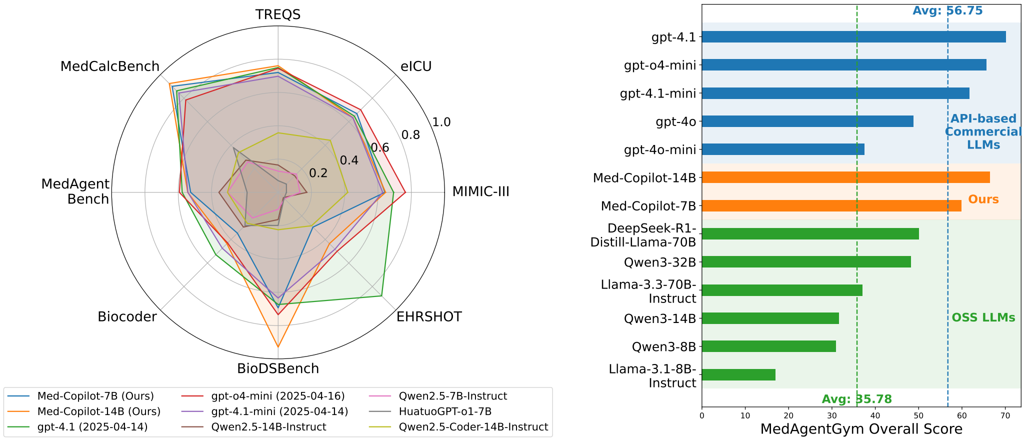
`MedAgentGym`: Training LLM Agents for Code-Based Medical Reasoning at Scale

Ran Xu[§], **Yuchen Zhuang**[†], **Yishan Zhong**[†], **Yue Yu**[†], **Xiangru Tang**[∝], **Hang Wu**[†],
May D. Wang[†], **Peifeng Ruan**[◇], **Donghan Yang**[◇], **Tao Wang**[◇],
Guanghua Xiao[◇], **Carl Yang**[§], **Yang Xie**[◇], **Wenqi Shi**[◇]
[§]Emory University, [†]Georgia Tech, [∝]Yale University, [◇]UT Southwestern Medical Center

Abstract

We introduce `MedAgentGym`, a scalable and interactive training environment designed to enhance coding-based biomedical reasoning capabilities in large language model (LLM) agents. `MedAgentGym` comprises $72,413$ task instances across 129 categories derived from 12 authentic real-world biomedical scenarios. Tasks are encapsulated within executable sandbox environments, each featuring detailed task specifications, interactive feedback mechanisms, verifiable ground truth annotations, and scalable training trajectory generation. Extensive benchmarking of 29 LLMs reveals substantial performance disparities in biomedical data science between commercial and open-source LLMs. Leveraging efficient multi-threaded and multi-turn trajectory sampling in `MedAgentGym`, `Med-Copilot` achieves performance gains of $+43.02\%$ and $+45.28\%$ from offline and online reinforcement learning, respectively, demonstrating `MedAgentGym` as an effective training ground while establishing itself as a cost-effective, privacy-preserving alternative competitive with proprietary LLMs (`gpt-4o`). By offering a unified execution environment with a comprehensive benchmark and accessible, extensible training resources, `MedAgentGym` delivers an integrated platform to develop LLM-based coding assistants for advanced biomedical data science.



(a) Benchmark evaluations of SOTA and domain-specific LLMs across eight environments in `MedAgentGym`.

(b) Overall score of `MedAgentGym`.

Figure 1: An overview of leader-board evaluation on `MedAgentGym`. Substantial performance disparities between proprietary and OSS LLMs highlight the critical need for continued development of privacy-preserving, affordable LLMs to enhance complex code-based medical reasoning.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: The Second Workshop on GenAI for Health: Potential, Trust, and Policy Compliance.

1 Introduction

The exponential growth of healthcare data has fundamentally transformed modern biomedical research, intensifying the need for integration of advanced computational methods with medical domain expertise [93, 42]. Biomedical researchers routinely face data science challenges that demand both medical data analysis knowledge and programming proficiency, such as querying large-scale databases, conducting statistical analyses, processing genomic sequences, and building predictive models from electronic health records (EHRs) [50, 35, 92]. While recent advances in large language models (LLMs) have demonstrated significant capabilities in advanced reasoning [52, 14], including code generation [8] and scientific discovery [74, 80, 102], it remains challenging to translate real-world biomedical data science requirements into executable computational solutions [90, 89].

Developing effective biomedical coding agents poses unique challenges beyond knowledge-intensive medical reasoning [86, 87] and general-purpose code generation [107, 27]. Within biomedical research and clinical practice, direct deployment of proprietary LLMs remains infeasible due to strict privacy requirements and prohibitive operational costs [45, 69], whereas OSS LLMs exhibit substantial deficiencies in biomedical coding capabilities (Figure 1). Mitigating this performance disparity calls for addressing two infrastructure gaps: (1) *comprehensive, code-centric biomedical reasoning benchmarks* to diagnose agent limitations and support rigorous, reproducible evaluation; and (2) *specialized, interactive training environments* to develop the complex reasoning and robust coding capabilities required for real-world biomedical data science.

In this study, we introduce `MedAgentGym`, a scalable and agentic training environment designed to systematically enhance the coding-centric reasoning capabilities of LLM agents for biomedical data science workflows. Grounded in diverse real-world biomedical scenarios, `MedAgentGym` provides:

- **Comprehensive suite of code-centric biomedical reasoning tasks.** `MedAgentGym` encompasses 72,413 biomedical *coding-centric* instances across 129 categories grounded in 12 real-world biomedical scenarios[1]. We standardize a rich collection of biomedical data science tasks as executable problems with verifiable ground truth, spanning structured medical information retrieval, numerical clinical reasoning, bioinformatics, and machine learning (ML) modeling. Tasks incorporate diverse data modalities, including EHR tables, clinical notes, genomics, drugs, and biological sequences, which require medical domain-specific reasoning capabilities.
- **Scalable and interactive training infrastructure.** `MedAgentGym` provides an optimized, user-friendly environment to accelerate agent training. Each instance is encapsulated within *executable, isolated, and reproducible* Docker environments with pre-install dependencies, supporting multi-threading, parallel execution, and sequential sampling. `MedAgentGym` ensures efficient trajectory collection and facilitate large-scale automated evaluation compatible with diverse agent scaffolds.
- **Extensive benchmarking and effective agent training for biomedical data science.** Through an extensive benchmark of 29 proprietary and open-source LLMs, we identify critical deficiencies in biomedical data analysis and predictive modeling. `MedAgentGym` effectively strengthens agentic training: `Med-Copilot-7B` achieves gains of +43.02% and +45.28% through offline and online reinforcement learning (RL), respectively, and performs comparably to `gpt-4o` on both in- and out-of-distribution tasks.

2 Related Works

Coding-Centric Reasoning in Biomedical Data Science. Most existing medical benchmarks primarily evaluate LLMs on knowledge-intensive, narrative reasoning [25, 53, 82]. Although several efforts target isolated biomedical algorithmic tasks [76, 16, 90] or simulate portions of clinical workflows [64, 38, 37], they do not capture a complete set of tasks in the full end-to-end lifecycle of biomedical data science, from data extraction [35, 61] to model development [92, 85]. Complementing these benchmarks, `MedAgentGym` emphasizes computation- and coding-intensive tasks that require LLM agents to retrieve, transform, analyze, and compute biomedical data while generating and executing code with pre-installed biomedical libraries and dependencies to produce verifiable solutions.

---

[1] We emphasize that `MedAgentGym` mainly focuses on computational *code generation* for biomedical reasoning, rather than traditional medical coding systems [72] such as ICD-9 or ICD-10.

Table 1: Summary of related medical reasoning benchmarks with data resources and task execution environments. `MedAgentGym` is among the first publicly available training environments for improving LLM-based coding agents in biomedicine, uniquely integrating *executable environments, interactive feedback, and task-isolated run-time facilities* for coding-based medical reasoning. "DS", "Bioinfo", and "ML" denote "data science", "bioinformatics" (biostatics), and "machine learning", respectively.

	Domain		Task				Environment & Facility				Scale (#Instances)		
Datasets (↓)	QA	Coding	Database	DS	Bioinfo	ML	Execution	Interaction	Isolation	Training	# Train	# Test	# Traj.
MedMCQA [53]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	3K	4.18K	✗
MedQA [24]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	11.4K	1.27K	✗
PubMedQA [25]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	450	500	✗
BioASQ [82]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	745	140	✗
MedAgentsBench [77]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	–	862	✗
MIRAGE [98]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	–	7.66K	✗
HealthBench [1]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	–	5K	✗
EHRSQL [35]	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	15.5K	1.73K	✗
MedCalcBench [30]	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	10.1K	1.05K	✗
MedAgentBench [23]	✗	✓	✗	✓	✗	✗	✓	✓	✗	✗	–	300	✗
BioCoder [76]	✗	✓	✗	✓	✓	✗	✓	✗	✓	✗	–	1.24K	✗
BioDSBench [90]	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗	–	128	✗
EHRSHOT [92]	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	–	15	✗
`MedAgentGym` (Ours)	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	**59.2K**	**13.2K**	**6.7K**

Scalable and Interactive Training Environment for Biomedical Coding Agents. Agentic RL [14, 65, 67] shifts LLM post-training from passive sequence generation to autonomous agents operating in complex, dynamic settings, including medical reasoning [97, 22, 3, 34, 94, 83]. Within such a framework, agents interact iteratively with their environment, receiving observations and executing actions, while the environment returns reward signals and state updates [91, 5, 66, 49]. However, most biomedical reasoning and data science benchmarks (Table 1) are single-pass evaluations without executable environments or agent-level interaction signals [108, 1, 95]. In contrast, `MedAgentGym` uniquely provides an executable and interactive biomedical coding environment covering comprehensive range of tasks. It also supports efficient multi-turn trajectory sampling through multi-threaded rollouts, thus enabling scalable and systematic improvement via agentic fine-tuning beyond prompting [70, 17].

3 `MedAgentGym`

3.1 Problem Formulation

We formulate coding-based medical reasoning as a structured problem-solving task: given a problem description $x \in \mathcal{X}$, the goal is to generate a code snippet $c \in \mathcal{C}$ that produces an output $y \in \mathcal{Y}$. Each instance (x, y) is paired with a ground truth output y^*, and the correctness is verified using $\mathcal{E} : \mathcal{C} \times \mathcal{Y} \to \{0, 1\}$, where $\mathcal{E} = \mathbb{I}(y = y^*)$. Existing medical reasoning datasets typically provide only question-answer pairs (x, y^*) without code solutions c or only include a single predefined code solution per task. To address this constraint, `MedAgentGym` enables scalable generation and sampling of multiple coding trajectories $c^{(0)}, c^{(1)}, \cdots, c^{(k)}$ with corresponding executions $y^{(0)}, y^{(1)}, \cdots, y^{(k)}$ through parallel execution of LLM agents. Each trajectory is either single-turn or multi-turn, depending on task complexity and user requirements. Crucially, `MedAgentGym` captures both *positive* trajectories $\{c^{(i)} | y^{(i)} = y^*\}$ that succeed and *negative* trajectories $\{c^{(i)} | y^{(i)} \neq y^*\}$ including error messages as learning signals. The details of the data and the interactive environment are in section 3.2 and section 3.3.

3.2 Data Construction: From Individual Datasets to Comprehensive and Unified Benchmark

Task and Data Identification. `MedAgentGym` focuses on verifiable medical reasoning tasks that benefit from code-based solutions. *Clinically*, we prioritize tasks originating from real-world healthcare settings and validated by a multidisciplinary panel of healthcare experts. For example, `MedAgentGym` involves MIMIC-III and eICU in EHRSQL [35] collected from 222 hospital staff members and annotated by human programmers. *Computationally*, we integrate diverse biomedical coding tasks,

Table 2: Dataset statistics for `MedAgentGym` and its lightweight subset designed for leaderboard evaluation. *For open-ended tasks without explicit ground truth (*e.g.*, ML coding), we follow standard RL settings by using the same dataset for training and evaluation. $^{+}$Only the test set is utilized for external evaluation, while the corresponding training data remains accessible. Similarly, we provide a subset of training data in leaderboard for efficiency and convenience, without restricting the incorporation of supplementary training resources.

	Data Sources				**Tasks (all)**				**Tasks (leader-board)**		
Dataset	Type	#Patients	#Table	#Elements	Category	#Train	#Test	#Total	#Train	#Test	#Total
Training and Internal Validation (In-Distribution)											
MIMIC-III [28, 35]	Tabular	<1K	17	1.4M	9	9,318	1,122	10,440	552	581	1,133
eICU [55, 35]	Tabular	<1K	10	1.5M	9	6,213	611	6,824	559	610	1,169
TREQS [84]	Tabular	100	5	2.5M	4	8,988	996	9,984	897	995	1,892
MedCalcBench [30]	Text	1K	–	–	55	10,053	1,047	11,100	1,005	1,046	2,051
MedAgentBench [23]	Tabular	100	–	700K	10	433	109	542	239	59	298
BioCoder [76]	Text	–	–	–	8	981	157	1,138	981	156	1,137
EHRSHOT [92]	Tabular	63K	31	1.2M	15	15	15	15*	15	15	15*
BioDSBench [90]	Text	–	–	–	12	50	49	99	50	49	99
`MedAgentGym` (Internal)	–	65K	63	7.3M	113	36,036	4,106	40,142	4,283	3,511	7,794
External Validation (Out-of-Distribution)$^{+}$											
EHR-SeqSQL [61]	Tabular	<1K	17	1.4M	4	18,950	7,913	26,863	1,000	500	1,500
EHRCon [32]	Tabular	46K	13	–	3	3,229	976	4,205	1,000	500	1,500
MIMIC-Extract [85]	Tabular	35K	4	35K	3	3	3	3*	3	3	3*
N-PowerAI [60]	Text	–	–	–	6	960	240	1200	960	240	1200
`MedAgentGym` (External)	–	82K	34	1.4M	16	23,142	9,132	32,271	2,963	1,243	4,203
Overall											
`MedAgentGym`	–	**146K**	**80**	**7.4M**	**129**	**59,175**	**13,238**	**72,413**	**7,243**	**4,754**	**11,997**

ranging from *structured medical information retrieval* to *open-ended biomedical research*, ensuring comprehensive coverage and task diversity.

