

Ballan-Bench：面向真实 Unity 游戏客户端开发的 可验证 Code Agent 评测与数据构造方法

Andong Wang

技术报告，2026 年 8 月

摘要：代码大模型评测正从函数级程序合成扩展至仓库级软件工程。Unity 游戏客户端还包含 Scene、Prefab、序列化资源、运行时状态与跨帧行为，补丁正确性难以仅由编译或静态测试判定。本文提出 Ballan-Bench：从人工核验的 Gold 实现出发，经受控功能消融生成待测工程，移除 Git 历史、参考实现与隐藏测试后交由 Code Agent 完成，再由基于 Unity CLI、Pipeline package、unity mcp 和 Unity Test Framework 的 Headless Grader 评分。我们在双向传送门与视角旋转支持两项任务上完成 110 次正式 rollout，微平均通过率为 53.6%，微平均得分为 0.692；任务级宏平均通过率为 55.5%，宏平均得分为 0.715。相较主要采用 Pygame、Web 或 Godot 的既有游戏开发基准，Ballan-Bench 面向持续演进的 Unity 工程，覆盖 C#、Scene、Prefab、资源引用、Editor 生命周期与运行时行为。受七天周期、两小时容器、4 GB 内存和无图形 Debian 环境限制，当前评测集中于可机械判别的工程行为，静态美学、时域动态质量与实机性能仍有待后续研究。

关键词：Code Agent；游戏开发；Unity；软件工程评测；自动化测试；数据标注；多模态 Agent

1 引言

1.1 背景与问题

早期代码大模型评测通常以函数描述、输入输出样例和单元测试为核心。HumanEval 建立了以执行正确性衡量代码生成能力的经典范式 [1]。SWE-bench 将任务推进到真实仓库中的 Issue 修复，要求模型阅读上下文、修改多个文件并通过项目测试 [2]。AgentBench 等工作进一步强调多轮决策、工具调用和环境反馈 [3]。这些基准推动评测对象从一次性文本生成扩展到具有状态和执行反馈的软件工程过程。

游戏客户端开发包含更复杂的工程状态。Unity 项目的功能分布在 C# 脚本、Scene、Prefab、ScriptableObject、资源导入设置、序列化引用和 Project Settings 中。一次补丁可能顺利编译，却因组件挂载、生命周期、引用关系、物理配置或跨帧状态变化产生错误。若任务涉及角色移动、机关、动画、镜头或交互反馈，正确性还依赖运行环境中的连续行为。

商业项目中的验收标准具有明显层次。代码能否编译、对象是否存在、状态机是否到达目标状态等条件适合自动化；角色动作是否自然、特效是否清晰、镜头是否舒适、关卡节奏是否合理则通常需要领域人员观察。面向游戏客户端的 Code Agent 评测需要兼顾工程真实性与奖励稳定性，并明确当前自动验证可以覆盖的能力边界。

Ballan-Bench 的目标是将真实 Unity 开发需求转化为可复现的 Agent 任务。其基本原则包括：保留既有工程上下文；从人工核验的正确实现构造任务；隔离答案和隐藏测试；在 Headless 环境中重复执行；记录完整轨迹和错误；使用可解释的部分分与通过判定。

1.2 相关工作与定位

近期研究开始直接评估模型生成或修改游戏的能力。V-GameGym 收集 Pygame 样本，并将代码执行结果转化为截图和视频进行多模态评价 [4]。WebGameBench 以浏览器

原生游戏为任务载体，使用真实交互检查输入、规则、状态转换和重启流程 [5]。GameDevBench 在 Godot 4 既有项目中构造局部开发任务，并结合脚本测试与视觉反馈 [6]。GameCraft-Bench 同样采用 Godot，要求 Agent 从产品式规格生成完整项目与回放轨迹 [7]。GameEngineBench 将评测推进到 Unreal Engine 5 的真实 C++ 运行时环境，并在求解后注入隐藏测试 [8]。表 1 概括了这些工作的任务形态。

Ballan-Bench 关注持续演进的 Unity 客户端工程。Agent 在现有架构、资源和关卡约束下补全经消融的 Feature，输出为对既有仓库的受控修改。Scene、Prefab、C# 编译、资源序列化、Editor 生命周期与 Play Mode 行为共同构成任务环境。Gold 来源于人工核验的完整实现，隐藏评分关注功能恢复、运行时行为和回归安全。这一设置接近商业 Unity 项目中的客户端开发形态。

