2. URL: https://arxiv.org/abs/2402.04396v2 (visited on 10/22/2026).

[11] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh. *Extreme Compression of Large Language Models via Additive Quantization*. Version v4 dated 2024-09-11. Jan. 11, 2024. arXiv: 2401.06118v4. URL: https://arxiv.org/abs/2401.06118v4 (visited on 10/22/2026).

[12] S. Ashkboos et al. *QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs*. Version v2 dated 2024-10-29. Mar. 30, 2024. arXiv: 2404.00456v2. URL: https://arxiv.org/abs/2404.00456v2 (visited on 10/22/2026).

[13] Z. Liu et al. *SpinQuant: LLM quantization with learned rotations*. Version v4 dated 2025-02-20. May 26, 2024. arXiv: 2405.16406v4. URL: https://arxiv.org/abs/2405.16406v4 (visited on 10/22/2026).

[14] S. Ma et al. *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits*. Version v1 dated 2024-02-27. Feb. 27, 2024. arXiv: 2402.17764v1. URL: https://arxiv.org/abs/2402.17764v1 (visited on 10/22/2026).

[15] Z. Liu et al. *LLM-QAT: Data-Free Quantization Aware Training for Large Language Models*. Version v1 dated 2023-05-29. May 29, 2023. arXiv: 2305.17888v1. URL: https://arxiv.org/abs/2305.17888v1 (visited on 10/22/2026).

[16] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. *QLoRA: Efficient Finetuning of Quantized LLMs*. Version v1 dated 2023-05-23. May 23, 2023. arXiv: 2305.14314v1. URL: https://arxiv.org/abs/2305.14314v1 (visited on 10/22/2026).

[17] Y. Li, Y. Yu, C. Liang, P. He, N. Karampatziakis, W. Chen, and T. Zhao. *LoftQ: LoRA-Fine-Tuning-Aware Quantization for Large Language Models*. Version v4 dated 2023-11-28. Oct. 12, 2023. arXiv: 2310.08659v4. URL: https://arxiv.org/abs/2310.08659v4 (visited on 10/22/2026).

[18] Z. Zhang et al. *H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models*. Version v3 dated 2023-12-18. June 24, 2023. arXiv: 2306.14048v3. URL: https://arxiv.org/abs/2306.14048v3 (visited on 10/22/2026).

[19] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava. *Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time*. Version v2 dated 2023-08-28. May 26, 2023. arXiv: 2305.17118v2. URL: https://arxiv.org/abs/2305.17118v2 (visited on 10/22/2026).

[20] Y. Li et al. *SnapKV: LLM Knows What You are Looking for Before Generation*. Version v2 dated 2024-06-17. Apr. 22, 2024. arXiv: 2404.14469v2. URL: https://arxiv.org/abs/2404.14469v2 (visited on 10/22/2026).

[21] Z. Cai et al. *PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling*. Version v4 dated 2025-05-15. June 4, 2024. arXiv: 2406.02069v4. URL: https://arxiv.org/abs/2406.02069v4 (visited on 10/22/2026).

[22] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu. *KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache*. Version v2 dated 2024-07-25. Feb. 5, 2024. arXiv: 2402.02750v2. URL: https://arxiv.org/abs/2402.02750v2 (visited on 10/22/2026).

[23] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami. *KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization*. Version v6 dated 2025-05-28. Jan. 31, 2024. arXiv: 2401.18079v6. URL: https://arxiv.org/abs/2401.18079v6 (visited on 10/22/2026).

[24] H. Kang, Q. Zhang, S. Kundu, G. Jeong, Z. Liu, T. Krishna, and T. Zhao. *GEAR: An Efficient KV Cache Compression Recipe for Near-Lossless Generative Inference of LLM*. Version v4 dated 2024-09-30. Mar. 8, 2024. arXiv: 2403.05527v4. URL: https://arxiv.org/abs/2403.05527v4 (visited on 10/22/2026).