Verifiable Instances Preparation. To standardize tasks across various sources, each instance in `MedAgentGym` is structured with: (1) a problem description, (2) verifiable ground-truth outputs, and (3) optional data resources (*e.g.*, EHRs). Additionally, standardized system and user prompts are designed to initiate the problem-solving process (see appendix G). `MedAgentGym` is highly flexible, easily accommodating new tasks that include clear descriptions and verifiable ground-truth outputs. For coding-centric tasks that provide only reference code implementations (*e.g.*, BioCoder [76]), we validate task correctness based on the execution output of these reference solutions, generating definitive output signatures. This transformation is necessary because multiple valid code implementations may yield identical execution results, making the execution outcome–rather than the code itself–a more reliable and consistent verification signal. For tasks involving additional data resources (*e.g.*, EHRSQL [35]), we include metadata on data access and sources. Additional task-specific preparation details are documented in appendix C.

Data Statistics. `MedAgentGym` is a unified training environment built upon a large-scale, high-quality dataset comprising approximately 72,000 task instances across 1,293 categories from 12 real-world biomedical scenarios. Notably, with `MedAgentGym`, we collect large-scale agent trajectories to support coding agent development (section 5). To ensure reproducible and robust evaluation, we define clear train/test splits, separate internal and external validation sets, and perform n-gram ($n = 10$) string match to eliminate the data contamination issue. Table 2 provides statistics for `MedAgentGym`. To accommodate diverse research needs, we offer two versions of `MedAgentGym`: (1) a comprehensive, full-scale dataset for extensive exploration and detailed analysis, and (2) a balanced, lightweight subset for efficient leaderboard training and evaluation.

3.3 Coding Environment: From Static Benchmark to Interactive Interface

Isolated and Executable Environment. To ensure robust and reproducible coding-based medical reasoning, `MedAgentGym` provides isolated executable environments through Docker containers tailored to each task. These containers come pre-installed with all required dependencies, including specialized biomedical packages (*e.g.*, `AlignIO` in BioCoder [76]), facilitating reliable task execution. To address critical data safety concerns, each Docker environment guarantees: (1) *environmental integrity*, where isolation prevents contamination or data corruption potentially caused by LLM-generated code, preserving both the computational environment and the underlying data systems [100];

(2) *medical data security*, where secure containerization enforces compliance with medical data usage policies, safeguarding sensitive patient information. Additionally, `MedAgentGym` supports extensive flexibility for integrating new tasks, where users can define customized Docker environments through configuration files. If certain packages are not initially available, a terminal tool allows LLM agents to dynamically install the required dependencies within their isolated environments.

Interactive Feedback. `MedAgentGym` incorporates interactive feedback mechanisms, effectively bridging LLMs with coding interpreters: (1) *robust parsing:* To begin, the output generated by LLMs is formatted in structured JSON, facilitating straightforward parsing and code execution. In cases of execution errors, iterative JSON regeneration is employed to maximize successful code execution rates. (2) *debugging and error grounding:* Compile-time and runtime error messages are systematically translated into a unified natural language format, making them more accessible to LLMs and significantly improving debugging efficiency and interpretability.

Efficient Trajectory Collection. Each task in `MedAgentGym` is packaged in a reproducible Docker image with built-in support for *multi-threading*, *parallel execution*, and *sequential sampling*. Specifically, we integrate two widely used multi-threading backend engines, `Ray`[2] and `Joblib`[3], to accelerate trajectory sampling. This infrastructure ensures efficient and scalable trajectory collection, supporting both extensive experimentation and systematic evaluation across multiple scenarios.

Plug-and-Play. A key strength of `MedAgentGym` lies in its flexible and modular architecture, which readily supports the integration of new biomedical coding tasks. This inherent extensibility enables `MedAgentGym` to continually adapt to evolving advancements in biomedical sciences and artificial intelligence methodologies. Additionally, its trajectory sampling approach allows the straightforward transformation of traditional, non-executable medical reasoning tasks into coding-based scenarios with verifiable outputs, significantly broadening the scope and complexity of tasks that can be systematically evaluated. Moreover, users can define custom Docker environments through configuration files, and, if specific software packages are initially absent, a built-in terminal tool facilitates dynamic installation within each isolated execution environment, further improving `MedAgentGym` in runtime adaptability and user-friendliness.

4 Evaluating LLMs as Medical Coding Agents with `MedAgentGym`

4.1 Experiments Setup

Tasks and Datasets. Building upon `MedAgentGym`, we train and evaluate `Med-Copilot` on 7,794 *coding-based medical reasoning* tasks from 113 categories across 8 datasets: (1) MIMIC-III [28] and (2) eICU [55] from EHRSQL [35], (3) TREQS [84], (4) MedCalcBench [30], (5) MedAgentBench [23], (6) BioCoder [76], (7) EHRSHOT [92], and (8) BioDSBench [90]. Moreover, we conduct experiments for *out-of-distribution* evaluation on 4,203 tasks from the following 4 datasets: (9) EHR-SeqSQL [61], (10) EHRCon [32], (11) MIMIC-Extract [85], and (12) N-PowerAI [60]. Note that we do not consider general medical question-answering tasks [25, 53, 24], as they are orthogonal to coding-aided reasoning. We include detailed task and dataset information in appendix C.

Baselines. We extensively benchmark the following state-of-the-art LLMs on `MedAgentGym`. (i) *API-based proprietary LLMs*, including gpt-4o-mini [21], gpt-4o [21], gpt-4.1-mini [51], gpt-4.1 [51], gpt-o4-mini [52], and codex-mini [4]; (ii) *OSS LLMs*, including gemma-3 [12], Qwen3 [58], Qwen2.5 [99], Llama-3 [10], Ministral [46], and DeepSeek-R1 [14]; (iii) *coding LLMs*, including codex-mini [4], Qwen2.5-Coder-7B-Instruct, and -14B-Instruct [20]; and (iv) *medical reasoning LLMs* or medical domain-specific LLMs, including medgemma-4b-it (`gemma-3-4b-pt`) [13], HuatuoGPT-o1-7B (`Qwen2.5-7B-Instruct`) [3], m1-7B-23K (`Qwen2.5-7B-Instruct`) [19], MedReason-8B (`Llama-3.1-8B-Instruct`) [94], and Baichuan-M1-14B-Instruct [83]. Additional model details are available in appendix D.

Agent Scaffolds. Following CodeAct [88], we establish a default agent scaffold for systematically evaluating coding-based medical reasoning. Interactions within `MedAgentGym` are modeled as a Partially Observable Markov Decision Process (POMDP), focusing on sampled medical reasoning tasks $p \in \mathcal{P}$. At each timestep t, the agent observes $o_t \in \mathcal{O}$ and selects an action $a_{t+1} \in \mathcal{A}$ based

---

[2] `https://github.com/ray-project/ray`

[3] `https://joblib.readthedocs.io/en/stable/`

Table 3: Test set (zero-shot) results (success rate, SR) of `MedAgentGym`. **Bold** indicates the best result at each scale. †: We only consider Microsoft Azure OpenAI API services due to credentialed health data use agreement. ‡ and $^{\vee}$ denote coding LLMs and medical reasoning LLMs, respectively. Notations are consistent across tables.

Datasets (→) **Baselines (↓) / Metrics (→)**	**MIMIC.** SR	**eICU** SR	**TREQS** SR	**MedCalc.** SR	**MedAgent.** SR	**BioCoder** SR	**BioDS.** SR	**EHRSHOT** Acc	**Avg.** Score
API-based Proprietary LLMs†									
`gpt-4o-mini (2024-07-28)` [21]	35.97	16.57	38.39	73.11	40.38	30.12	57.35	7.84	37.47
`gpt-4o (2024-08-06)` [21]	43.04	43.44	53.47	73.97	54.23	30.12	58.16	33.53	48.75
`gpt-4.1-mini (2025-04-14)` [51]	62.79	63.44	69.75	84.36	54.23	47.46	63.47	48.28	61.72
`gpt-4.1 (2025-04-14)` [51]	69.36	64.75	**74.97**	**86.23**	57.63	**52.95**	67.35	**87.93**	**70.15**
`gpt-o4-mini (2025-04-16)` [52]	**76.45**	**70.16**	74.47	78.45	**59.32**	42.94	**73.47**	50.07	65.67
‡`codex-mini (2025-05-16)` [4]	67.30	64.75	74.57	82.49	58.76	48.78	67.64	58.76	65.38
OSS (Base Size): < 10B parameters									
`Qwen3-1.7B` [58]	20.12	10.62	15.08	46.24	16.95	15.38	6.12	1.87	16.55
`Qwen3-4B` [58]	27.23	30.77	28.85	52.80	15.25	19.16	20.41	23.85	27.29
`gemma-3-4b-it` [12]	27.36	29.10	24.52	42.49	18.64	17.95	8.16	4.37	21.57
$^{\vee}$`medgemma-4b-it` [13]	15.51	13.11	14.85	41.89	17.62	26.74	17.82	1.33	18.61
`Qwen3-8B` [58]	**29.08**	**34.53**	**37.37**	**54.59**	20.34	20.51	**24.49**	**25.71**	**30.83**
`Qwen2.5-7B-Instruct` [99]	13.08	15.57	12.76	25.91	**30.36**	21.79	10.20	5.42	17.43
`Llama-3.1-8B-Instruct` [10]	16.67	25.00	19.17	27.53	16.95	18.59	9.19	2.36	16.97
`Ministral-8B-Instruct-2410` [46]	16.70	14.92	25.39	49.81	22.03	23.72	12.24	7.79	22.27
‡`Qwen2.5-Coder-7B-Instruct` [20]	9.12	10.66	15.63	24.62	18.75	10.60	17.24	10.55	14.65
$^{\vee}$`HuatuoGPT-o1-7B` [3]	4.99	7.04	7.04	38.05	18.64	28.21	19.88	5.03	16.11
$^{\vee}$`m1-7B-23K` [19]	6.88	9.56	7.04	28.24	9.32	20.26	14.71	0.00	12.00
$^{\vee}$`MedReason-8B` [94]	9.12	9.51	9.15	43.31	21.46	**31.42**	17.42	3.88	18.16
OSS (Large Size): 10 - 30B parameters									
`Qwen3-14B` [58]	31.50	31.97	30.05	**61.38**	22.03	22.60	**26.53**	26.77	31.60
`Qwen2.5-14B-Instruct` [99]	17.21	14.07	16.43	27.40	**35.59**	**29.49**	16.33	4.45	20.12
`DeepSeek-R1-Distill-Qwen-14B` [14]	35.12	38.52	32.96	48.09	32.20	21.29	24.49	11.39	30.51
‡`Qwen2.5-Coder-14B-Instruct` [20]	**41.82**	**44.26**	**35.78**	33.75	30.42	26.28	22.45	**28.37**	**32.89**
$^{\vee}$`Baichuan-M1-14B-Instruct` [83]	4.50	12.19	7.36	1.82	21.46	16.34	17.42	0.00	10.14
OSS (XL Size): > 30B parameters									
`Qwen3-32B` [58]	52.48	60.95	53.82	63.82	45.93	32.67	28.57	47.29	48.19
`Qwen2.5-32B-Instruct` [99]	54.56	45.41	62.81	69.96	40.67	27.45	22.45	18.13	42.68
`QwQ-32B` [59]	62.31	56.72	**66.15**	67.69	**47.46**	**42.31**	14.29	**55.05**	**51.50**
`DeepSeek-R1-Distill-Qwen-32B` [14]	62.18	58.36	65.82	60.14	43.56	28.66	26.53	31.17	47.05
`Llama-3.3-70B-Instruct` [10]	39.93	25.08	24.98	**84.99**	39.40	27.55	24.49	29.93	37.04
`DeepSeek-R1-Distill-Llama-70B` [14]	**64.59**	**64.92**	56.98	76.96	28.81	32.05	**42.86**	33.42	50.07

on interaction history. We define four primary action types: (a) `request_info`: retrieve relevant data from sources such as EHRs; (b) `terminal`: manage dependencies or local files within isolated Docker environments. (c) `code_execution`: execute code generated by LLMs through an integrated interpreter; and (d) `debugging:` translate code execution errors into natural language explanations enriched with detailed error information for LLM comprehension. See appendix E for evaluation and implementation details.

4.2 Results: Benchmarking LLMs and Reasoning Models with `MedAgentGym`

Table 3 benchmarks the state-of-the-art LLMs on `MedAgentGym`. We summarize key observations from our zero-shot leaderboard evaluation as follows: ◇ **Significant Performance Gap Between Commercial API-based and OSS LLMs.** Commercial API-based LLMs substantially outperform OSS models across all task categories. This evident performance gap highlights the *critical need for continued development* of lightweight OSS LLMs that match commercial performance while addressing real-world privacy and cost constraints. ◇ **Task-Specific Performance Variations between Structured and Open-ended Medical Tasks.** LLMs consistently perform better on structured tasks (*e.g.*, database queries, medical calculations) compared to open-ended tasks requiring advanced coding and reasoning (*e.g.*, data analysis, outcome prediction). ◇ **Suboptimal Outcomes in Dedicated Coding and Medical Domain-Specific LLMs.** Both coding-specialized LLMs and domain-specific medical reasoning models deliver suboptimal performance, revealing that *coding-based medical reasoning represents a unique capability* not adequately captured by specialization in either coding or medical reasoning.

5 `Med-Copilot`: Training LLMs as Coding Agents for Medical Reasoning

In this section, we leverage `MedAgentGym` to systematically enhance lightweight OSS LLMs as proficient coding agents (`Med-Copilot`) for medical reasoning. We first explore a two-stage fine-tuning

Table 5: `Med-Copilot` performance on `MedAgentGym` finetuned with sampled trajectories.