1.3 本文贡献

本文的主要工作包括：

- 提出一套以人工核验 Gold 为起点、以功能消融为任务基线的 Unity Code Agent 数据构造流程；
- 将 Unity CLI、Pipeline package、unity mcp 与 Test Framework 组织为 Agent 操作和隐藏评分的分层基础设施；
- 在双向传送门与视角旋转支持两项任务上跑通快照制作、rollout、Headless grading、Prompt 契约修订和跨任务汇总；
- 总结任务边界、奖励设计、版本冻结、泄漏控制和失败轨迹分析的领域标注实践；
- 从工业验收与通用智能两个视角讨论游戏 Code Agent 的研究空间。

2 Ballan-Bench 方法

2.1 任务数据单元与设计目标

Ballan-Bench 将一项可复现的 Agent 任务组织为一组相互绑定的工件，见表 2。工程快照提供初始状态，Prompt 定

表 1: 近期游戏开发与游戏生成基准的任务形态比较。

基准	环境	任务起点与输出	评测方式	主要关注点
V-GameGym [4]	Pygame / UI 沙箱	从自然语言生成自包含游戏代码与运行产物	代码、截图和视频的多模态评分	视觉游戏生成、审美与可玩性
WebGameBench [5]	浏览器原生游戏	从结构化需求生成并部署完整 Web 应用	浏览器交互与运行时 Agent 标签	需求到可用应用的交付能力
GameDevBench [6]	Godot 4	在教程来源的既有项目中完成局部任务	Godot 脚本测试，辅以截图或视频	多模态资产理解与局部游戏开发
GameCraft-Bench [7]	Godot 4	从产品式规格生成完整游戏与回放轨迹	构建门禁、回放证据与多模态 rubric	端到端游戏生成和交互验证
GameEngineBench [8]	Unreal Engine 5	在真实游戏代码库中完成限定范围的原生 C++ 任务	求解后注入隐藏测试并由裁判审计	既有引擎工程内的运行时集成型 C++ 开发
Ballan-Bench	Unity 6.3 LTS	从 Gold 消融快照出发，在既有工程中补充全功能	隐藏 Test Framework / Headless Grader	接近工业客户端环境的仓库级功能开发

表 2: Ballan-Bench 的任务数据单元。

组成	内容
初始工程	经功能消融和净化的 Unity 项目快照
任务描述	玩家可观察目标、工程范围、必要约束与交付条件
工具与预算	Shell/MCP 权限、Unity 版本、时间、内存与调用限制
Gold	经人工运行核验的完整实现与固定 Commit/Tag
负例基线	可编译、可运行且明确缺失目标能力的消融版本
隐藏验证	Edit Mode、Play Mode、自定义 Headless 检查与评分规则
过程记录	命令、观察、文件 Diff、Console、测试日志与错误阶段
结果标签	归一化得分、Pass/Fail、错误类型、耗时与成本

义需求边界，工具与预算限定可执行动作，Gold 和消融基线校准奖励，隐藏 Grader 产生结果标签，轨迹与日志支持失败分析。任一工件发生变化都可能改变任务难度或评价含义，因此需要统一版本管理。

任务校准优先保留当前模型仍有提升空间的样本。低通过率区间（1%–20%）适合观察稀疏成功和能力上界；中等通过率区间（20%–60%）适合稳定比较模型；0% 任务可用于识别能力边界，但需排除环境、Prompt 或 Grader 缺陷。高通过率和完全通过任务仍可承担回归测试、基本能力检查和训练早期课程作用，其区分能力会随模型提升而下降。

2.2 Unity Toolkits for AI

本文将 Unity CLI、Unity Pipeline package、unity mcp 与 Unity Test Framework 的组合称为“Unity Toolkits for AI”。该名称用于描述 Ballan-Bench 中的系统组织，不对应 Unity 官方发布的单一产品。

Unity CLI 位于 Editor 进程外，负责 Editor 安装与定位、项目管理、Batch Mode、构建、测试、结构化输出和退出码 [9]。Unity Pipeline package 安装在项目与 Editor 内，通过本地 HTTP API 暴露运行中 Editor 的命令、状态和自定义操作 [10]。unity mcp 是 Unity CLI 内置的协议适配层，将 Pipeline 中注册的命令映射为 MCP tools；能够执行 Shell 的 Agent 可直接调用 unity command 与 unity eval，依赖结构化 Tool Calling 的 Harness 可经 MCP 获得同一组 Editor 能力 [11]。Unity Test Framework 提供 Edit Mode、Play Mode 与 Player 测试，可由 unity test、Editor 命令行或 TestRunnerAPI 运行 [12]。