[25] P. Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Version v4 dated 2021-04-12. May 22, 2020. arXiv: 2005.11401v4. URL: https://arxiv.org/abs/2005.11401v4 (visited on 10/22/2026).

[26] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. *Dense Passage Retrieval for Open-Domain Question Answering*. Version v3 dated 2020-09-30. Apr. 10, 2020. arXiv: 2004.04906v3. URL: https://arxiv.org/abs/2004.04906v3 (visited on 10/22/2026).

[27] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang. *REALM: Retrieval-Augmented Language Model Pre-Training*. Version v1 dated 2020-02-10. Feb. 10, 2020. arXiv: 2002.08909v1. URL: https://arxiv.org/abs/2002.08909v1 (visited on 10/22/2026).

[28] L. Zheng et al. *SGLang: Efficient Execution of Structured Language Model Programs*. Version v2 dated 2024-06-06. Dec. 12, 2023. arXiv: 2312.07104v2. URL: https://arxiv.org/abs/2312.07104v2 (visited on 10/22/2026).

[29] H. Shen, H. Wu, Y. Zhao, and R. Mullins. *TriAxialKV: Toward Extreme Low-Precision KV-Cache Quantization for Agentic Inference Tasks*. Version v1 dated 2026-05-16. May 16, 2026. arXiv: 2605.17170v1. URL: https://arxiv.org/abs/2605.17170v1 (visited on 10/22/2026).

[30] Y. Fang, S. Wei, H. Hu, and X. Shen. *ReCache: Efficient KV Cache Reuse and Compression for Tool-Augmented LLM Agents*. Version v1 dated 2026-08-20. Aug. 20, 2026. arXiv: 2608.19662v1. URL: https://arxiv.org/abs/2608.19662v1 (visited on 10/22/2026).

[31] W. Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. Version v1 dated 2023-09-12. Sept. 12, 2023. arXiv: 2309.06180v1. URL: https://arxiv.org/abs/2309.06180v1 (visited on 10/22/2026).

[32] Y. Zhao et al. *Atom: Low-bit Quantization for Efficient and Accurate LLM Serving*. Version v3 dated 2024-04-16. Oct. 29, 2023. arXiv: 2310.19102v3. URL: https://arxiv.org/abs/2310.19102v3 (visited on 10/22/2026).

[33] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han. *QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving*. Version v3 dated 2025-05-01. May 7, 2024. arXiv: 2405.04532v3. URL: https://arxiv.org/abs/2405.04532v3 (visited on 10/22/2026).

[34] Y. Sheng et al. *FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU*. Version v2 dated 2023-06-12. Mar. 13, 2023. arXiv: 2303.06865v2. URL: https://arxiv.org/abs/2303.06865v2 (visited on 10/22/2026).

[35] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini. *Splitwise: Efficient generative LLM inference using phase splitting*. Version v2 dated 2024-05-20. Nov. 30, 2023. arXiv: 2311.18677v2. URL: https://arxiv.org/abs/2311.18677v2 (visited on 10/22/2026).

[36] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang. *DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving*. Version v3 dated 2024-06-06. Jan. 18, 2024. arXiv: 2401.09670v3. URL: https://arxiv.org/abs/2401.09670v3 (visited on 10/22/2026).

[37] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, and M. Farajtabar. *LLM in a flash: Efficient Large Language Model Inference with Limited Memory*. Version v3 dated 2024-07-30. Dec. 12, 2023. arXiv: 2312.11514v3. URL: https://arxiv.org/abs/2312.11514v3 (visited on 10/22/2026).

[38] Y. Leviathan, M. Kalman, and Y. Matias. *Fast Inference from Transformers via Speculative Decoding*. Version v2 dated 2023-05-18. Nov. 30, 2022. arXiv: 2211.17192v2. URL: https://arxiv.org/abs/2211.17192v2 (visited on 10/22/2026).