Datasets (→)	**MIMIC-III**	**eICU**	**TREQS**	**MedCalc.**	**MedAgent.**	**BioCoder**	**BioDS.**	**EHRSHOT**	**Avg.**	Δ
Base (↓) / Metrics (→)	SR	SR	SR	SR	SR	SR	SR	Acc	Score	
`Qwen2.5-7B-Instruct`	13.08	15.57	12.76	25.91	30.36	21.79	10.20	5.42	16.89	–
+SFT	57.83	61.48	72.66	89.06	50.85	28.33	55.10	15.62	53.87	*(+36.98)*
+DPO	64.13	66.91	72.02	90.06	52.54	34.62	69.39	29.55	59.90	*(+43.02)*
+PPO	66.10	67.25	**73.88**	74.52	51.33	32.71	65.47	32.40	57.96	*(+41.07)*
+GRPO	**68.21**	**68.73**	70.50	**92.33**	**55.87**	**37.40**	**71.11**	**33.18**	**62.17**	*(+45.28)*
`Qwen2.5-14B-Instrust`	17.21	14.07	16.43	27.40	35.59	29.49	16.33	4.45	20.12	–
+SFT	61.45	62.46	76.38	94.36	52.54	39.80	89.80	34.58	63.92	*(+43.80)*
+DPO	64.54	63.52	76.08	92.45	54.32	43.56	92.96	43.56	66.37	*(+46.25)*
+PPO	67.55	68.53	**78.32**	94.86	53.22	45.88	91.33	56.79	69.56	*(+49.44)*
+GRPO	**68.78**	**69.34**	76.84	**95.81**	**57.41**	**49.32**	**94.78**	**59.05**	**71.42**	*(+51.30)*

framework (section 5.1), followed by a detailed analysis of model scaling behaviors (section 5.2). We then introduce a self-improvement strategy to further boost agent performance (section 5.3) and conduct additional analysis on model generalization, ablation, and error patterns (section 5.4).

5.1 Two-Stage Fine-tuning with Trajectory Sampling

Training Setup. We select `Qwen-2.5-Instruct-7B` and `-14B` [99] as our backbones. To enable effective evaluation within `MedAgentGym`, we utilize a consistent CodeAct-style scaffold, allowing LLM agents to iteratively reason and refine biomedical code through interactive environment feedback. Detailed training setups, including hyperparameters, are provided in appendix E.

Table 4: Trajectory Composition (%).

Actions (→)	**request info**	**terminal**	**code**	**debug**
MIMIC-III	71.07	0	28.84	0.08
eICU	72.17	0	27.13	0.70
TREQS	64.27	0	35.54	0.19
MedCalc.	0	0	74.91	25.09
Structured	**51.88**	**0**	**41.61**	**6.52**
MedAgent.	0	0	100	0
BioCoder	0	0.29	96.11	3.60
BioDS.	0	6.30	87.60	6.90
EHRSHOT	0	0.43	59.43	40.14
Open-ended	**0**	**1.76**	**85.79**	**12.46**
`MedAgentGym`	**32.71**	**0.14**	**57.11**	**10.04**

Trajectory Sampling. `MedAgentGym` facilitates efficient parallel trajectory sampling using `ray` and `joblib` backends. Specifically, we roll out (1) 2,137 successful trajectories using `gpt-4.1-mini` with a temperature of 0 to warm up the fine-tuning for smaller OSS models. Each successful trajectory contains 9.25 turns between the LLM and the code interpreter on average. Although `MedAgentGym` contains extensive training data and allows repeated sampling, the current trajectory count primarily reflects computational budget constraints.

Table 4 details the proportion of action types (section 4.1) in trajectories. Structured tasks predominantly involve data retrieval (over 50%) from databases or resources, complemented by coding and debugging steps. In contrast, open-ended tasks require significant coding and debugging efforts due to diverse question types, often necessitating terminal interactions to install specialized biomedical packages. In addition to 2,137 positive trajectories for SFT, we prepare additional trajectory pairs for RL such as direct preference optimization (DPO), including (2) 1,646 offline pairs sampled from `gpt-4.1-mini`, and (3) 2,939 online pairs. For both types, we use the initial prompt interactions as shared context and contrast successful final codes against intermediate erroneous attempts. We release all 6K trajectories above to accelerate coding agent development.

Two-Stage Fine-Tuning. We benchmark two policy improvement methods: (1) SFT directly mimics high-reward trajectories consisting exclusively of successful outcomes, whereas (2) DPO optimizes the policy by favoring selected responses over rejected ones. We further consider a two-stage fine-tuning framework, initially warming up with SFT and subsequently refining with DPO.

Results: SFT and DPO. Table 5 compares several post-training methods, revealing that simple SFT over successful trajectories significantly boosts performance on structured coding tasks, demonstrating its effectiveness in capturing structured coding patterns. Besides, DPO is particularly beneficial for optimizing open-ended task performance. Although DPO alone slightly underperforms compared to SFT, combining an initial SFT warm-up with subsequent DPO further improves overall results by leveraging their complementary strengths.

Results: PPO and GRPO. As an initial feasibility study, we apply online RL methods, including Proximal Policy Optimization (PPO) [65] and Group Relative Policy Optimization (GRPO) [67], to enable `Med-Copilot` to actively explore tasks and dynamically generate higher-quality training

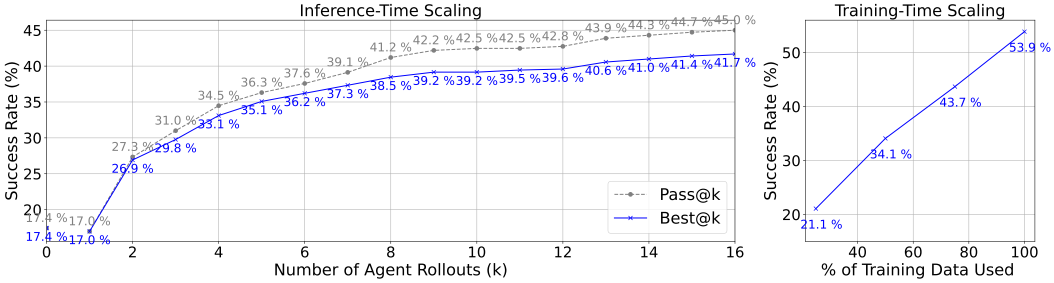


Figure 2: Scalable improvements of LLM agents in `MedAgentGym`. For inference-time scaling, we employ $T = 0$ for the initial rollout and $T = 0.6$ for the rest. For train-time scaling, we set $T = 0$.

data through interaction. The evaluation module of `Med-Copilot` is employed to provide two reward signals: a correctness reward and a format reward, the latter indicating whether the generated output contains code blocks. As shown in Table 5, both PPO and GRPO achieve markedly stronger performance, suggesting enhanced generalization capabilities in diverse biomedical scenarios compared with offline RL.

5.2 Scaling LLM Agent Improvements with `MedAgentGym`

Verifier Training Setup. In addition to directly training coding agents, `MedAgentGym` facilitates the development of an outcome-supervised reward model (ORM) to evaluate generated solutions effectively. Inspired by prior work [6, 54], we formalize the verifier task as predicting the probability that a given trajectory successfully solves a coding task. Formally, we represent a trajectory as an interleaved sequence $\tau = [o_1, a_1, o_2, a_2, \cdots, o_n, a_n]$, $r \in [0, 1]$, where each observation o_k comprises elements such as task descriptions, code execution results, and error feedback. We fine-tune a `Qwen2.5-7B-Instruct` model as a verifier with binary predictions 'YES' (l_y) or 'NO' (l_n), from which we compute success probability: $r = \exp(l_y)/(\exp(l_y) + \exp(l_n))$.

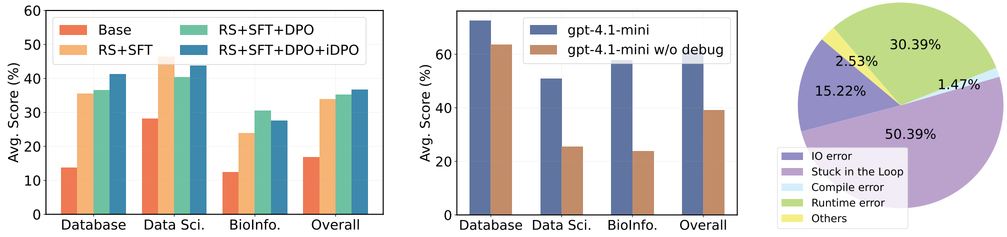
Verifier Training Data. We construct the verifier training dataset by combining two sets of trajectories originally sampled for agent training: (1) *off-policy trajectories*, consisting of 2,742 samples from `gpt-4.1-mini`; and (2) *on-policy trajectories*, comprising 2,939 samples generated by the agent. Combining both on- and off-policy trajectories, we ensure a balanced dataset of successful and unsuccessful trajectories, filtering to fit within a maximum context length of 32k tokens.

Results: Inference and Training-Time Scaling. We introduce two additional evaluation metrics: (1) *Pass@K*: the fraction of tasks solved by at least one trajectory from K sampled attempts; and (2) *Best@K*: the fraction of accurately selects successful trajectories. Figure 2 (left) illustrates the performance scaling with increasing trajectory sampling. Pass@K significantly improves from 17.0% at $K = 1$ to 45.0% at 16, while Best@K shows steady advancement from 17.0% to 41.7%. The relatively small gap between metrics indicates that our trained verifier effectively identifies successful trajectories, unleashing its potential as a reward model for integration into advanced online RL frameworks such as PPO [65] and GRPO [67]. Figure 2 (right) examines agent performance as a function of increased training data volumes in SFT. We observe consistent performance improvements with greater training data availability, suggesting additional computational resources dedicated to sampling further trajectories are likely to yield continued performance gains.

5.3 Model Performance Scaling with Self-improvement

Self-Improvement Training Setup. Beyond expert-generated trajectories, we explore self-improvement by refining the model using its own outputs. We employ rejection sampling fine-tuning (filtered behavior cloning), using the verifier from section 5.2 to score rollouts. We collect 4,298 trajectory pairs, each comprising the highest-scored correct and lowest-scored incorrect trajectories per prompt. Starting from `Qwen2.5-7B-Instruct`, we perform SFT on 1,000 randomly sampled successful trajectories, followed by DPO using eight new rollouts per task and another 4,298 scored pairs. We repeat this DPO step iteratively (iDPO) for further refinement.

Results: Rejection Sampling (RS) and iDPO. Figure 3(a) illustrates consistent performance gains across one SFT stage and two subsequent DPO stages. However, we observe diminishing returns over successive iterations. Initially, rejection sampling SFT significantly boosts performance by effectively capturing successful coding patterns. Subsequent DPO stages show smaller incremental improve-



(a) Self-Improvement (b) Effect of Debug (c) Error Types

Figure 3: Additional studies. For better visualization, we include ML as part of Bio-info. category.

ments, reflecting the model's diminishing exploration space as it tackles increasingly challenging tasks, ultimately converging toward an approximate Nash equilibrium.

5.4 Additional Experimental Analysis

External Evaluation. Table 6 summarizes external evaluation results on `MedAgentGym`. `Med-Copilot` models modestly improve performance on open-ended, reasoning-intensive tasks (*e.g.*, MIMIC-Extract). However, improvements remain limited, with occasional declines in certain domain-specific tasks, indicating challenges in generalizing across specialized biomedical contexts. In particular, incorporating advanced optimization techniques, such as PPO [65] and GRPO [67], can effectively mitigate such limitations and lead to better generalization on unseen, out-of-distribution tasks.

Table 6: External test set results on `MedAgentGym`.

Datasets (→) **Base (↓) / Metrics (→)**	**EHR-SeqSQL** SR	**EHRCon** SR	**MIMIC-Extract** Acc	**N-PowerAI** SR	**Avg.** Score
API-based Proprietary LLMs† *(for reference)*					
`gpt-4o-mini` [21]	50.80	23.20	2.67	16.03	26.03
`gpt-4o` [21]	58.40	35.79	9.82	20.71	34.69
`gpt-4.1-mini` [51]	70.60	52.40	5.62	25.66	43.20
`gpt-4.1` [51]	78.20	**63.00**	10.41	33.53	51.06
`gpt-o4-mini` [52]	**100.00**	51.00	**16.88**	**36.15**	**53.94**
OSS LLMs					
`Qwen3-1.7B` [58]	33.60	17.20	1.90	14.72	16.86
`Qwen3-4B` [58]	44.80	26.20	4.59	19.30	23.72
`Qwen3-8B` [58]	52.00	31.40	6.82	20.12	27.59
$^{\vee}$`HuatuoGPT-o1-7B` [3]	33.25	19.80	2.11	12.45	16.90
`Qwen2.5-7B-Inst` [99]	42.20	27.20	1.34	11.66	20.60
`Med-Copilot` (SFT, 7B)	42.40	28.80	1.95	10.48	20.91
`Med-Copilot` (DPO, 7B)	43.40	23.00	2.14	14.82	20.84
`Med-Copilot` (PPO, 7B)	45.60	24.40	4.30	17.19	22.87
`Med-Copilot` (GRPO, 7B)	61.25	46.80	10.80	27.65	36.63
`Qwen3-14B` [58]	69.00	45.00	9.24	23.59	36.71
`Qwen2.5-Coder-14B-Inst` [20]	52.40	42.00	6.77	28.95	32.53
`Qwen2.5-14B-Inst` [99]	46.40	39.20	4.51	21.57	27.92
`Med-Copilot` (DPO, 14B)	42.20	40.80	2.75	25.89	27.91
`Med-Copilot` (PPO, 14B)	66.40	43.70	7.15	32.01	37.32
`Med-Copilot` (GRPO, 14B)	**72.80**	**56.60**	**14.91**	**43.77**	**47.02**
`R1-Dis-Qwen-14B` [14]	56.00	40.80	2.37	17.60	29.19
`Qwen3-32B` [58]	64.80	54.40	12.17	31.26	42.16

Effect of Interactive Coding. Figure 3(b) shows that removing debugging capabilities significantly decreases model performance across all tasks. Interactive coding mechanism in `MedAgentGym` substantially contributes to successful coding-based medical reasoning by enabling the model to effectively interpret and rectify execution errors.

Error Analysis. Figure 3(c) summarizes common error types encountered by the strongest evaluated LLM, `gpt-4.1`. Loop-related issues dominate, accounting for 50.39% of errors, where agents repeatedly execute the same action in the final turns, indicating difficulty in adapting or exploring alternative strategies. This highlights the need to promote effective exploration and enhance robustness in solving complex medical reasoning tasks. Additional experimental results, including cost analysis and human studies, are available in appendix F.

6 Conclusion

We present `MedAgentGym`, an executable, privacy-preserving, and extensible training environment for scaling code-based biomedical reasoning in LLM agents. With 72K task instances across 129 categories, `MedAgentGym` enables comprehensive benchmarking of 29 proprietary and OSS LLMs for biomedical data science within a modular, decoupled architecture that supports flexibility and extensibility. `Med-Copilot` further demonstrates that systematic training and trajectory sampling with `MedAgentGym` improve coding proficiency for biomedical data science tasks. `MedAgentGym` has the potential to accelerate progress from structured medical information retrieval tasks toward more open-ended computational research questions in clinical research and biomedical discovery.