图 1 展示了两条主要链路。Agent 操作链路为 Code Agent 经 Shell 或 MCP 进入 Unity CLI，再由 Pipeline 控制 Editor；隐藏评分链路由 Grader 编排 CLI 和 Test Framework，收集结构化结果并生成分数。Unity 官方工具显著降低了环境编排、Editor 控制、运行观测和测试执行的成本。Feature 选择、消融边界、测试覆盖与评分规则仍由任务设计者负责。

2.3 以 Gold 为起点的任务构造

Ballan-Bench 将人工确认的完整实现视为任务生产起点。正常游戏开发继续发生在主分支；某项 Feature 完成并通过人工运行检查后，使用固定 Commit 或 Tag 保存 Gold 节点。Gold 证明需求在当前工程与 Unity 版本中可实现，也为隐藏 Grader 提供已知正例。

任务构造流程如下：

- 1. 从 Gold 节点建立独立消融分支；
- 2. 移除目标 Feature 的关键逻辑、配置或资源连接，同时保留依赖系统与关卡上下文；
- 3. 确认消融工程能够导入、编译并进入目标场景；
- 4. 在 Gold 与消融版本上分别执行隐藏 Grader，要求前者达到预期高分，后者稳定失败；
- 5. 从被测快照中移除 Grader、测试程序集、参考实现痕迹和 Git 历史；
- 6. 固定快照、Prompt、工具、Unity 版本与资源预算，发布正式任务版本。

消融范围直接影响任务语义。删除过多依赖会把功能补全转化为工程重建，并将失败原因混入无关缺失；保留过多答案痕迹则会降低难度，甚至形成直接复制路径。理想消融基线应保持工程健康、明确缺失目标能力，并为多种合理实现保留空间。Gold 与消融版本提供正负锚点，常见错误实现、合理替代实现和刻意绕过实现则用于变异测试，检查 Grader 的决策区域。

2.4 Rollout、隐藏评分与轨迹记录

不同模型配置接收相同工程快照、Prompt、工具权限、Unity 版本和容器预算。Agent 可编辑文件、检索项目、调用 CLI/MCP、读取 Console 并运行自定义检查。正式测试在 Agent 结束后注入，模型无法读取隐藏断言或 Gold Diff。

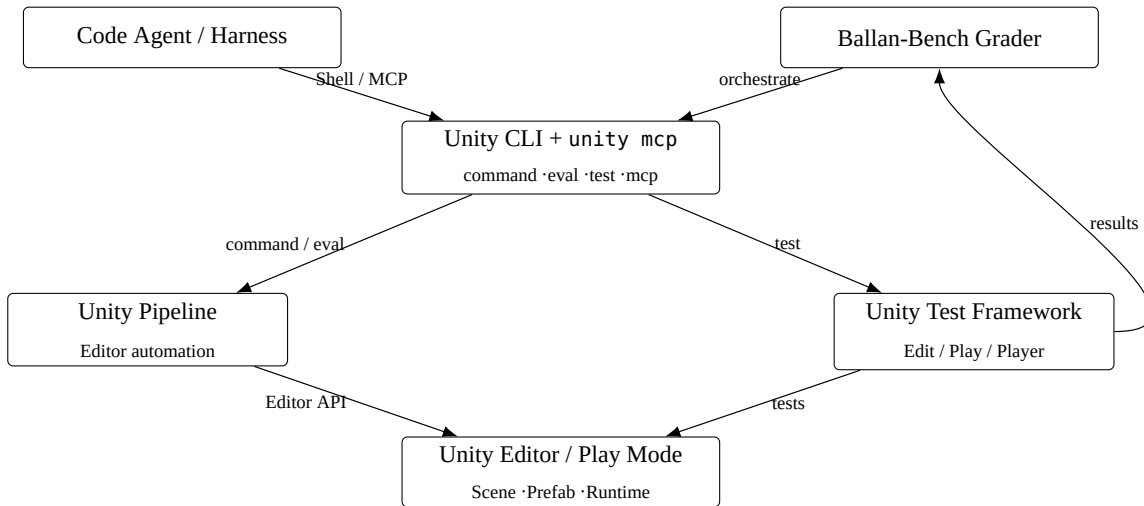


图 1: Unity Toolkits for AI 在 Ballan-Bench 中的分层架构。Agent 通过 Shell 或 MCP 进入 Unity CLI；Pipeline 负责运行中 Editor 的自动化，Test Framework 负责隐藏验证。

