

Code RL Mid-training项目文档

项目仓库: https://gitlab.xaminim.com/nlp/code_rl_midtraining

摘要

提出“在mid-training阶段进行 FIM (Fill-in-the-Middle) 代码强化学习 (Code RL)”的方法, 以增强语言模型的编码能力, 并由此迁移提升其作为code agent的综合能力。核心做法是在真实代码上下文中, 让模型补全目标函数体; 以 LLM-as-a-judge 基于“成功标准与 ground truth”进行语义评审来提供奖励; 随后用小规模 SFT对齐, 并在 SWE-bench-verified 等基准上闭环评测。

关键insight: 模型的coding能力与其作为agent base model的能力正相关。在补全目标函数体时, 模型常需调用第三方库 (如 numpy) 或文件内的其他函数, 这与 agent 任务中的“调用工具 (tools)”在本质上相似: 识别可用函数/库、遵守接口、进行参数规划、处理错误路径。因此, FIM Code RL 不仅提升“代码生成能力”, 也能训练“类似 agent 的工具使用与调用规划能力”, 有望改善如SWE-bench打patch的表现。

背景

- Code agent任务 (如在复杂仓库内定位问题、生成补丁、执行多步工具调用) 正在成为评估与应用落地的关键方向。
- 社区观察表明: 模型的纯编码能力与其代理能力呈显著正相关。编码能力越强, 模型越能:
 - 准确理解上下文与接口约束;
 - 稳健生成补丁并处理异常路径;
 - 更高效地利用反馈迭代修复。
- mid-training 是在基础模型与指令阶段之间注入结构化技能的关键窗口。FIM 形式允许在真实上下文中“局部补全”, 自然贴合大规模代码语料, 便于扩展数据规模。

Motivation (为什么要做)

- 能力迁移性强: FIM 补全目标函数时, 模型需要选择并调用库函数或文件内其他函数, 规划参数、遵循接口、处理错误, 这与 agent 的工具使用高度同构, 能直接迁移到如 SWE-bench “打 patch” 的子技能上。
- 数据规模可达: 相比端到端执行测试用例、搭建运行环境的 pipeline, FIM 任务更易自动化切片与标注, 适合 mid-training 所需的大数据量。
- 敏捷可迭代: 运行时评测 (test case) 在多仓库/多语言场景下工程成本高。用 LLM-as-a-judge 的语义评审能快速迭代, 如果发现LLM-as-a-judge效果不好, 再考虑引入运行环境测test case的方案。

方法概览（怎么做）

- 任务形式：FIM。提供完整文件上下文，将目标函数体替换为占位符，要求模型仅生成该函数体。
- 奖励来源：LLM-as-a-judge 对照“成功标准 + ground truth”进行结构化评分，生成密集与稀疏奖励。
- 训练流程：FIM Code RL（mid-training）→ 小规模 SFT 对齐 → 评测（SWE-bench-verified、TerminalBench 等）。

案例示意（一个完整 FIM 任务样例）

说明：

- 目标：补全 `get_graphql_response` 函数，使其正确调用 GitHub GraphQL API，处理 HTTP 和 GraphQL 错误并返回 JSON。
- 训练数据提供：任务目标、成功标准、完整代码上下文（含占位符）、LLM 评审模板。