References

[1] R. K. Arora, J. Wei, R. S. Hicks, P. Bowman, J. Quinonero-Candela, F. Tsimpourlas, M. Sharman, M. Shah, A. Vallone, A. Beutel, J. Heidecke, and K. Singhal. Healthbench: Evaluating large language models towards improved human health. *OpenAI Blog*, 2025.

[2] J. S. Chan, N. Chowdhury, O. Jaffe, J. Aung, D. Sherburn, E. Mays, G. Starace, K. Liu, L. Maksin, T. Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[3] J. Chen, Z. Cai, K. Ji, X. Wang, W. Liu, R. Wang, J. Hou, and B. Wang. Huatuogpt-o1, towards medical complex reasoning with llms. *arXiv preprint arXiv:2412.18925*, 2024.

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] D. Chezelles, T. Le Sellier, M. Gasse, A. Lacoste, A. Drouin, M. Caccia, L. Boisvert, M. Thakkar, T. Marty, R. Assouel, et al. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024.

[6] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[7] J. Coelho, J. Ning, J. He, K. Mao, A. Paladugu, P. Setlur, J. Jin, J. Callan, J. Magalhães, B. Martins, et al. Deepresearchgym: A free, transparent, and reproducible evaluation sandbox for deep research. *arXiv preprint arXiv:2505.19253*, 2025.

[8] G. DeepMind. Alphaevolve: A coding agent for scientific and algorithmic discovery. *Google DeepMind Blog*, 2025.

[9] A. Drouin, M. Gasse, M. Caccia, I. H. Laradji, M. D. Verme, T. Marty, D. Vazquez, N. Chapados, and A. Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? In *Forty-first International Conference on Machine Learning*, 2024.

[10] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[11] S. Gao, R. Zhu, Z. Kong, A. Noori, X. Su, C. Ginder, T. Tsiligkaridis, and M. Zitnik. Txagent: An ai agent for therapeutic reasoning across a universe of tools. *arXiv preprint arXiv:2503.10970*, 2025.

[12] Gemma. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[13] Google. Medgemma hugging face. `https://huggingface.co/collections/google/medgemma-release-680aade845f90bec6a3f60c4`, 2025. Accessed: [2025-05-20].

[14] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[15] S. Guo, C. Deng, Y. Wen, H. Chen, Y. Chang, and J. Wang. DS-agent: Automated data science by empowering large language models with case-based reasoning. In *Forty-first International Conference on Machine Learning*, 2024.

[16] HAI@Stanford. Holistic evaluation of large language models for medical applications. *Blog Post*, 2025.

[17] K. Huang, S. Zhang, H. Wang, Y. Qu, Y. Lu, Y. Roohani, R. Li, L. Qiu, G. Li, J. Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, 2025.

[18] Q. Huang, J. Vora, P. Liang, and J. Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

[19] X. Huang, J. Wu, H. Liu, X. Tang, and Y. Zhou. m1: Unleash the potential of test-time scaling for medical reasoning with large language models. *arXiv preprint arXiv:2504.00869*, 2025.

[20] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[21] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[22] S. Jiang, Y. Liao, Z. Chen, Y. Zhang, Y. Wang, and Y. Wang. Meds^3: Towards medical small language models with self-evolved slow thinking, 2025.

[23] Y. Jiang, K. C. Black, G. Geng, D. Park, A. Y. Ng, and J. H. Chen. Medagentbench: Dataset for benchmarking llms as agents in medical applications. *arXiv preprint arXiv:2501.14654*, 2025.

[24] D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

[25] Q. Jin, B. Dhingra, Z. Liu, W. Cohen, and X. Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2567–2577, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[26] Q. Jin, Z. Wang, Y. Yang, Q. Zhu, D. Wright, T. Huang, W. J. Wilbur, Z. He, A. Taylor, Q. Chen, et al. Agentmd: Empowering language agents for risk prediction with large-scale clinical tool learning. *arXiv preprint arXiv:2402.13225*, 2024.

[27] L. Jing, Z. Huang, X. Wang, W. Yao, W. Yu, K. Ma, H. Zhang, X. Du, and D. Yu. DSBench: How far are data science agents from becoming data science experts? In *The Thirteenth International Conference on Learning Representations*, 2025.

[28] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[29] H. Kang and C. Xiong. Researcharena: Benchmarking llms' ability to collect and organize information as research agents. *arXiv preprint arXiv:2406.10291*, 2024.

[30] N. Khandekar, Q. Jin, G. Xiong, S. Dunn, S. Applebaum, Z. Anwar, M. Sarfo-Gyamfi, C. Safranek, A. Anwar, A. Zhang, et al. Medcalc-bench: Evaluating large language models for medical calculations. *Advances in Neural Information Processing Systems*, 37:84730–84745, 2024.

[31] Y. Kim, C. Park, H. Jeong, Y. S. Chan, X. Xu, D. McDuff, H. Lee, M. Ghassemi, C. Breazeal, and H. W. Park. MDAgents: An Adaptive Collaboration of LLMs for Medical Decision-Making. In *Advances in Neural Information Processing Systems*, 2024.

[32] Y. Kwon, J. Kim, G. Lee, S. Bae, D. Kyung, W. Cha, T. Pollard, A. JOHNSON, and E. Choi. EHRCon: Dataset for checking consistency between unstructured notes and structured tables in electronic health records. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[33] Y. Lai, J. Zhong, M. Li, S. Zhao, and X. Yang. Med-r1: Reinforcement learning for generalizable medical reasoning in vision-language models. *arXiv preprint arXiv:2503.13939*, 2025.

[34] W. Lan, W. Wang, C. Ji, G. Yang, Y. Zhang, X. Liu, S. Wu, and G. Wang. Clinicalgpt-r1: Pushing reasoning capability of generalist disease diagnosis with large language model. *arXiv preprint arXiv:2504.09421*, 2025.

[35] G. Lee, H. Hwang, S. Bae, Y. Kwon, W. Shin, S. Yang, M. Seo, J.-Y. Kim, and E. Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601, 2022.

[36] B. Li, T. Yan, Y. Pan, J. Luo, R. Ji, J. Ding, Z. Xu, S. Liu, H. Dong, Z. Lin, et al. Mmedagent: Learning to use medical tools with multi-modal agent. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8745–8760, 2024.

[37] J. Li, S. Wang, M. Zhang, W. Li, Y. Lai, X. Kang, W. Ma, and Y. Liu. Agent Hospital: A Simulacrum of Hospital with Evolvable Medical Agents. *arXiv preprint arXiv:2405.02957*, 2024.

[38] S. Li, V. Balachandran, S. Feng, J. Ilgen, E. Pierson, P. W. W. Koh, and Y. Tsvetkov. Mediq: Question-asking llms and a benchmark for reliable interactive clinical reasoning. *Advances in Neural Information Processing Systems*, 37:28858–28888, 2024.

[39] S. S. Li, J. Mun, F. Brahman, J. S. Ilgen, Y. Tsvetkov, and M. Sap. Aligning llms to ask good questions a case study in clinical reasoning. *arXiv preprint arXiv:2502.14860*, 2025.

[40] Y. Liao, S. Jiang, Y. Wang, and Y. Wang. Reflectool: Towards reflection-aware tool-augmented clinical agents. *arXiv preprint arXiv:2410.17657*, 2024.

[41] V. Liévin, C. E. Hother, A. G. Motzfeldt, and O. Winther. Can large language models reason about medical questions? *Patterns*, 5(3), 2024.

[42] T. Liu, S. Han, X. Luo, H. Wang, P. Lu, B. Zhu, Y. Wang, K. Li, J. Chen, R. Qu, et al. Towards artificial intelligence research assistant for expert-involved learning. *arXiv preprint arXiv:2505.04638*, 2025.

[43] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[44] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[45] B. Meskó and E. J. Topol. The imperative for regulatory oversight of large language models (or generative ai) in healthcare. *NPJ Digital Medicine*, 6(1):120, 2023.

[46] Ministral. Un ministral, des ministraux. *Ministral Blog*, 2025.

[47] L. Mitchener, J. M. Laurent, B. Tenmann, S. Narayanan, G. P. Wellawatte, A. White, L. Sani, and S. G. Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology. *arXiv preprint arXiv:2503.00096*, 2025.

[48] M. Moor, O. Banerjee, Z. S. H. Abad, H. M. Krumholz, J. Leskovec, E. J. Topol, and P. Rajpurkar. Foundation models for generalist medical artificial intelligence. *Nature*, 616(7956):259–265, 2023.

[49] D. Nathani, L. Madaan, N. Roberts, N. Bashlykov, A. Menon, V. Moens, A. Budhiraja, D. Magka, V. Vorotilov, G. Chaurasia, et al. Mlgym: A new framework and benchmark for advancing ai research agents. *arXiv preprint arXiv:2502.14499*, 2025.

[50] A. Nimmolrat, K. Sutham, and O. Thinnukool. Patient triage system for supporting the operation of dispatch centres and rescue teams. *BMC medical informatics and decision making*, 21:1–16, 2021.

[51] OpenAI. Introducing gpt-4.1 in the api. *OpenAI Blog*, 2025.

[52] OpenAI. Openai o3 and o4-mini system card. *OpenAI Blog*, 2025.

[53] A. Pal, L. K. Umapathi, and M. Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on health, inference, and learning*, pages 248–260. PMLR, 2022.

[54] J. Pan, X. Wang, G. Neubig, N. Jaitly, H. Ji, A. Suhr, and Y. Zhang. Training software engineering agents and verifiers with SWE-gym. In *ICLR 2025 Third Workshop on Deep Learning for Code*, 2025.

[55] T. J. Pollard, A. E. Johnson, J. D. Raffa, L. A. Celi, R. G. Mark, and O. Badawi. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific data*, 5(1):1–13, 2018.

[56] C. Qian, E. C. Acikgoz, H. Wang, X. Chen, A. Sil, D. Hakkani-Tür, G. Tur, and H. Ji. Smart: Self-aware agent for tool overuse mitigation. *arXiv preprint arXiv:2502.11435*, 2025.

[57] J. Qiu, X. Qi, T. Zhang, X. Juan, J. Guo, Y. Lu, Y. Wang, Z. Yao, Q. Ren, X. Jiang, X. Zhou, D. Liu, L. Yang, Y. Wu, K. Huang, S. Liu, H. Wang, and M. Wang. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025.

[58] Qwen. Qwen3: Think deeper, act faster. *Qwen Blog*, 2025.

[59] Qwen. Qwq-32b: Embracing the power of reinforcement learning, March 2025.

[60] P. Ruan, I. Villanueva-Miranda, J. Liu, D. M. Yang, Q. Zhou, G. Xiao, and Y. Xie. N-power ai: A specialized agent framework for automated sample size and power analysis in clinical trial design. *bioRxiv*, pages 2025–02, 2025.

[61] J. Ryu, S. Cho, G. Lee, and E. Choi. Ehr-seqsql: A sequential text-to-sql dataset for interactively exploring electronic health records. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 16388–16407, 2024.

[62] S. Schmidgall and M. Moor. Agentrxiv: Towards collaborative autonomous research. *arXiv preprint arXiv:2503.18102*, 2025.

[63] S. Schmidgall, Y. Su, Z. Wang, X. Sun, J. Wu, X. Yu, J. Liu, Z. Liu, and E. Barsoum. Agent laboratory: Using llm agents as research assistants, 2025.

[64] S. Schmidgall, R. Ziaei, C. Harris, E. Reis, J. Jopling, and M. Moor. Agentclinic: a multimodal agent benchmark to evaluate ai in simulated clinical environments. *arXiv preprint arXiv:2405.07960*, 2024.

[65] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[66] Y. Shao, V. Samuel, Y. Jiang, J. Yang, and D. Yang. Collaborative gym: A framework for enabling and evaluating human-agent collaboration. *arXiv preprint arXiv:2412.15701*, 2024.

[67] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[68] G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.

[69] W. Shi, R. Xu, Y. Zhuang, Y. Yu, H. Sun, H. Wu, C. Yang, and M. D. Wang. Medadapter: Efficient test-time adaptation of large language models towards medical reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, volume 2024, page 22294, 2024.

[70] W. Shi, R. Xu, Y. Zhuang, Y. Yu, J. Zhang, H. Wu, Y. Zhu, J. C. Ho, C. Yang, and M. D. Wang. EHRAgent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22315–22339, 2024.

[71] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

[72] A. Soroush, B. S. Glicksberg, E. Zimlichman, Y. Barash, R. Freeman, A. W. Charney, G. N. Nadkarni, and E. Klang. Large language models are poor medical coders—benchmarking of medical code querying. *NEJM AI*, 1(5):AIdbp2300040, 2024.

[73] Z. Stojanovski, O. Stanley, J. Sharratt, R. Jones, A. Adefioye, J. Kaddour, and A. Kopf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards, 2025.

[74] K. Swanson, W. Wu, N. L. Bulaong, J. E. Pak, and J. Zou. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*, pages 2024–11, 2024.

[75] X. Tang, Y. Liu, Z. Cai, Y. Shao, J. Lu, Y. Zhang, Z. Deng, H. Hu, K. An, R. Huang, et al. Ml-bench: Evaluating large language models and agents for machine learning tasks on repository-level code. *arXiv preprint arXiv:2311.09835*, 2023.

[76] X. Tang, B. Qian, R. Gao, J. Chen, X. Chen, and M. B. Gerstein. Biocoder: a benchmark for bioinformatics code generation with large language models. *Bioinformatics*, 40, 2024.

[77] X. Tang, D. Shao, J. Sohn, J. Chen, J. Zhang, J. Xiang, F. Wu, Y. Zhao, C. Wu, W. Shi, et al. Medagentsbench: Benchmarking thinking models and agent frameworks for complex medical reasoning. *arXiv preprint arXiv:2503.07459*, 2025.

[78] X. Tang, A. Zou, Z. Zhang, Z. Li, Y. Zhao, X. Zhang, A. Cohan, and M. Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 599–621, 2024.

[79] S. Tayebi Arasteh, T. Han, M. Lotfinia, C. Kuhl, J. N. Kather, D. Truhn, and S. Nebelung. Large language models streamline automated machine learning for clinical studies. *Nature Communications*, 15(1):1603, 2024.

[80] N. Team, B. Zhang, S. Feng, X. Yan, J. Yuan, Z. Yu, X. He, S. Huang, S. Hou, Z. Nie, et al. Novelseek: When agent becomes the scientist–building closed-loop system from hypothesis to verification. *arXiv preprint arXiv:2505.16938*, 2025.