评分由编译状态、资产配置、运行时行为和回归检查等可观测量子项构成。部分得分描述任务完成程度，Pass 要求全部关键条件成立。被测 LLM 在执行阶段出现命令错误、编译失败、运行时异常、超时或不可恢复的工程破坏时，该次 rollout 按 0 分且未通过处理，并保留在统计分母中。这些错误反映模型的计划、工具使用和恢复能力。

每次 rollout 建议记录模型与思考配置、Prompt 和快照版本、Unity 工具链版本、工具调用、文件 Diff、Console 与测试日志、错误发生阶段、最终得分、Pass/Fail、耗时和成本。成功轨迹有助于分析有效的工程定位与验证策略；部分成功与错误轨迹可用于细化能力维度、发现 Grader 漏洞和构造后续训练数据。

2.5 质量控制与信息隔离

Agent 数据的可信度依赖任务、环境和奖励的一致性。Ballan-Bench 在任务发布前执行以下检查：人工确认 Gold 行为并保存运行证据；确认消融基线不存在无关错误；扫描 Git 元数据、测试文件名、注释、日志与资源命名中的答案泄漏；在 Gold、消融、常见错误和替代实现上回归 Grader；固定 Unity Editor、CLI、Pipeline、Test Framework 和系统镜像版本；将正式任务版本与后续修订分离，并保存可追溯的版本号和哈希。

3 任务构建案例

3.1 项目与共同边界

两项任务均构建于 Unity 6.3 LTS (6000.3.22f1) 的 Ballan-Bench 工程。项目采用 3D 重力球关卡环境，包含玩家球体、场景碰撞、关卡对象与基础输入系统。Prompt 描述玩家可观察行为、允许修改范围和必要约束，避免指定具体类、函数或实现步骤。被测实现只需满足行为与接口契约，无需复现 Gold 的内部代码组织。

3.2 双向传送门任务

Gold 版本在目标关卡中实现一对双向 Portal，并完成 C# 脚本、Prefab 和 Scene 集成。消融分支移除目标功能核心，保留输入系统、关卡结构、必要资源和非目标功能，使项目仍能在 Debian Headless 环境中加载和执行。隐藏脚本 `portal_grader.py` 在 Gold 上得到 1.0，在消融基线上得到 0.0，随后从被测快照中移除测试内容与 Git 历史。

传送门 Grader 将验证拆为静态工程检查和运行时行为检查。静态部分关注工程编译、Portal 对象与组件、Prefab/Scene 引用；运行时部分在 Play Mode 中驱动球体进入 Portal，检查成对连接、双向传送、目标位置与朝向、运动状态延续、重复触发保护及场景流程。Edit Mode 适合读取资产和序列化状态，Play Mode 用于跨帧观察触发与运动。

3.3 视角旋转支持任务

视角旋转支持任务要求在既有 Ballance 风格球体控制工程中加入相机视角旋转能力：按住 Shift 并输入水平方向键时触发一次 90 度左转或右转；右摇杆水平输入执行相同行为，并在回中后才允许再次触发。组合键不得同时驱动球体，右摇杆也不得参与球体移动。正常移动、既有玩法与相机行为需要保持可用。

任务还冻结了一项接口契约：活动玩家输入组件必须继续提供公开的 `Vector2 MoveIntent` 属性，名称、可见性与类型均不得改变。该接口既属于工程兼容性要求，也供隐藏 Grader 发现输入组件并驱动运行时测试。任务主要覆盖输入路由、离散状态机、相机与球体控制解耦、Unity 生命周期理解和接口兼容性。

3.4 Prompt 契约修订与版本冻结

首次 Prompt v2 未明确声明 `MoveIntent` 兼容约束，而 Grader 已将该接口作为场景发现与运行时驱动的必要条件。Flash-High 与 Pro-High 各执行 20 次，均得到 0% 通过率和 0.10 平均分。该结果反映任务描述与隐藏验证之间的契约不一致，因而不纳入模型能力统计。