训练样本JSON 结构示例：

Code block

```
1  {
2      "task_id": "fim_task_full_0001",
3      "language": "python",
4      "difficulty": "medium",
5      "task_brief": "补全代码仓库中获取 GitHub GraphQL 赞助者数据的函数，使其能安全地携带
Token 访问 API，并正确处理 HTTP 与 GraphQL 层面的错误，返回解析后的 JSON 数据。",
6      "success_criteria": [
7          "函数应从 settings.sponsors_token 读取 token，并以 Authorization: token
<value> 形式放入 headers。",
8          "请求使用 httpx.post，超时参数使用 settings.httpx_timeout。",
9          "请求体必须包含 query、variables (含 after)、operationName: \"Q\"。",
10         "当 HTTP 状态码非 200 时，记录错误日志（包含 after 与响应文本）并抛异常。",
11         "当响应 JSON 中包含 errors 字段时，记录错误日志（包含 errors 与响应文本）并抛异
常。",
12         "正常情况返回解析后的 JSON 字典对象。"
13     ],
14     "function_brief_for_model": "调用 GitHub GraphQL API，传入查询与分页变量，必要时
抛出异常并记录日志，返回 JSON 数据。",
15     "code_context_with_placeholder": "
16 import logging
17 import secrets
18 import subprocess
19 from collections import defaultdict
20 from pathlib import Path
21 from typing import Any
22
```

```

23 import httpx
24 import yaml
25 from github import Github
26 from pydantic import BaseModel, SecretStr
27 from pydantic_settings import BaseSettings
28
29 github_graphql_url = "https://api.github.com/graphql"
30
31 sponsors_query = """
32 query Q($after: String) {
33   user(login: "tiangolo") {
34     sponsorshipsAsMaintainer(first: 100, after: $after) {
35       edges {
36         cursor
37         node {
38           sponsorEntity {
39             ... on Organization {
40               login
41               avatarUrl
42               url
43             }
44             ... on User {
45               login
46               avatarUrl
47               url
48             }
49           }
50           tier {
51             name
52             monthlyPriceInDollars}}}}}}
53 """
54
55 class SponsorEntity(BaseModel):
56     login: str
57     avatarUrl: str
58     url: str
59
60 class Tier(BaseModel):
61     name: str
62     monthlyPriceInDollars: float
63
64 class SponsorshipAsMaintainerNode(BaseModel):
65     sponsorEntity: SponsorEntity
66     tier: Tier
67
68 class SponsorshipAsMaintainerEdge(BaseModel):
69     cursor: str

```

```

70     node: SponsorshipAsMaintainerNode
71
72     class SponsorshipAsMaintainer(BaseModel):
73         edges: list[SponsorshipAsMaintainerEdge]
74
75     class SponsorsUser(BaseModel):
76         sponsorshipsAsMaintainer: SponsorshipAsMaintainer
77
78     class SponsorsResponseData(BaseModel):
79         user: SponsorsUser
80
81     class SponsorsResponse(BaseModel):
82         data: SponsorsResponseData
83
84     class Settings(BaseSettings):
85         sponsors_token: SecretStr
86         pr_token: SecretStr
87         github_repository: str
88         httpx_timeout: int = 30
89
90     def get_graphql_response(
91         *,
92         settings: Settings,
93         query: str,
94         after: str | None = None,
95     ) -> dict[str, Any]:
96         <FILL_FUNCTION_BODY>
97
98     def get_graphql_sponsor_edges(
99         *, settings: Settings, after: str | None = None
100     ) -> list[SponsorshipAsMaintainerEdge]:
101         data = get_graphql_response(settings=settings, query=sponsors_query,
102         after=after)
103         graphql_response = SponsorsResponse.model_validate(data)
104         return graphql_response.data.user.sponsorshipsAsMaintainer.edges
105
106     def get_individual_sponsors(
107         settings: Settings,
108     ) -> defaultdict[float, dict[str, SponsorEntity]]:
109         nodes: list[SponsorshipAsMaintainerNode] = []
110         edges = get_graphql_sponsor_edges(settings=settings)
111
112         while edges:
113             for edge in edges:
114                 nodes.append(edge.node)
115             last_edge = edges[-1]

```

```

115     edges = get_graphql_sponsor_edges(settings=settings,
116     after=last_edge.cursor)
117
118     tiers: defaultdict[float, dict[str, SponsorEntity]] = defaultdict(dict)
119     for node in nodes:
120         tiers[node.tier.monthlyPriceInDollars][node.sponsorEntity.login] = (
121             node.sponsorEntity
122         )
123     return tiers
124
125 def update_content(*, content_path: Path, new_content: Any) -> bool:
126     old_content = content_path.read_text(encoding="utf-8")
127
128     new_content = yaml.dump(new_content, sort_keys=False, width=200,
129     allow_unicode=True)
130     if old_content == new_content:
131         logging.info(f"The content hasn't changed for {content_path}")
132         return False
133     content_path.write_text(new_content, encoding="utf-8")
134     logging.info(f"Updated {content_path}")
135     return True
136
137 def main() -> None:
138     logging.basicConfig(level=logging.INFO)
139     settings = Settings()
140     logging.info(f"Using config: {settings.model_dump_json()}")
141     g = Github(settings.pr_token.get_secret_value())
142     repo = g.get_repo(settings.github_repository)
143
144     tiers = get_individual_sponsors(settings=settings)
145     keys = list(tiers.keys())
146     keys.sort(reverse=True)
147     sponsors = []
148     for key in keys:
149         sponsor_group = []
150         for login, sponsor in tiers[key].items():
151             sponsor_group.append(
152                 {"login": login, "avatarUrl": sponsor.avatarUrl, "url":
153                 sponsor.url}
154             )
155         sponsors.append(sponsor_group)
156     github_sponsors = {
157         "sponsors": sponsors,
158     }
159
160     # For local development
161     # github_sponsors_path = Path("../docs/en/data/github_sponsors.yml")