[81] R. Thapa, Q. Wu, K. Wu, H. Zhang, A. Zhang, E. Wu, H. Ye, S. Bedi, N. Aresh, J. Boen, et al. Disentangling reasoning and knowledge in medical large language models. *arXiv preprint arXiv:2505.11462*, 2025.

[82] G. Tsatsaronis, G. Balikas, P. Malakasiotis, I. Partalas, M. Zschunke, M. R. Alvers, D. Weissenborn, A. Krithara, S. Petridis, D. Polychronopoulos, et al. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. *BMC bioinformatics*, 16:1–28, 2015.

[83] B. Wang, H. Zhao, H. Zhou, L. Song, M. Xu, W. Cheng, X. Zeng, Y. Zhang, Y. Huo, Z. Wang, et al. Baichuan-m1: Pushing the medical capability of large language models. *arXiv preprint arXiv:2502.12671*, 2025.

[84] P. Wang, T. Shi, and C. K. Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361, 2020.

[85] S. Wang, M. B. A. McDermott, G. Chauhan, M. Ghassemi, M. C. Hughes, and T. Naumann. Mimic-extract: a data extraction, preprocessing, and representation pipeline for mimic-iii. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, CHIL '20, pages 222–235, New York, NY, USA, 2020. Association for Computing Machinery.

[86] W. Wang, Z. Ma, M. Ding, S. Zheng, S. Liu, J. Liu, J. Ji, W. Chen, X. Li, L. Shen, et al. Medical reasoning in the era of llms: A systematic review of enhancement techniques and applications. *arXiv preprint arXiv:2508.00669*, 2025.

[87] W. Wang, Z. Ma, Z. Wang, C. Wu, J. Ji, W. Chen, X. Li, and Y. Yuan. A survey of llm-based agents in medicine: How far are we from baymax? *arXiv preprint arXiv:2502.11211*, 2025.

[88] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.

[89] Z. Wang, B. Danek, and J. Sun. Biodsa-1k: Benchmarking data science agents for biomedical research. *arXiv preprint arXiv:2505.16100*, 2025.

[90] Z. Wang, B. Danek, Z. Yang, Z. Chen, and J. Sun. Can large language models replace data scientists in clinical research? *arXiv preprint arXiv:2410.21591*, 2024.

[91] Z. Wang, K. Wang, Q. Wang, P. Zhang, L. Li, Z. Yang, K. Yu, M. N. Nguyen, L. Liu, E. Gottlieb, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

[92] M. Wornow, R. Thapa, E. Steinberg, J. Fries, and N. Shah. Ehrshot: An ehr benchmark for few-shot evaluation of foundation models. *Advances in Neural Information Processing Systems*, 36:67125–67137, 2023.

[93] M. Wornow, Y. Xu, R. Thapa, B. Patel, E. Steinberg, S. Fleming, M. A. Pfeffer, J. Fries, and N. H. Shah. The shaky foundations of clinical foundation models: A survey of large language models and foundation models for emrs. *ArXiv preprint*, abs/2303.12961, 2023.

[94] J. Wu, W. Deng, X. Li, S. Liu, T. Mi, Y. Peng, Z. Xu, Y. Liu, H. Cho, C.-I. Choi, et al. Medreason: Eliciting factual medical reasoning steps in llms via knowledge graphs. *arXiv preprint arXiv:2504.00993*, 2025.

[95] K. Wu, E. Wu, R. Thapa, K. Wei, A. Zhang, A. Suresh, J. J. Tao, M. W. Sun, A. Lozano, and J. Zou. Medcasereasoning: Evaluating and learning diagnostic reasoning from clinical case reports. *arXiv preprint arXiv:2505.11733*, 2025.

[96] Z. Xi, Y. Ding, W. Chen, B. Hong, H. Guo, J. Wang, D. Yang, C. Liao, X. Guo, W. He, et al. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*, 2024.

[97] P. Xia, J. Wang, Y. Peng, K. Zeng, X. Wu, X. Tang, H. Zhu, Y. Li, S. Liu, Y. Lu, et al. Mmedagent-rl: Optimizing multi-agent collaboration for multimodal medical reasoning. *arXiv preprint arXiv:2506.00555*, 2025.

[98] G. Xiong, Q. Jin, Z. Lu, and A. Zhang. Benchmarking retrieval-augmented generation for medicine. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 6233–6251, 2024.

[99] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[100] J. Yang, C. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[101] J. Yang, K. Leret, C. E. Jimenez, A. Wettig, K. Khandpur, Y. Zhang, B. Hui, O. Press, L. Schmidt, and D. Yang. Swe-smith: Scaling data for software engineering agents, 2025.

[102] J. Yuan, X. Yan, B. Shi, T. Chen, W. Ouyang, B. Zhang, L. Bai, Y. Qiao, and B. Zhou. Dolphin: Closed-loop open-ended auto-research through thinking, practice, and feedback. *arXiv preprint arXiv:2501.03916*, 2025.

[103] D. Zhang, S. Zhoubian, M. Cai, F. Li, L. Yang, W. Wang, T. Dong, Z. Hu, J. Tang, and Y. Yue. Datascibench: An llm agent benchmark for data science. *arXiv preprint arXiv:2502.13897*, 2025.

[104] S. Zhang, Q. Liu, G. Qin, T. Naumann, and H. Poon. Med-rlvr: Emerging medical reasoning from a 3b base model via reinforcement learning. *arXiv preprint arXiv:2502.19655*, 2025.

[105] Y. Zhang, Q. Jiang, X. XingyuHan, N. Chen, Y. Yang, and K. Ren. Benchmarking data science agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5677–5700, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[106] W. Zhao, M. Yuksekgonul, S. Wu, and J. Zou. Sirius: Self-improving multi-agent systems via bootstrapped reasoning. *arXiv preprint arXiv:2502.04780*, 2025.

[107] T. Zheng, G. Zhang, T. Shen, X. Liu, B. Y. Lin, J. Fu, W. Chen, and X. Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

[108] Y. Zhu, Z. He, H. Hu, X. Zheng, X. Zhang, Z. Wang, J. Gao, L. Ma, and L. Yu. Medagentboard: Benchmarking multi-agent collaboration with conventional methods for diverse medical tasks. *arXiv preprint arXiv:2505.12371*, 2025.

[109] Y. Zuo, S. Qu, Y. Li, Z. Chen, X. Zhu, E. Hua, K. Zhang, N. Ding, and B. Zhou. Medxpertqa: Benchmarking expert-level medical reasoning and understanding. *arXiv preprint arXiv:2501.18362*, 2025.

A Limitations and Ethical Statements

A.1 Limitations

Our proposed `MedAgentGym` and associated methods require substantial computational resources for trajectory sampling, model fine-tuning, and iterative self-improvement procedures. Although we achieve significant improvements with relatively lightweight OSS LLMs, further scaling and advanced optimization methods (*e.g.*, PPO) require increased computing infrastructure, limiting accessibility for resource-constrained research groups. Additionally, our current dataset size and trajectory collection are primarily constrained by computational budget rather than data availability, potentially limiting the full exploration of model scaling behavior.

A.2 Broader Impacts

Potential Positive Societal Impacts. `MedAgentGym` can significantly enhance the development of accessible, affordable, and privacy-preserving AI tools for clinical decision-making. Improved coding-based medical reasoning capabilities in open-source LLMs have the potential to democratize access to advanced computational healthcare assistance, benefiting clinicians, researchers, and healthcare systems globally, particularly in resource-limited settings. The plug-and-play architecture also allows continuous adaptation to new medical knowledge and practices, fostering sustainable and community-driven innovation in healthcare technology.

Potential Negative Societal Impacts. Despite the benefits, the introduction and widespread deployment of sophisticated computational frameworks like `MedAgentGym` may unintentionally widen existing healthcare inequities. Institutions with limited computational resources (including both Microsoft Azure API service and high-performance computing clusters) or inadequate data infrastructure may struggle to access or fully benefit from these technological advancements, potentially exacerbating disparities in healthcare capabilities across regions or socioeconomic groups. Moreover, reliance on publicly available datasets may perpetuate existing biases due to uneven data representation, potentially disadvantaging underrepresented patient populations and rare disease conditions.

A.3 Ethical Statements

Table 7: Data Access and License Information of 12 datasets in `MedAgentGym`. "Custom" represents additional dataset- or task-specific license and data access requirements (*e.g.*, DUA or credentials).

Dataset	Data License	Data Access	Code License	Code Access
Training and Internal Validation (In-Distribution)				
MIMIC-III [28, 35]	Custom	MIMIC-III on PhysioNet	CC-BY-4.0	MIMIC-III on EHRSQL
eICU [55, 35]	Custom	eICU on PhysioNet	CC-BY-4.0	eICU on EHRSQL
TREQS [84]	Custom	MIMIC-III on PhysioNet	MIT	TREQS on GitHub
MedCalcBench [30]	CC-BY-SA 4.0	MedCalcBench	Public	MedCalcBench on GitHub
MedAgentBench [23]	MIT	MedAgentBench (FHIR Server)	MIT	MedAgentBench on GitHub
BioCoder [76]	CC-BY-4.0	BioCoder on Huggingface	N/A	BioCoder on GitHub
BioDSBench [90]	MIT	BioDSBench	MIT	BioDSBench on GitHub
EHRSHOT [92]	Custom	EHRShot (Standford)	Apache	EHRSHOT on Github
External Validation (Out-of-Distribution)				
EHR-SeqSQL [61]	Custom	MIMIC-III on PhysioNet	N/A	EHR-SeqSQL on GitHub
EHR-Con [32]	Custom	MIMIC-III on PhysioNet	MIT	EHR-Con on GitHub
MIMIC-Extract [85]	Custom	MIMIC-III on PhysioNet	MIT	MIMIC-Extract on GitHub
N-PowerAI [60]	N/A	N-Power AI Supp. Mat.	N/A	N-Power AI on Webpage

Data Privacy and Licensing. We carefully curated `MedAgentGym` with strict adherence to ethical standards, using publicly available datasets or datasets with appropriate privacy protections and anonymizations. Table 7 lists the access requirements for the 12 datasets in `MedAgentGym` and the code base for data processing or task implementation. We explicitly designed isolated Docker environments to ensure data privacy and security. Nevertheless, ethical usage of our methods and models in clinical settings requires rigorous validation, transparency in limitations, and close collaboration with healthcare professionals. We encourage responsible deployment, emphasizing human oversight, continuous evaluation, and clear communication of model capabilities and uncertainties to mitigate ethical and practical risks.

LLM Usage Statement. In compliance with the PhysioNet Credentialed Health Data Use Agreement (version 1.5.0)[4], we strictly prohibit transferring confidential patient data (*e.g.*, MIMIC-III and eICU) to third-party entities, including external online services and APIs. To responsibly utilize the Azure OpenAI Service, we adhere closely to PhysioNet's guidelines on responsible GPT usage[5]. Specifically, we have opted out of the human review process by completing the Azure OpenAI Additional Use Case Form[6], thereby ensuring no third-party entity (such as Microsoft) accesses or processes sensitive patient information. We consistently monitor our data handling practices and strictly adhere to applicable guidelines and privacy regulations, maintaining the highest ethical standards in our research and operations.

B Related Works

Medical Agents (Coding). Recent advances have demonstrated that LLMs exhibit strong capabilities in medical reasoning and planning leveraging extensive biomedical knowledge [71, 48, 41], fueling increased interest in developing LLM-based autonomous agents tailored specifically for medical tasks [26, 11, 36, 40, 78, 31]. In particular, LLM-based agents have shown promise in specialized computational tasks, including querying EHR databases [70], performing biostatistical calculations [60], and conducting bioinformatics analyses [76, 90, 79]. As shown in Figure 4, integrating coding capabilities into LLM-based agents further enhances performance on tasks traditionally approached through natural language reasoning (*e.g.*, MIMIC-III, eICU [35]), as well as numerical and rule-based medical reasoning (*e.g.*, MedCalcBench [30]). However, existing coding-based medical agents rely primarily on prompt engineering without systematic improvement, limiting their robustness and scalability when addressing complex and diverse coding tasks in real-world biomedical scenarios. In contrast, `MedAgentGym` specifically targets reasoning-intensive coding tasks by introducing a unified, scalable, and interactive training environment that systematically improves the coding-based medical reasoning capabilities of LLM agents.

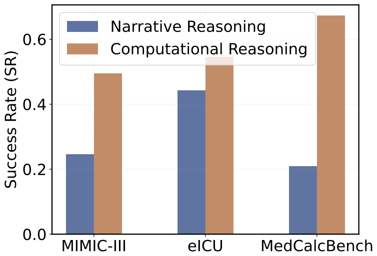


Figure 4: Coding empowers computational medical reasoning (w/ `gpt-4-turbo`).

Medical Reasoning Models. Recent advancements have substantially improved medical reasoning capabilities of LLMs through RL [19, 33, 104, 22, 94, 3, 34, 83, 39, 104]. For example, M1 [19] improves by distilling knowledge from the reasoning traces generated by DeepSeek-R1 [14]. MedS3 [22] employs Monte Carlo Tree Search (MCTS) to generate rule-verifiable reasoning trajectories and employs process-reward models to select optimal reasoning paths during inference. Similarly, HuatuoGPT-o1 [3] and ClinicalGPT-R1 [34] integrate domain-specific verifiers to guide RL fine-tuning processes for improved clinical reasoning. Extending beyond language modeling, Med-R1 [33] and MedXpertQA [109] adapt RL methodologies to vision-language models, effectively addressing medical visual question answering tasks. Despite these developments, current medical reasoning models predominantly target natural language-based reasoning, with limited attention given to coding-intensive scenarios common in biomedical research and clinical practice.

Medical Reasoning Benchmarks. Most existing medical reasoning benchmarks focus primarily on evaluating LLM performance through closed-form medical QA tasks [53, 24, 25, 82, 77, 98, 1]. In addition, AgentClinic [64] further evaluates diagnosis prediction within simulated clinical scenarios, while MedHELM [16] provides comprehensive evaluations across various medical NLP tasks. Despite these extensive benchmarking efforts, existing benchmarks – including recent concurrent works such as MedAgentBoard [108], HealthBench [1], and MedCaseReasoning [95] – typically focus on evaluation scenarios, with limited emphasis on dedicated training environments aimed at systematically improving medical reasoning capabilities [81], especially within coding-intensive and interactive medical scenarios.