Prompt v3 显式冻结 MoveIntent 的公开性、名称与类型，其余目标和 Grader v2 保持不变。随后对两个模型各重新执行 20 次 High rollout。Prompt v3 的 SHA256 为：

```
21463be4a5050b4b65cde63e054861d0
4315d460b494cb580f9738f216622e2b
```

这一修订说明，隐藏验证依赖的公开接口、场景发现方式和运行入口需要写入任务契约。契约缺失会把接口猜测混入目标能力，导致 0% 结果难以解释。无效版本仍应保存 Prompt、Grader、轨迹与统计记录，以支持数据审计；正式比较仅使用冻结后的有效版本。

3.5 视角旋转支持 Grader 与失败信号

视角旋转支持 Grader 首先执行结构门禁，检查活动输入组件以及公开 Vector2 MoveIntent 契约。运行时测试覆盖 Shift+A/D、Shift+Left/Right 的相反 90 度转向，以及右摇杆单步触发、回中复位和不驱动球体等行为。完全通过要求结构门禁与四项运行时检查全部成立，部分分用于区分接口保留但行为未完整实现的结果。

两项任务共同覆盖仓库检索、已有架构理解、C# 逻辑、Prefab/Scene/序列化配置、输入路由、跨帧状态、Console 反馈利用与自验证。典型失败包括定位错误扩展点、只修改脚本而遗漏资源配置、传送状态异常、触发器反复触发、键盘组合未产生离散转向、摇杆缺少回中门禁，以及编译、导入或 Play Mode 运行失败。

4 初步实验

4.1 实验设置

实验比较 DeepSeek-V4-Flash 与 DeepSeek-V4-Pro，并覆盖 Non-Thinking、Medium、High 和 Max。所有配置使用冻结的任务快照、Prompt、Unity 版本、工具权限与 Grader。环境为 Debian Headless、No Graphics，内存上限 4 GB，单个容器时限 2 小时。早期每模型 3 次的传送门 High 试跑仅用于打通流程，不纳入正式统计。

正式数据共 110 次 rollout：传送门任务 70 次，覆盖四档思考预算；视角旋转支持任务 40 次，仅比较两个模型的 High 配置。被测 LLM 执行阶段产生的 Error 统一按 0 分、未通过处理。

4.2 指标

Pass Rate 表示随机一次 rollout 满足全部关键条件的概率，Average Score 衡量部分完成程度。AnyPass 表示同批次至少一次通过，AllPass 表示同批次全部通过。当前全部正式配置均满足 AnyPass，且均不满足 AllPass；主分析使用 Pass Rate 与 Average Score。跨任务汇总同时报告按 rollout 加权的微平均和任务等权的宏平均。

4.3 双向传送门结果

表 3 按思考预算从低到高列出传送门任务的正式结果。70 次 rollout 中共有 34 次完全通过，总体通过率为 48.6%，加权平均得分为 0.632。任务位于中等通过率区间，在当前模

型集合上保持可解性和失败空间。

按模型聚合，Flash 完成 20/35 次，通过率 57.1%，平均得分 0.717；Pro 完成 14/35 次，通过率 40.0%，平均得分 0.547。Flash 的通过率高 17.1 个百分点，平均得分高 0.170。当前样本来自单题，Wilson 95% 区间仍有重叠，因此该差异属于初步观察。

按思考配置跨模型聚合后，Non-Thinking、Medium、High 与 Max 的通过率分别为 45%、45%、55% 和 50%，平均得分分别为 0.605、0.625、0.680 和 0.605。High 在本题上取得较高均值，Max 未继续提升，且其样本量更少。当前数据未呈现思考预算与成绩之间稳定的单调关系。

4.4 视角旋转支持结果

视角旋转支持的正式实验均采用 Prompt v3、Code Grader v2 和 High 配置，结果见表 4。40 次 rollout 中共有 25 次完全通过，总体通过率为 62.5%，平均得分为 0.798。Flash 完成 16/20 次，Pro 完成 9/20 次，二者相差 35 个百分点。当前结果仅覆盖单题和单一思考配置，不能单独作为跨模型总体排名。

15 次未完全通过中，12 次通过结构门禁并得到 0.55 分：四项运行时检查通过两项，但 Shift+A/D 与 Shift+Left/Right 未产生离散 90 度转向。另有 3 次均来自 Flash，结构门禁通过，但 Unity 未生成测试结果并以 exit code 1 结束，最终得到 0.10 分。当前报告保持中性记录，后续需结合 stdout/stderr 区分模型实现问题与评测平台问题。