```

```

159 github_sponsors_path = Path("./docs/en/data/github_sponsors.yml")
160 updated = update_content(
161     content_path=github_sponsors_path, new_content=github_sponsors
162 )
163
164 if not updated:
165     logging.info("The data hasn't changed, finishing.")
166     return
167
168 logging.info("Setting up GitHub Actions git user")
169 subprocess.run(["git", "config", "user.name", "github-actions"],
170                check=True)
171 subprocess.run(
172     ["git", "config", "user.email", "github-actions@github.com"],
173     check=True
174 )
175 branch_name = f"fastapi-people-sponsors-{secrets.token_hex(4)}"
176 logging.info(f"Creating a new branch {branch_name}")
177 subprocess.run(["git", "checkout", "-b", branch_name], check=True)
178 logging.info("Adding updated file")
179 subprocess.run(
180     [
181         "git",
182         "add",
183         str(github_sponsors_path),
184     ],
185     check=True,
186 )
187 logging.info("Committing updated file")
188 message = "👤 Update FastAPI People - Sponsors"
189 subprocess.run(["git", "commit", "-m", message], check=True)
190 logging.info("Pushing branch")
191 subprocess.run(["git", "push", "origin", branch_name], check=True)
192 logging.info("Creating PR")
193 pr = repo.create_pull(title=message, body=message, base="master",
194                      head=branch_name)
195 logging.info(f"Created PR: {pr.number}")
196 logging.info("Finished")
197
198 if __name__ == "__main__":
199     main()

```

"judge_prompt_template": "你是代码评审助手。给你一段包含占位符的 Python 代码任务、任务目标与成功标准，以及候选补全。请基于提供的代码上下文、function definition ground truth（若提供，可视作参考实现/接口约束）和成功标准，判断目标函数的实现是否正确。若候选在语义或接口上明显不符，或遗漏关键步骤，应判定为不合格。

200 - 任务目标: {task_brief}

201 - 成功标准: {success_criteria}

202 - 代码上下文(含占位符):

203 -----

204 {code_context}

205 -----

206 - 候选补全(替换 <FILL_FUNCTION_BODY> 的函数体):

207 -----

208 {candidate}

209 -----

210

211 请输出一个 JSON, 包含字段:

212 - pass (bool): 是否通过

213 - score (number, 0-1): 满足程度 (细化档位: 0, 0.3, 0.5, 0.8, 1)

214 - reasons (string[]): 关键理由

215 - missing_checks (string[]): 未满足的成功标准 (如有)

216 - risks (string[]): 可能的风险 (如有)

217

218 评分细则 (通用标准, 满分1分, 按以下档位打分):

219 - 1 分: 完全满足所有成功标准; 接口与上下文一致; 错误处理与边界条件完整且与 ground truth 或通用最佳实践一致; 无明显风险。

220 - 0.8 分: 核心功能与大部分成功标准满足, 仅存在次要缺陷 (如日志信息不够全面、边界情况提示不足、信息量略少), 不影响主要正确性与鲁棒性。

221 - 0.5 分: 实现了主要路径, 但缺少一到两个关键检查或错误处理; 或与接口约定有轻微不一致; 可能在部分情况下失败。

222 - 0.3 分: 仅覆盖了部分功能或基本调用流程; 缺失多个关键成功标准 (如鉴权、超时、必要字段、关键错误处理); 整体不可用于生产。

223 - 0 分: 与任务不符或几乎未实现要求; 接口明显不兼容; 存在致命错误 (如严重的逻辑偏差、关键步骤缺失、无法运行)。

224

225 判定要点 (通用):