---

[4] https://physionet.org/about/licenses/physionet-credentialed-health-data-license-150/

[5] https://physionet.org/news/post/gpt-responsible-use

[6] https://aka.ms/oai/additionalusecase

Medical Agent Training Environments. To advance medical agents with narrative reasoning, AgentClinic [64] and AgentHospital [37] simulate hospital workflows focused on diagnostic tasks, while MediQ [38] offers interactive simulations designed for medical information retrieval. Beyond medicine, specialized environments have emerged for systematically evaluating and improving LLM agents across diverse tasks [106, 91], such as software engineering [54, 100, 101], reasoning [73], web browsing [9], agent planning and collaboration [96, 66], data science [15, 27, 103, 105], machine learning engineering [49, 18, 2, 75], automated research [7, 29, 62, 63], and scientific discovery [80, 102]. Inspired by these interactive training frameworks, `MedAgentGym` uniquely targets real-world biomedical scenarios, aiming to rigorously benchmark and systematically enhance coding-based medical reasoning capabilities of LLM agents.

C Task and Data Details

C.1 Overview

`MedAgentGym` integrates a comprehensive suite of datasets that encompass both structured and open-ended biomedical reasoning tasks.

Computational Task Category. *Structured tasks* primarily include database query scenarios, such as those from MIMIC-III, eICU, TREQS, EHR-SeqSQL, EHRCon, and MedCalcBench (rule- or equation-based), which require precise formulation of executable queries against structured EHR data. *Open-ended tasks* include biomedical data analysis and medical coding scenarios drawn from datasets such as MedAgentBench, BioCoder, BioDSBench, EHRSHOT, MIMIC-Extract, and N-PowerAI, demanding nuanced and flexible code generation for complex analysis, statistical reasoning, or clinical decision-making.

Biomedical Application Category. `MedAgentGym` spans multiple biomedical subdomains, including *Database queries* (DB, including MIMIC-III, eICU, TREQS, EHR-SeqSQL, and EHRCon), *Data Science* (DS, including MedCalcBench and MedAgentBench), *Bioinformatics* (Bioinfo, including BioCoder, BioDSBench, N-PowerAI), and *Machine Learning* (ML, including EHRSHOT and MIMIC-Extract). In several empirical analysis, we include ML as part of the Bioinfo. category for better visualization.

In- & Out-of-Distribution. We further categorize `MedAgentGym` into *in-distribution* tasks (EHRSQL, TREQS, MedCalcBench, BioCoder, BioDSBench, MedAgentBench, EHRSHOT) and *out-of-distribution* tasks (EHR-SeqSQL, EHRCon, MIMIC-Extract, N-PowerAI), facilitating a rigorous evaluation of model generalization and adaptability. To highlight intrinsic differences between these distributions, Figure 5 shows the distribution of sampled code trajectories. The resulting visualization demonstrates significant divergence in trajectory complexity, interaction frequency, and required code refinement steps between in-distribution and out-of-distribution tasks, underscoring the challenges posed by novel biomedical reasoning contexts.

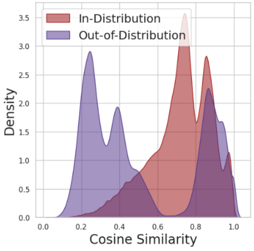


Figure 5: Inter-distribution similarity.

C.2 Training and Internal Testing (In-Distribution) Dataset Details

EHRSQL: MIMIC-III and eICU. EHRSQL [35] comprises text-to-SQL tasks that leverage electronic health records from MIMIC-III [28] and eICU [55]. They evaluate the ability of LLMs (and agents) to translate clinical questions posed by healthcare professionals into executable SQL queries. This includes handling complex queries involving temporal logic and conditional abstention.

TREQS. TREQS [84] is a text-to-SQL benchmark tailored specifically to clinical question answering using the MIMIC-III dataset. It emphasizes generating accurate SQL queries from template-based natural language questions against a simplified schema comprising five core tables, with an emphasis on large result-set handling.

MedCalcBench. MedCalcBench [30] provides a structured evaluation of clinical calculation capabilities in LLMs. Each instance poses a patient-specific clinical scenario requiring precise medical calculations such as clinical scores or medication dosages, accompanied by expert-curated stepwise solutions for validation.

MedAgentBench. MedAgentBench [23] is a simulated EHR environment designed to evaluate LLM-driven clinical workflows. It features realistic patient scenarios across ten task categories, requiring agents to perform clinical reasoning, EHR querying via FHIR interfaces, and clinical decision support.

BioCoder. BioCoder [76] assesses the capability of LLMs to generate accurate bioinformatics code solutions. It comprises practical coding challenges derived from authentic bioinformatics software, requiring the generation and verification of functionally correct Python methods.

BioDSBench. BioDSBench [90] evaluates LLM proficiency in biomedical data science coding tasks, involving the generation of Python or R code to replicate analytical workflows derived from actual biomedical research studies. Tasks span statistical analyses, data manipulations, and visualization routines.

EHRSHOT. EHRSHOT [92] benchmarks LLMs on few-shot clinical prediction tasks leveraging real-world, longitudinal, deidentified EHR data. It focuses on rapid adaptation to tasks such as risk prediction and forecasting clinical outcomes given limited labeled examples.

C.3 External Evaluation (Out-of-Distribution) Dataset Details

EHR-SeqSQL. EHR-SeqSQL [61] extends text-to-SQL evaluation to sequential, multi-turn interactions, emulating realistic clinical dialogues. Tasks require maintaining context across multiple SQL queries, assessing LLM capability in handling compositional and contextual reasoning.

EHRCon. EHRCon [32] involves assessing clinical note consistency with structured EHR records, focusing on identifying discrepancies. It serves as a verification task requiring precise alignment between unstructured clinical text and corresponding database entries.

MIMIC-Extract. MIMIC-Extract[85] provides structured, preprocessed time-series patient data derived from the MIMIC-III dataset, used in clinical predictive modeling such as mortality risk or intervention prediction, enabling standardized assessments of time-series reasoning capabilities.

N-PowerAI. N-PowerAI [60] evaluates LLM capabilities in performing statistical sample-size and power analyses for clinical trial design. It requires multi-step statistical reasoning and the generation of precise numeric results corresponding to various clinical scenarios.

C.4 Train-Test Set Split

For datasets that provide predefined training, validation, and test splits, we combine the training and validation subsets into a single unified training set and retain the original test subset exclusively for evaluation. In cases where datasets lack predefined splits, we randomly allocate 50% of the instances to training, assigning the remaining 50% to the test set. For tasks containing more than 1000 samples in both training and test sets, we create a lighter subset through downsampling to support efficient leaderboard-based training and evaluation. Specifically, we leverage task-specific metadata to perform uniform sampling within each fine-grained category, thereby maintaining diversity, ensuring balanced representation, and preserving the original data distribution.

C.5 Data Pre-processing Details

C.5.1 Structured Tasks

For database querying related datasets, including **MIMIC-III**, **eICU**, **TREQS**, and **EHR-SeqSQL**, each task instance is structured into a JSON format comprising: (1) the contextual description and the corresponding natural-language query, (2) the ground-truth SQL query, and (3) the resulting answer from the database execution. Instances yielding null results upon SQL execution, indicating the absence of a valid answer, are excluded from the dataset.

For **EHRCon**, we organize the data into structured databases that link patient records through hospital admission IDs, complemented by a separate database containing associated clinical notes. Each task is formulated as a JSON object consisting of: (1) admission ID, (2) relevant medical terminology, (3) count of detected inconsistencies, and (4) a binary indicator denoting the presence or absence of inconsistencies.

For **MedCalcBench**, each instance initially consists of a patient note, a specific medical calculation query, a ground-truth answer, and a detailed step-by-step solution. To accurately evaluate the coding capabilities of LLM agents without direct guidance, we remove all intermediate calculation hints, presenting only the patient note and the calculation query for model inference.

For **N-PowerAI**, statistical analysis tasks are augmented through attribute substitution. Specifically, each original instance is expanded 100-fold by systematically replacing an attribute with a randomly chosen equivalent from a predefined valid range, preserving the integrity and interpretability of the statistical context. Each augmented instance includes recalculated values for sample size (N) and statistical power, stored systematically within JSON-formatted records.

C.5.2 Open-ended Tasks

MedAgentBench instances require LLM agents to follow natural-language instructions to perform tasks within a FHIR-compliant interactive medical environment. We retain original instructions, solutions, and Medical Record Numbers (MRNs). To derive verifiable evaluation signals, we execute the provided ground-truth solutions on the server-side environment to obtain authoritative reference answers.

BioCoder tasks require implementing biostatistics algorithms or addressing scientific programming challenges. Each instance comprises a problem description, context-specific code, test cases, and expected outputs. While evaluation datasets already contain all necessary components, training instances initially lack context-specific code and test cases. To address this gap, we employ the `o3-mini` model to auto-generate relevant context code and corresponding test cases based on provided ground-truth functions. Generated functions undergo rigorous validation via a code interpreter, retaining only verified, error-free instances. Additionally, we exclusively utilize the Python-based subset of BioCoder, deferring the JavaScript subset for subsequent integration.

BioDSBench instances involve biomedical data analysis tasks derived from real-world datasets. Features are systematically organized into directories by task, with each task's description and reference Python implementation captured within JSON structures.

For datasets dedicated to predictive model development (*e.g.*, **EHRSHOT** and **MIMIC-Extract**), initial features are provided in pre-processed form but necessitate additional table joining, filtering, and integration to produce final training inputs. While labels accompany these tasks, explicit reference Python implementations are not provided, as evaluation metrics directly measure the accuracy of model predictions on predefined test subsets. Distinct subsets of training, validation, and testing data and labels are explicitly maintained and separately utilized for both training and evaluation phases.

D Baseline Details

We include additional details of the coding and medical domain-specific LLMs used in our experiments:

- **Qwen2.5-Coder-Instruct** [20] is derived from the Qwen2.5 series and further fine-tuned explicitly on large-scale coding datasets and coding-specific instruction sets. This targeted training substantially enhances their capabilities in code generation, debugging, and programmatic reasoning, outperforming general-purpose models of similar scale on coding tasks.
- **medgemma-4b-it** (`gemma-3-4b-pt`) [13] is a medical-domain variant based on gemma architecture and fine-tuned specifically on medical QA and instruction datasets, which provide strong capabilities for medical reasoning and question answering.
- **HuatuoGPT-o1-7B** (`Qwen2.5-7B-Instruct`) [3], built on the Qwen2.5-7B architecture, is extensively fine-tuned in clinical reasoning datasets via PPO with verifier-based rewards to enhance complex reasoning capabilities. Specifically, it incorporates a medical-specific verifier model that guides the generation of complex reasoning trajectories. HuatuoGPT-o1-7B excels in medical reasoning tasks by explicitly generating intermediate reasoning steps that facilitate iterative refinement and introspective evaluation.
- **m1-7B-23K** (`Qwen2.5-7B-Instruct`) [19] is fine-tuned on approximately 23,000 rigorously curated medical QA examples, significantly enhancing its domain-specific knowledge and reasoning capabilities.

- **MedReason-8B** (`Llama-3.1-8B-Instruct`) [94] is specifically fine-tuned for medical questions-answering and clinical reasoning tasks. Its training emphasizes the generation of step-by-step rationales, enabling robust performance on medical reasoning and diagnostic tasks.
- **Baichuan-M1-14B-Instruct** [83] is a 14B-parameter medical LLM pre-trained from scratch on approximately 20 trillion tokens of medical domain-specific content and high-quality general text. It integrates specialized modeling across over 20 medical specialties with advanced architectural modifications enhancing context understanding and long-sequence reasoning.

E Implementation Details

Evaluation Metrics. Following existing agent benchmarks [43], we adopt *success rate (SR)* as the primary evaluation metric. For *database, data science, and bioinformatics* tasks with explicit ground truths, we compare LLM-generated code execution outputs with reference solutions using exact match. For open-ended *ML* tasks in clinical decision support, we measure performance using *accuracy (Acc)* across provided test cases. Note that these code generation tasks inherently have infinite solution spaces, unlike traditional classification problems with bounded solution spaces (*e.g.*, even random guessing can yield around 50% accuracy in binary classification). The *overall score* is computed by averaging performance across tasks in test sets of `MedAgentGym` (leaderboard), providing a comprehensive evaluation of coding-based medical reasoning capabilities within `MedAgentGym`.

Experimental Setup Details. We limit interactions to a maximum of 15 turns per session, providing agents full access to interaction histories and constraining runtime to 120 seconds per session. Input tokens are capped at $32,768$, with output limited to $8,192$ tokens per round. We use Python 3.10 as the primary language for agent-code execution due to its modular design and suitability for biomedical computations. To enable interactive feedback (section 3.3), we employ a rule-based parser converting LLM outputs to JSON, facilitating seamless code execution, and utilize `gpt-4.1-mini` to translate execution errors into grounded explanations. We configure all baseline LLMs following established best practices for reproducibility. Specifically, instruction-following LLMs are configured with a temperature of zero, while reasoning models use a temperature of 0.6. For all experiments with `Qwen-3` series, we switch to thinking mode for optimal performance under complex reasoning scenarios (*e.g.*, logic, math, and coding).

SFT. For SFT experiments, smaller models (up to 8B parameters) are trained using eight NVIDIA A100 GPUs, whereas the 14B-parameter model is trained on eight NVIDIA H200 GPUs. We utilize the AdamW optimizer [44] with a learning rate of $1e-4$. The training batch size is set to 8, and the maximum input token length per batch is configured to 40,000 tokens.

DPO. DPO experiments are conducted using the same hardware configurations as SFT experiments. We employ the AdamW optimizer with a reduced learning rate of $5e-6$. Training utilizes a batch size of 64 and a KL-divergence coefficient (β) of 0.1 to regulate the divergence from the initial policy.

PPO & GRPO. PPO and GRPO experiments are conducted using the same hardware configurations as SFT experiments. All online RL experiments are conducted using VeRL framework [68]. We integrate the VeRL package and dependencies inside the `Med-Copilot` docker image to enable communication between the reward functions and the evaluation module. PPO and GRPO training is performed with a batch size of 128 and a learning rate of 1×10^{-5}. The temperature parameter during model rollout is consistently set to 0.6. Throughout training, the coefficient for the KL divergence regularization term is fixed at $\beta = 1 \times 10^{-3}$.