4.5 跨任务聚合

表 5 汇总两项任务。微平均覆盖 110 次 rollout：59 次完全通过，通过率 53.6%，平均得分 0.692。任务等权宏平均通过率为 55.5%，宏平均得分为 0.715。Flash 在两项任务中的聚合结果均高于 Pro；由于样本仅来自两个任务，该结果仍不足以形成稳定的客户端能力排名。

两题配置覆盖不同：传送门包含四档思考预算，视角旋转支持当前只有 High。因此，跨任务聚合用于描述现有数据集，不用于推断思考预算效应。

4.6 结果解读

当前实验支持以下观察：

1. 两项任务与 Grader 均产生非退化分布，成功、部分完成与失败样本同时存在；
2. Flash 在两项任务的当前聚合结果均高于 Pro，跨任务综合排序仍需更多互补任务复现；
3. 传送门任务中增加思考预算未稳定改善成绩，视角旋转支持尚未进行多档思考预算比较；
4. Prompt v2 到 v3 的修订表明，公开接口与场景发现方式等隐藏验证依赖需要作为任务契约显式冻结；
5. Error 和未产出测试结果的样本保留在分母中，同时应继续标注工程定位、代码实现、Unity 集成、运行验证、异常恢复与基础设施阶段。

表 3: 双向传送门任务的正式 rollout 结果。

模型	思考配置	Rollout	通过	Pass Rate	Ave. Score
DeepSeek-V4-Flash	Non-Thinking	10	6	60%	0.76
DeepSeek-V4-Pro	Non-Thinking	10	3	30%	0.45
DeepSeek-V4-Flash	Medium	10	4	40%	0.57
DeepSeek-V4-Pro	Medium	10	5	50%	0.68
DeepSeek-V4-Flash	High	10	7	70%	0.81
DeepSeek-V4-Pro	High	10	4	40%	0.55
DeepSeek-V4-Flash	Max	5	3	60%	0.74
DeepSeek-V4-Pro	Max	5	2	40%	0.47
合计	–	70	34	48.6%	0.632

表 4: 视角旋转支持任务的正式 rollout 结果。

模型	Rollout	通过	Pass Rate	Ave. Score
V4-Flash	20	16	80%	0.843
V4-Pro	20	9	45%	0.753
合计	40	25	62.5%	0.798

表 5: 两项任务的跨任务聚合结果。

聚合维度	Rollout	通过	Pass Rate	Ave. Score
V4-Flash	55	36	65.5%	0.763
V4-Pro	55	23	41.8%	0.622
微平均合计	110	59	53.6%	0.692
任务宏平均	–	–	55.5%	0.715

4.7 后续实验协议

后续任务将覆盖逻辑 Bug 修复与隐藏回归、多帧状态和物理机关、脚本–Prefab–Scene–配置数据的跨系统 Feature、运行时性能与内存约束，以及更长程的工程阅读、方案选择、实现、自测和回归修复。最终排名将采用任务级宏平均，避免 rollout 数量较多的单题主导结果；功能正确性与跨 rollout 稳定性优先，时间、Token 和调用成本用于区分成绩接近的配置。

5 局限性与研究方向

5.1 实验条件限制

当前原型受以下资源边界约束：

- **开发周期：**环境搭建、任务制作、rollout 和报告初版共 7 天，任务覆盖与 Grader 迭代有限；
- **容器时限：**单容器最多 2 小时，限制长时间资源导入、构建、反复 Play Mode 调试与长程任务；
- **操作系统：**Debian Headless + No Graphics 无法执行依赖真实 GPU 渲染和窗口交互的验收；
- **内存：**4 GB 上限限制大型 Unity 项目、复杂资产和并行测试；

- **任务规模：**当前定量结果来自两项任务，仍不足以支撑稳定模型排名或广泛能力结论；
- **工具成熟度：**Unity CLI 与 Pipeline package 处于快速迭代阶段，复现实验需要固定版本 [9, 10]。

这些条件促使实验优先选择可通过脚本和 Headless 运行机械判别的功能。当前结论主要说明方法可行性、任务校准和数据生产流程。

5.2 工业界的主观验收

Code Agent 已显著缩短代码检索、样板实现和局部迭代耗时。商业游戏中的许多需求仍需策划、美术、动画、技术美术或程序人工验收。团队常用“看起来对”“手感正确”“节奏自然”等语言表达目标，这些判断依赖项目风格、玩家预期和长期积累的领域经验。