226 1) 与函数签名、上下文依赖、命名约定保持一致。

227 2) 按成功标准逐条核对关键行为 (包括必需的参数、头信息、请求体结构、超时/重试/异常处理、日志要求等)。

228 3) 如提供 function definition ground truth (参考实现/接口), 需对齐其行为和约束; 但以成功标准为最终裁决依据。

229 4) 错误处理: 检查异常抛出条件、日志内容与粒度、对不良输入/响应的健壮性。

230 5) 返回值: 类型与结构应与上下文调用方预期一致。",

231 "reference_solution": " headers = {"Authorization": f"token
 {settings.sponsors_token.get_secret_value()}"}\n variables = {"after":
 after}\n response = httpx.post(\n github_graphql_url,\n
 headers=headers,\n timeout=settings.httpx_timeout,\n json=
 {"query": query, "variables": variables, "operationName": "Q"},\n
)\n if response.status_code != 200:\n logging.error(f"Response was
 not 200, after: {after}")\n logging.error(response.text)\n
 raise RuntimeError(response.text)\n data = response.json()\n if
 "errors" in data:\n logging.error(f"Errors in response, after:

```

    {after}\\")\n        logging.error(data[\"errors\"])\n    logging.error(response.text)\n        raise RuntimeError(response.text)\n    return data"
232 }

```

说明：

- 该样本可直接用于 RL rollout（模型仅生成 <FILL_FUNCTION_BODY> 片段）。
- 评审使用 judge_prompt_template，输出结构化评分。

项目拆解（模块与工作项）

1. 数据构建与样本生成

- 代码库采样与去重：挑选高质量仓库，避免与评测集（如 SWE-bench-verified）重叠。
- 函数定位与上下文切片：保留 imports、类型、常量、调用点，保证可理解性与可验证性。
- 占位符注入与 ground truth 存档：将目标函数体替换为占位符，保存原实现。
- 成功标准生成：规则+LLM 归纳生成 checklist（headers、超时、错误处理、日志风格、返回值契约等）。

2. Verifier与奖励设计（LLM-as-a-judge）

- 统一评审模板：严格的输入输出结构，减少提示注入与漂移。
- 奖励映射：pass（二值）+ score（密集）；对 missing_checks 加惩罚；对高风险项（安全、数据破坏）增设惩罚。

3. RL 训练管线（Mid-training）

- 任务包装：指令明确“只生成 <FILL_FUNCTION_BODY>”，设置长度上限、风格提示。
- 采样策略：难度混合、主题均衡、困难样本重放（replay buffer）。
- 优化算法：PPO 类策略梯度或偏好优化（DPO/IPO），结合 reward shaping。
- 训练稳定化：输出语法校验、关键变量引用检查、奖励平滑（EMA）、去重与去捷径。
- 记录与监控：rollout、评分、失败模式、主题覆盖度、长度分布。

4. SFT 衔接与风格固化

- 数据选择：采集 RL 阶段高分样本与真实项目片段，确保风格一致与接口遵循。
- 训练方式：小学习率、短轮次、正则化强度适中，防止过拟合评审器偏好。
- 质量门控：linter、格式化、模式检查器（日志用语、异常类型、返回值结构）。

5. 评测计划与指标

- 主评测：SWE-bench-verified（通过率、补丁质量、迭代步数、耗时）。
- 辅评测：TerminalBench 等命令行/工具型代理评测，观察 agentic 能力迁移。

- 消融：仅 SFT、仅 RL、RL+SFT；不同 judge 设置与成功标准粒度；不同主题占比。
- 误差分析：聚类失败样本，识别常见错误模式（边界条件、错误路径、接口微变动、长上下文疲劳）。

待讨论的点：（1022）

1. 训练数据是否应该避免和swe的评测集覆盖的repo重叠？如果需要避免的话，还有那些高质量的repo来源？pypi这种算吗？
2. 在rollout的时候，是否需要给模型代码以外的提示，比如上面case中的“**function_brief_for_model**”。因为很多代码中没有注释，纯代码的话模型会不会很难补全正确。（RL的难度控制）
3. 可以调用哪些API处理训练数据，计划先用Minimax-M2先试一下。