F Additional Experimental Results

F.1 Effect of Pre-defined Toolset

Figure 6 compares the performance of GPT-4-based agents on the MIMIC-III dataset with and without predefined toolsets integrated into our agent scaffold. This illustrates our agent scaffold's ability to flexibly accommodate external tools. Interestingly, despite providing a set of predefined tools, including functions for database loading, data filtering, value retrieval, arithmetic calculations, date computations, and SQL execution (see additional details of toolset in [70]), we observe a surprising

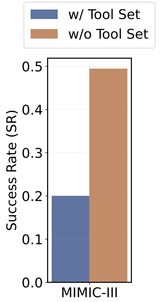


Figure 6: Effect of toolset.

decline in agent performance. It suggests that the LLM agent inherently generates more flexible and contextually appropriate code when unencumbered by predefined function constraints, aligning with the observations reported by [56, 57].

F.2 Code Quality and Efficiency

Table 8: Additional evaluation on code quality and efficiency.

Datasets (→)	**MIMIC.**	**eICU**	**TREQS**	**MedCalc.**	**MedAgent.**	**BioCoder**	**BioDS.**	**EHRSHOT**	**Avg.**
gpt-4.1 (2025-04-14)									
#turns	25.91	26.59	20.65	10.73	17.28	22.08	21.75	8.71	19.21
complexity	0.01	0.06	0.01	4.09	0.23	7.77	0.17	20.97	4.16
maintainability	95.14	95.99	96.62	88.38	91.04	68.20	92.67	56.24	85.54
loc	9.26	9.67	4.17	19.00	18.89	24.82	28.97	144.69	32.43
lloc	5.86	6.33	3.00	15.20	10.79	21.84	16.44	110.51	23.75
gpt-4.1-mini (2025-04-14)									
#turns	19.66	19.90	16.35	9.18	19.20	23.08	16.53	22.60	18.31
complexity	0.02	0.04	0.01	3.51	0.03	7.30	0.26	19.85	3.88
maintainability	95.62	96.06	98.93	87.01	94.43	69.43	92.54	57.77	86.47
loc	16.49	14.47	6.85	23.37	13.08	25.98	28.17	171.69	37.51
lloc	8.05	7.22	3.68	17.58	7.92	20.78	15.40	119.58	25.03
Qwen2.5-7B-Instruct									
#turns	17.23	14.81	12.38	5.98	14.39	25.42	9.31	15.33	14.36
complexity	0.02	0.02	0.01	4.41	0.01	4.78	0.30	11.09	2.58
maintainability	96.54	96.02	98.58	82.65	80.20	81.67	95.66	54.69	85.75
loc	16.81	17.07	8.72	28.54	49.09	20.81	22.00	137.85	37.61
lloc	7.52	8.23	4.38	18.09	25.34	15.46	11.79	90.58	22.67
`Med-Copilot` *(7B)*									
#turns	20.74	17.80	14.31	7.86	16.24	28.97	16.80	29.73	19.06
complexity	0.01	0.01	0.01	3.81	0.01	5.08	0.04	18.66	3.45
maintainability	94.58	95.01	98.49	83.76	82.64	81.40	97.68	62.47	87.00
loc	21.58	19.88	12.00	25.42	53.67	24.76	17.16	141.50	39.50
lloc	9.95	9.10	5.73	17.74	26.26	17.82	9.11	95.97	23.96

For a comprehensive evaluation, we further report additional evaluation metrics on code quality and efficiency, including (1) **number of turns** for interaction effectiveness, (2) cyclomatic **complexity** for code complexity, (3) **maintainability** index for code readability, and (4) **line-of-code (loc)** and (5) **logical line-of-code (lloc)** for code efficiency (Table 8). Comparing different tasks (take gpt-4.1 for example), we observe that machine learning tasks such as EHRSHOT involve significantly higher complexity and longer code. Comparing different models (averaged across datasets), we observe that advanced closed-source models generate more complex and longer code; after training, `Med-Copilot` produces structurally efficient and more maintainable code compared to backbone models.

F.3 Cost Analysis

Table 9: Statistics of input and output tokens per question for API-based commercial LLMs.

Datasets (→)	**MIMIC.**	**eICU**	**TREQS**	**MedCalc.**	**MedAgent.**	**BioCoder**	**BioDS.**	**EHRSHOT**	**Avg.**
Input									
`gpt-4o-mini` [21]	3430.83	1947.72	1689.71	651.92	9501.86	5166.50	5068.88	5986.20	4180.45
`gpt-4o` [21]	4399.87	3122.02	1823.31	739.48	8474.81	5133.71	21077.12	3235.71	6000.75
`gpt-4.1-mini` [51]	1869.37	1691.45	1430.15	834.73	8087.50	2621.79	7369.35	4466.07	3546.30
`gpt-4.1` [51]	3730.90	2979.57	1754.18	759.64	7912.81	2728.24	3035.45	2092.14	3124.12
`gpt-o4-mini` [52]	2005.11	1688.73	1534.84	1306.49	7586.32	2193.82	50768.08	2858.79	8742.77
Output									
`gpt-4o-mini` [21]	1206.00	714.72	918.45	379.28	4206.73	4170.56	1479.87	10484.53	2945.02
`gpt-4o` [21]	840.16	852.41	696.61	537.09	2821.00	4144.91	7278.49	9127.14	3287.23
`gpt-4.1-mini` [51]	952.68	991.78	880.43	1000.06	2892.98	3328.07	1308.73	23276.67	4328.93
`gpt-4.1` [51]	771.91	781.86	753.88	787.45	2051.20	2846.58	1627.78	5163.57	1848.03
`gpt-o4-mini` [52]	1586.65	1392.11	893.76	2407.87	1718.22	3144.74	1952.88	8083.71	2647.49

Table 9 summarizes input and output token statistics for various API-based proprietary LLMs evaluated on datasets within `MedAgentGym`. Notably, the input and output token lengths per query vary significantly across models and tasks. Among these models, `gpt-4.1-mini` achieves relatively low average input and moderate output token counts, which implies more efficient token utilization during inference compared to larger variants such as `gpt-4o` and `gpt-o4-mini`. Conversely, `gpt-o4-mini` incurs higher average input costs. Figure 7 presents the API cost per 100 tasks. Overall, smaller GPT variants (*e.g.*, `gpt-4.1-mini` and `gpt-4o-mini`) offer superior token-efficiency, translating into lower computational and API costs without substantial compromise in performance, demonstrating their effectiveness as cost-efficient solutions for large-scale biomedical reasoning applications.

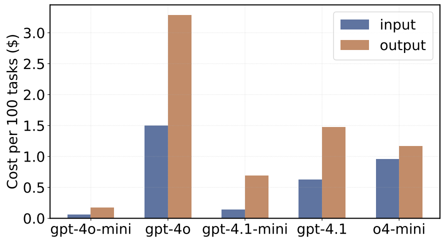
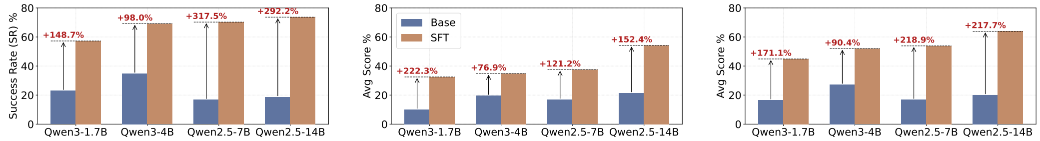


Figure 7: Cost information.

F.4 Structured and Open-ended Tasks



(a) Structured Tasks (b) Open-Ended Tasks (c) Overall

Figure 8: `Med-Copilot` SFT performance on `MedAgentGym` across various backbone LLMs.

Figure 8 shows substantial performance gains from SFT across four OSS backbone LLMs of varying sizes. Simple SFT on successful trajectories markedly boosts performance on structured coding tasks, indicating its effectiveness in capturing structured coding patterns. DPO, in contrast, is particularly effective for optimizing performance on open-ended tasks.

F.5 Human Study

Table 10: Human Performance evaluation on structured and open-ended tasks from `MedAgentGym`.

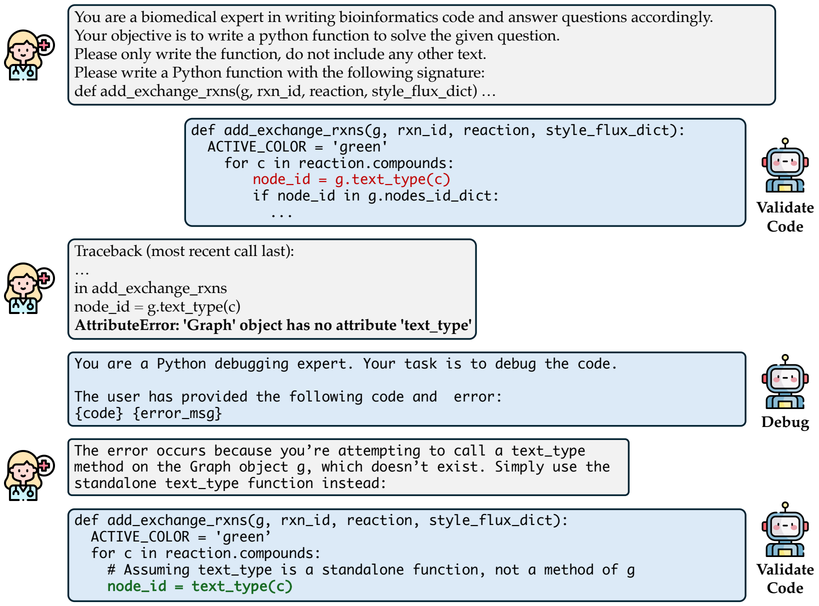
Dataset (↓)	**# Attempt**	**# Correct**	**SR**	**Total Time (min)**	**Avg Time (min)**
Structured					
MIMIC-III [28, 35]	10	8	80%	74	7.40
eICU [55, 35]	8	5	63%	63	7.88
TREQS [84]	10	7	70%	39	3.90
EHR-SeqSQL [61]	10	8	80%	67	6.70
MedCalcBench [30]	7	5	71%	57	8.14
N-PowerAI [60]	7	6	86%	96	13.7
Structured Task (Total)	**52**	**39**	**75%**	**396**	**7.62**
Open-ended					
MedAgentBench [23]	6	6	100%	89	14.833
EHRCon [32]	6	1	17%	241	40.17
BioDSBench [90]	3	0	0%	195	65.00
BioCoder [76]	8	2	25%	142	17.75
EHRSHOT [92]	5	–	89%	185	37.00
MIMIC-Extract [85]	3	–	94%	215	71.67
Open-ended Task (Total)	**31**	**–**	**45%**	**1067**	**34.419**

To systematically compare coding styles and performance differences between human programmers and automated agents, we conducted a human evaluation involving 83 tasks randomly selected from the test subsets of the 12 datasets included in `MedAgentGym`. This evaluation set comprises 52 structured and 31 open-ended biomedical coding tasks. The human participants are biomedical engineers and research scientists with over six years of experience in computational biology, relational database querying, HTTP-based interactions, and machine learning development. The human evaluation study was conducted under the approval of the Institutional Review Board (IRB). Participants voluntarily contributed to the evaluation and did not receive monetary compensation.

Table 10 summarizes the results of human evaluation study conducted to establish reference performance benchmarks across representative structured and open-ended medical reasoning tasks from the `MedAgentGym` benchmark. Human experts completed selected instances from each dataset, documenting the number of attempts, correctly solved instances, overall SR, total time spent, and average time per task (in minutes). Results indicate that, on average, the human subject required approximately 4.5 times longer to solve open-ended tasks relative to structured tasks, while achieving a 40% lower success rate, reflecting the increased complexity and cognitive load associated with open-ended biomedical reasoning scenarios.

In addition, we also performed a quantitative analysis on 250+ trajectories (randomly sampled over 10% of our trajectory collection in section 5.1) and confirmed that the vast majority of successful solutions followed a logically sound path, with cases of 'correct answer from flawed code' being exceptionally rare (<1%).

F.6 Case Study



You are a biomedical expert in writing bioinformatics code and answer questions accordingly.
Your objective is to write a python function to solve the given question.
Please only write the function, do not include any other text.
Please write a Python function with the following signature:
def add_exchange_rxns(g, rxn_id, reaction, style_flux_dict) …

```
def add_exchange_rxns(g, rxn_id, reaction, style_flux_dict):
  ACTIVE_COLOR = 'green'
    for c in reaction.compounds:
        node_id = g.text_type(c)
        if node_id in g.nodes_id_dict:
          ...
```

Traceback (most recent call last):
…
in add_exchange_rxns
node_id = g.text_type(c)
AttributeError: 'Graph' object has no attribute 'text_type'

```
You are a Python debugging expert. Your task is to debug the code.

The user has provided the following code and  error:
{code} {error_msg}
```

```
The error occurs because you're attempting to call a text_type
method on the Graph object g, which doesn't exist. Simply use the
standalone text_type function instead:
```

```
def add_exchange_rxns(g, rxn_id, reaction, style_flux_dict):
  ACTIVE_COLOR = 'green'
  for c in reaction.compounds:
    # Assuming text_type is a standalone function, not a method of g
    node_id = text_type(c)
```

Figure 9: Case study of `gpt-4.1-mini` on BioCoder.

To illustrate the practical utility of interactive coding mechanism, we conduct a detailed case study involving a typical bioinformatics coding task in Figure 9. Specifically, the task requires writing a Python function (`add_exchange_rxns`) that modifies biochemical reaction graphs by integrating exchange reactions. Initially, the LLM agent-generated solution encountered an attribute error, mistakenly invoking a non-existent `text_type` method on a `Graph` object. Upon receiving explicit debugging feedback, the LLM agent effectively identified and corrected the mistake by utilizing the standalone `text_type` function rather than incorrectly calling it as a method of the graph instance. This case highlights the capability of debugging in `MedAgentGym` environment to provide targeted, actionable debugging feedback, enabling iterative code refinement and significantly enhancing agent-generated solutions for complex biomedical programming tasks.

F.7 Difficulty Analysis on External Evaluation Set

`MedAgentGym` includes four challenging unseen out-of-distribution medical coding tasks as external validation sets in section 5.4. For example, the original MIMIC-Extract task in `MedAgentGym` intentionally utilizes *raw, unprocessed* data as a *challenging, out-of-distribution* scenario designed specifically to assess model capabilities in feature engineering and data preprocessing. To illustrate

Table 11: Difficulty analysis on external sets for model generalization.