静态视觉问题包括构图、色彩、材质、信息层级、可读性和风格一致性。工程中的对象、Shader 参数和资源引用即使满足结构断言，最终画面仍可能缺乏表现力。工业价值评估需要同时覆盖可观测功能和审美质量。

5.3 时域动态与多模态 Grader

动画衔接、特效时机、镜头运动、物理反馈、战斗手感和关卡节奏属于时域动态问题。单帧截图无法表达连续过程，简单状态断言也难以描述动作自然度与反馈清晰度。未来可在具备图形能力的环境中生成确定性 Replay，采集关键帧、视频、对象状态和性能遥测，并组合以下信号：确定性测试负责结构与核心行为门禁；视觉语言模型评价构图、可读性与风格一致性；视频模型分析运动连续性、时机与因果关系；策划或美术对边界样本进行成对偏好标注；多评审者一致性与置信度描述主观标签的不确定性。

多模态模型能够降低初筛成本，其判断仍会受到 Prompt、参考图和模型偏好的影响。混合评分机制需要保留人工复核与不确定性表达。

5.4 机械可验证性的选择偏差

确定性 Grader 只能检查标注者编码的性质。若任务长期偏向容易自动测试的需求，数据集会强调编译、配置和有限

状态转换, 弱化需求澄清、架构权衡、开放设计与跨团队协作。固定初始状态或有限测试覆盖还可能诱发奖励投机。

缓解方式包括多 Seed、多初始状态、关键参数随机化、回归测试、Diff 审查和错误实现变异测试。Gold 与消融基线提供正负锚点; 合理替代实现与常见错误实现补充 Grader 的决策区域。正式版本冻结后, 修订应通过新版本发布并保留旧结果。

5.5 游戏世界与通用智能

游戏引擎提供可编程、可重复、可干预的世界。游戏开发同时包含现实规律的模拟与新规则的创造。Code Agent 需要阅读已有世界模型, 预测代码修改对环境状态的影响, 通过运行结果修正假设, 并将设计意图转换为可交互系统。

MineDojo 与 Voyager 展示了开放游戏环境在长期规划、技能积累、探索和环境反馈方面的研究价值 [13, 14]。Ballan-Bench 当前关注世界规则的代码侧构造与验证; 长期方向包括代码理解、视觉感知、物理推理、主动实验、长期规划与规则创造的联合评测。Agent 对代码的修改构成因果干预, 游戏运行结果提供可观测反馈, 这一闭环可用于研究 AI 如何理解、修改和创造复杂世界规则。

5.6 工业规模工程与实机性能验证

真实商业游戏仓库往往达到数百 GB, 完整资源导入、构建和打包的时间与存储成本远高于当前精简评测快照。在实际项目中, 单次完整打包常需 2-3 小时, 这一周期已超过或显著挤压当前容器的任务预算, 使 Agent 难以形成“修改-打包-实机验证-回归修复”的闭环。

游戏性能优化还依赖目标机型上的帧时间、内存、功耗、温度、加载与卡顿等遥测。不同硬件、驱动和内容场景会产生显著差异。面向此类任务, 当前可复现容器评测与工业级验收之间仍存在较大距离。后续可研究大型仓库与资产的分层缓存、增量构建、远程构建农场、目标设备池、标准化性能 Replay, 以及容纳数小时构建和多轮实机反馈的异步 Agent 评测协议。

6 结论

本文提出 Ballan-Bench, 一套面向真实 Unity 游戏客户端开发的可验证 Code Agent 评测与数据构造流程。该流程从人工核验的 Gold 实现出发, 通过独立分支进行功能消融, 使用隐藏 Grader 校准正负基线, 并在移除 Git 历史、参考实现和测试后生成模型任务快照。Agent rollout 与 grading 相互隔离, 执行错误按 0 分计入统计。

Unity CLI、Unity Pipeline package、unity mcp 与 Unity Test Framework 为环境搭建提供了关键便利。CLI 负责外部编排与 Editor 生命周期, Pipeline 提供运行中 Editor 的自动化后端, MCP 支持结构化工具调用, Test Framework 承担 Edit Mode、Play Mode 与 Player 验证。这组工具使 Scene、Prefab、Console、运行时状态和测试结果进入可编程工作流。