[39] Y. Qin et al. *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs*. Version v2 dated 2023-10-03. July 31, 2023. arXiv: 2307.16789v2. URL: https://arxiv.org/abs/2307.16789v2 (visited on 10/22/2026).

[40] M. Li et al. *API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs*. Version v2 dated 2023-10-25. Apr. 14, 2023. arXiv: 2304.08244v2. URL: https://arxiv.org/abs/2304.08244v2 (visited on 10/22/2026).

[41] X. Liu et al. *AgentBench: Evaluating LLMs as Agents*. Version v3 dated 2025-10-04. Aug. 7, 2023. arXiv: 2308.03688v3. URL: https://arxiv.org/abs/2308.03688v3 (visited on 10/22/2026).

[42] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. *τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains*. Version v1 dated 2024-06-17. June 17, 2024. arXiv: 2406.12045v1. URL: https://arxiv.org/abs/2406.12045v1 (visited on 10/22/2026).

[43] G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun, and T. Scialom. *GAIA: a benchmark for General AI Assistants*. Version v1 dated 2023-11-21. Nov. 21, 2023. arXiv: 2311.12983v1. URL: https://arxiv.org/abs/2311.12983v1 (visited on 10/22/2026).

[44] S. Zhou et al. *WebArena: A Realistic Web Environment for Building Autonomous Agents*. Version v4 dated 2024-04-16. July 25, 2023. arXiv: 2307.13854v4. URL: https://arxiv.org/abs/2307.13854v4 (visited on 10/22/2026).

[45] T. Xie et al. *OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments*. Version v2 dated 2024-05-30. Apr. 11, 2024. arXiv: 2404.07972v2. URL: https://arxiv.org/abs/2404.07972v2 (visited on 10/22/2026).

[46] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* Version v3 dated 2024-11-11. Oct. 10, 2023. arXiv: 2310.06770v3. URL: https://arxiv.org/abs/2310.06770v3 (visited on 10/22/2026).

[47] F. Vallecillos-Ruiz, G. d'Aloisio, M. Hort, L. Traini, A. Di Marco, and L. Moonen. *Smaller Models, Unexpected Costs: Trade-offs in LLM Quantization for Automated Program Repair*. Version v2 dated 2026-07-03. June 25, 2026. arXiv: 2606.27205v2. URL: https://arxiv.org/abs/2606.27205v2 (visited on 10/22/2026).

[48] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. *A Survey on Model Compression for Large Language Models*. Version v4 dated 2024-07-30. Aug. 15, 2023. arXiv: 2308.07633v4. URL: https://arxiv.org/abs/2308.07633v4 (visited on 10/22/2026).

[49] R. Gong et al. *A Survey of Low-bit Large Language Models: Basics, Systems, and Algorithms*. Version v3 dated 2025-11-12. Sept. 25, 2024. arXiv: 2409.16694v3. URL: https://arxiv.org/abs/2409.16694v3 (visited on 10/22/2026).

[50] Z. Wan et al. *Efficient Large Language Models: A Survey*. Version v4 dated 2024-05-23. Dec. 6, 2023. arXiv: 2312.03863v4. URL: https://arxiv.org/abs/2312.03863v4 (visited on 10/22/2026).

[51] Z. Zhou et al. *A Survey on Efficient Inference for Large Language Models*. Version v3 dated 2024-07-19. Apr. 22, 2024. arXiv: 2404.14294v3. URL: https://arxiv.org/abs/2404.14294v3 (visited on 10/22/2026).

[52] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, H. Jin, T. Chen, and Z. Jia. *Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems*. Version v2 dated 2025-07-23. Dec. 23, 2023. DOI: 10.1145/3754448. arXiv: 2312.15234v2. URL: https://arxiv.org/abs/2312.15234v2 (visited on 10/22/2026).

[53] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. *ReAct: Synergizing Reasoning and Acting in Language Models*. Version v3 dated 2023-03-10. Oct. 6, 2022. arXiv: 2210.03629v3. URL: https://arxiv.org/abs/2210.03629v3 (visited on 10/22/2026).