双向传送门与视角旋转支持两项任务共完成 110 次正

式 rollout, 微平均通过率为 53.6%, 微平均得分为 0.692; 任务级宏平均通过率为 55.5%, 宏平均得分为 0.715。两项任务均产生成功、部分完成和失败样本。Flash 在当前跨任务聚合中高于 Pro, 稳定排序仍需更多任务与统一配置复现。

从 Agent 数据标注角度, 一条完整样本应保存工程初态、任务描述、工具接口、资源预算、Gold、消融负例、隐藏验证、操作轨迹和最终标签。标注者需要界定 Feature 依赖、控制信息泄漏、把领域验收标准转换为可观测证据, 并审查 Grader 对替代实现和投机路径的响应。Prompt v2 到 v3 的修订进一步表明, 公开接口和场景发现方式属于任务契约, 契约缺失会直接削弱评测结论的可解释性。

Rollout 同时承担能力评测、难度校准、Grader 检查和训练数据发现功能。成功轨迹揭示有效的工程检索与验证策略; 部分成功轨迹帮助拆分能力; 编译错误、工具错误和运行异常暴露模型执行边界, 也可能反向发现 Prompt、环境与 Grader 的问题。任务生产适合采用“构造、运行、分析、修订、冻结”的迭代流程, 并在正式版本中维持可复现性。

当前方法集中于机械可验证的工程行为。后续工作将扩展 Bug 修复、物理行为、跨系统 Feature、性能约束和长程开发任务, 并探索确定性测试、图像与视频理解、运行时遥测和人工偏好相结合的评分机制, 为游戏工业中的 Agent 生产力评估及 AI 理解、修改和创造复杂世界规则的研究提供可复现的数据基础。

致谢

感谢 OpenAI GPT-5.6 Sol 模型与 DeepSeek-V4 模型对 Ballan-Bench 评测环境搭建所作出的贡献。

参考文献

- [1] M. Chen et al. “Evaluating Large Language Models Trained on Code.” arXiv:2107.03374, 2021. <https://arxiv.org/abs/2107.03374>.
- [2] C. E. Jimenez et al. “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?” ICLR, 2024. <https://arxiv.org/abs/2310.06770>.
- [3] X. Liu et al. “AgentBench: Evaluating LLMs as Agents.” ICLR, 2024. <https://arxiv.org/abs/2308.03688>.
- [4] W. Zhang et al. “V-GameGym: Visual Game Generation for Code Large Language Models.” arXiv:2509.20136, 2025. <https://arxiv.org/abs/2509.20136>.
- [5] W. Zhang et al. “WebGameBench: Requirement-to-Application Evaluation for Coding Agents via Browser-Native Games.” arXiv:2605.17637, 2026. <https://arxiv.org/abs/2605.17637>.

- [6] W. Chi et al. “GameDevBench: Evaluating Agentic Capabilities Through Game Development.” arXiv:2602.11103, v2, 2026. <https://arxiv.org/abs/2602.11103>.
- [7] T. Luo et al. “GameCraft-Bench: Can Agents Build Playable Games End-to-End in a Real Game Engine?” arXiv:2606.17861, 2026. <https://arxiv.org/abs/2606.17861>.
- [8] D. Le et al. “GameEngineBench: Evaluating Coding Agents on Real C++ Runtime Environments.” arXiv:2607.03525, 2026. <https://arxiv.org/abs/2607.03525>.
- [9] Unity Technologies. “Unity Command-Line Interface (CLI).” 2026. <https://docs.unity.com/en-us/unity-cli>. Accessed 2026-08-28.
- [10] Unity Technologies. “Unity Pipeline Package.” 2026. <https://docs.unity.com/en-us/unity-production-pipeline/local-tools-cli/unity-pipeline-package>. Accessed 2026-08-28.
- [11] Unity Technologies. “Unity CLI as the Replacement for the In-Editor MCP Server.” 2026. <https://docs.unity.com/en-us/unity-cli/replace-mcp-server-unity-cli>. Accessed 2026-08-28.
- [12] Unity Technologies. “Unity Test Framework: Running Tests.” 2026. <https://docs.unity3d.com/6000.3/Documentation/Manual/test-framework/running-tests.html>. Accessed 2026-08-28.
- [13] L. Fan et al. “MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge.” NeurIPS, 2022. <https://arxiv.org/abs/2206.08853>.
- [14] G. Wang et al. “Voyager: An Open-Ended Embodied Agent with Large Language Models.” arXiv:2305.16291, 2023. <https://arxiv.org/abs/2305.16291>.