Model	**MIMIC-Extract (raw)**	**MIMIC-Extract (processed)**	**BixBench**
gpt-4.1-mini	5.62	23.47	26.01
gpt-4.1	10.41	28.94	32.09
Qwen-2.5-7B-Instruct	1.34	17.06	18.92
`Med-Copilot` (7B)	2.14	25.88	28.72
Qwen-2.5-14B-Instruct	4.51	18.52	20.61
`Med-Copilot` (14B)	2.75	28.66	29.39

the difference clearly, we additionally evaluate *a fully pre-processed* version of MIMIC-Extract. As demonstrated in Table 11, providing structured data significantly improves model performance, highlighting the distinct difficulty posed by raw data.

To further demonstrate the generalization of `Med-Copilot`, we include an additional evaluation set, BixBench [47], a bioinformatics coding dataset comprising over 50 real-world scenarios of practical biological data analysis with nearly 300 associated open-answer questions. It is designed to measure the ability of LLM-based agents to explore biological datasets, perform long, multi-step analytical trajectories, and interpret the nuanced results of those analyses. Exceptional performance in BixBench demonstrates the robustness of `Med-Copilot` and its ability to generalize beyond the specific domain of medical coding to broader scientific analytical tasks.

G Prompt Details

G.1 MIMIC-III Prompts

We include prompt details for MIMIC-III tasks as follows:

MIMIC-III Prompt

```
You are a biomedical expert in handling EHR data and answer questions.
Your objective is to solve a coding problem with given EHR data, with
    the goal of finally give a concrete answer to the question.
Assume you have knowledge of several tables:
(1) Tables are linked by identifiers which usually have the suffix 'ID
    '. For example, SUBJECT_ID refers to a unique patient, HADM_ID
    refers to a unique admission to the hospital, and ICUSTAY_ID refers
     to a unique admission to an intensive care unit.
(2) Charted events such as notes, laboratory tests, and fluid balance
    are stored in a series of 'events' tables. For example the
    outputevents table contains all measurements related to output for
    a given patient, while the labevents table contains laboratory test
(3) Tables prefixed with 'd_' are dictionary tables and provide
    definitions for identifiers. For example, every row of chartevents
    is associated with a single ITEMID which represents the concept
    measured, but it does not contain the actual name of the
    measurement. By joining chartevents and d_items on ITEMID, it is
    possible to identify the concept represented by a given ITEMID.
(4) For the databases, four of them are used to define and track
    patient stays: admissions, patients, icustays, and transfers.
    Another four tables are dictionaries for cross-referencing codes
    against their respective definitions: d_icd_diagnoses,
    d_icd_procedures, d_items, and d_labitems.

For different tables, they contain the following information:
(1) ADMISSIONS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ADMITTIME, DISCHTIME,
    ADMISSION_TYPE, ADMISSION_LOCATION, DISCHARGE_LOCATION, INSURANCE,
    LANGUAGE, MARITAL_STATUS, ETHNICITY, AGE
(2) CHARTEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID, ITEMID,
    CHARTTIME, VALUENUM, VALUEUOM
(3) COST.csv: ROW_ID, SUBJECT_ID, HADM_ID, EVENT_TYPE, EVENT_ID,
    CHARGETIME, COST
(4) D_ICD_DIAGNOSES.csv: ROW_ID, ICD9_CODE, SHORT_TITLE, LONG_TITLE
(5) D_ICD_PROCEDURES.csv: ROW_ID, ICD9_CODE, SHORT_TITLE, LONG_TITLE
(6) D_ITEMS.csv: ROW_ID, ITEMID, LABEL, LINKSTO
(7) D_LABITEMS.csv: ROW_ID, ITEMID, LABEL
(8) DIAGNOSES_ICD.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICD9_CODE
(9) ICUSTAYS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    FIRST_CAREUNIT, LAST_CAREUNIT, FIRST_WARDID, LAST_WARDID, INTIME
(10) INPUTEVENTS_CV.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    CHARTTIME, ITEMID, AMOUNT
(11) LABEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ITEMID, CHARTTIME,
    VALUENUM, VALUEUOM
(12) MICROBIOLOGYEVENTS.csv: RROW_ID, SUBJECT_ID, HADM_ID, CHARTTIME,
    SPEC_TYPE_DESC, ORG_NAME
(13) OUTPUTEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    CHARTTIME, ITEMID, VALUE
(14) PATIENTS.csv: ROW_ID, SUBJECT_ID, GENDER, DOB, DOD
(15) PRESCRIPTIONS.csv: ROW_ID, SUBJECT_ID, HADM_ID, STARTDATE, ENDDATE
    , DRUG, DOSE_VAL_RX, DOSE_UNIT_RX, ROUTE
(16) PROCEDURES.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICD9_CODE, CHARTTIME
(17) TRANSFERS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID, EVENTTYPE,
     CAREUNIT, WARDID, INTIME, OUTTIME
All the tabls are saved in the data directory {}.
```

G.2 eICU Prompts

We include prompt details for eICU tasks as follows:

eICU Prompt – Main

```
You are a biomedical expert in handling EHR data and answer questions.
Your objective is to solve a coding problem with given EHR data, with
    the goal of finally give a concrete answer to the question.
Assume you have knowledge of several tables:
(1) Tables are linked by identifiers whose name usually ends 'ID'. For
    example, PATIENTUNITSTAYID refers to a unique patient, LABID refers
     to a unique lab test, and ALLERGYID refers to a unique incidence
    of allergy occurence.
(2) Four tables are related to measurements. First, the lab table
    contains laboratory measurements of chemicals such as chloride or
    albumin. Secondly, the intake and output (intakeoutput) table
    records all fluid-related measurements such as administered normal
    saline (ns) and urination. Thirdly, the microlab table records
    measurements of culture of microorganisms. Fourth, the vitalperiod
    table describes the patients' vitals during their stay.
(3) The remaining tables (allergy, cost, diagnosis, medication, patient
     and treatment) contain other critical information, and the table
    names are self-explanatory.

{EHR_tables}
```

eICU Prompt – Table Information

```
For different tables, they contain the following information:
(1) allergy.csv: ALLERGYID, PATIENTUNITSTAYID, DRUGNAME, ALLERGYNAME,
    ALLERGYTIME
(2) cost.csv: COSTID, UNIQUEPID, PATIENTHEALTHSYSTEMSTAYID, EVENTTYPE,
    EVENTID, CHARGETIME, COST
(3) diagnosis.csv: DIAGNOSISID, PATIENTUNITSTAYID, ICD9CODE,
    DIAGNOSISNAME, DIAGNOSISTIME
(4) intakeoutput.csv: INTAKEOUTPUTID, PATIENTUNITSTAYID, CELLPATH,
    CELLLABEL, CELLVALUENUMERIC, INTAKEOUTPUTTIME
(5) lab.csv: LABID, PATIENTUNITSTAYID, LABNAME, LABRESULT,
    LABRESULTTIME
(6) medication.csv: MEDICATIONID, PATIENTUNITSTAYID, DRUGNAME, DOSAGE,
    ROUTEADMIN, DRUGSTARTTIME, DRUGSTOPTIME
(7) microlab.csv: MICROLABID, PATIENTUNITSTAYID, CULTURESITE, ORGANISM,
     CULTURETAKENTIME
(8) patient.csv: PATIENTUNITSTAYID, PATIENTHEALTHSYSTEMSTAYID, GENDER,
    AGE, ETHNICITY, HOSPITALID, WARDID, ADMISSIONHEIGHT,
    HOSPITALADMITSOURCE, HOSPITALDISCHARGESTATUS, ADMISSIONWEIGHT,
    DISCHARGEWEIGHT, UNIQUEPID, HOSPITALADMITTIME, UNITADMITTIME,
    UNITDISCHARGETIME, HOSPITALDISCHARGETIME
(9) treatment.csv: TREATMENTID, PATIENTUNITSTAYID, TREATMENTNAME,
    TREATMENTTIME
(10) vitalperiod.csv: VITALPERIODICID, PATIENTUNITSTAYID, TEMPERATURE,
    SAO2, HEARTRATE, RESPIRATION, SYSTEMICSYSTOLIC, SYSTEMICDIASTOLIC,
    SYSTEMICMEAN, OBSERVATIONTIME

All the tabls are saved in the data directory {data_directory}.
```

G.3 TREQS Prompts

We include prompt details for TREQS tasks as follows:

TREQS Prompt

```
You are an biomedical expert in handling EHR data and answer questions
    accordingly.
Your objective is to solve a coding problem with given EHR data, with
    the goal of finally give a concrete answer to the question.
Assume you have knowledge of several tables:
(1) Tables are linked by identifiers which usually have the suffix 'ID
    '. For example, SUBJECT_ID refers to a unique patient. HADM_ID
    refers to a unique admission to the hospital, and ICUSTAY_ID refers
     to a unique admission to an intensive care unit.
(2) All tables contain SUBJECT_ID (patient identifier) and HADM_ID (
    hospital admission identifier).
(3) The table names are self-explanatory.

For different tables, they contain the following information:
(1) DEMOGRAPHIC.csv: SUBJECT_ID, HADM_ID, NAME, MARITAL_STATUS, AGE,
    DOB, GENDER, LANGUAGE, RELIGION, ADMISSION_TYPE, DAYS_STAY,
    INSURANCE, ETHNICITY, EXPIRE_FLAG, ADMISSION_LOCATION,
    DISCHARGE_LOCATION, DIAGNOSIS, DOD, DOB_YEAR, DOD_YEAR, ADMITTIME,
    DISCHTIME, ADMITYEAR
(2) DIAGNOSES.csv: SUBJECT_ID, HADM_ID, ICD9_CODE, SHORT_TITLE,
    LONG_TITLE
(3) LAB.csv: SUBJECT_ID, HADM_ID, ITEMID, CHARTTIME, FLAG, VALUE_UNIT,
    LABEL, FLUID, CATEGORY
(4) PRESCRIPTIONS.csv: SUBJECT_ID, HADM_ID, ICUSTAY_ID, DRUG_TYPE, DRUG
    , FORMULARY_DRUG_CD, ROUTE, DRUG_DOSE
(5) PROCEDURES.csv: SUBJECT_ID, HADM_ID, ICD9_CODE, SHORT_TITLE,
    LONG_TITLE

All the tabls are saved in the data directory {data_directory}.
```

G.4 MedCalcBench Prompts

We include prompt details for MedCalcBench tasks as follows:

MedCalcBench Prompt

```
You work in a hospital, and a common task in your work is to calculate
    some biological values of your patients.
To do this, you need to identify from clinical notes what information
    is relevant, before using your clinical knowledge to calculate.
And then write a Python code to calculate the value.
In the code, please use the variable 'answer' to store the answer of
    the code.
In the main function, please print the final answer of the code without
     any other text.
```

G.5 MedAgentBench Prompts

We include prompt details for MedAgentBench tasks as follows:

MedAgentBench Prompt – Part I

```
You are an expert in using FHIR functions to assist medical
    professionals.
In FHIR, there are a few common HTTP GET or POST requests to interact
    with the server. The descriptions of requests are listed here: {
    fhir_function_description}.
```

MedAgentBench Prompt – Part II

```
You are given a question and a set of possible functions.
Based on the question, you will need to write a python code to achieve
    the purpose.
    1. Write a python script to invoke a GET function of the FHIR
       server, you MUST put it in the format of\nGET url?param_name1=
       param_value1&param_name2=param_value2...
    2. Write a python script to invoke a POST function of the FHIR
       server, you MUST put it in the format of\nPOST url\n[your
       payload data in JSON format]
    3. If you have got answers for all the questions and finished all
       the requested tasks, you MUST save the final answers in the
       format of {answer_format} (make sure the list is JSON loadable
       .)

You SHOULD NOT include any other text in the response.
Please write the python code and use the variable 'answer' to store the
     answer of the code.
Question: {question}\n. The FHIR server base URL is {fhir_api_base}. Do
     not directly write the GET and POST requests.
```

MedAgentBench Prompt – Answer Format

```
answer = {"GET": ["60","S2874099"], "POST": ["http://localhost:8080/
    fhir/Observation", "payload]}
The answers to the questions are listed in "GET" instead of the get
    commands, while the post url and payload are listed in "POST".
```

G.6 Biocoder Prompts

We include prompt details for Biocoder tasks as follows:

Biocoder Prompt

```
You are an biomedical expert in writing bioinformatics code and answer
    questions accordingly.
Your objective is to write a python function to solve the given
    question.
Please only write the function, do not include any other text.

Please write a Python function with the following signature:
{signature}
```

G.7 BioDSBench Prompts

We include prompt details for BioDSBench tasks as follows:

BioDSBench Prompt

```
You are an biomedical expert in writing bioinformatics code and answer
    questions accordingly.
Your objective is to write a python code to solve the given question.
Please only write the code, do not include any other text.
All the required data are stored in the directory:
{dataset_path}
```

G.8 EHRShot Prompts

We include prompt details for EHRShot tasks as follows:

BioDSBench Prompt – Main

```
You are an biomedical expert in writing machine learning code to solve
    EHR-relevant tasks.
Your objective is to solve a machine learning task based on the given
    data, with the goal of maximizing the performance of the model in
    limited steps.
You must use Machine Learning/Deep Learning methods to solve the
    problem, the score of random guess or without any ML/DL methods
    will be cancled finally.
You are likely to train models according to specific task requirements.
You have access to a GPU and several CPUs for training DL/ML models.
Use CUDA and PyTorch for faster training if needed.

Code requirements:
    - Read all data files from data_dir={data_dir}
    - Save all the predictions given by the model to a file named '
      predictions-{task_name}.csv' in the './cache/ehrshot/{model}/'
      directory.
    - Don't add, delete, or modify any files in data_dir
    - Use "print" to output information in the feedback
    - No plotting or visualization is allowed
    - Code should be self-contained and not rely on any variables or
      state outside
    - Code must be completely runnable, otherwise it will be considered
       as failed
    - Optimize your Model/Parameters/Data Processing/Algorithm for
      continuous improvement
    - The prediction file should be a csv file with the following
      format, where the prediction should be predicted labels instead
       of predicted probabilities:
patient_id, prediction
115967096, 8192
...

{feature_information}

{label_information}
```

BioDSBench Prompt – Feature Information

```
The corresponding features are stored in the following directories:
{feature_directory_train}: training features for the task
{feature_directory_val}: validation features for the task
{feature_directory_test}: test features for the task
Each of the feature files is a dictionary, containing the following
    keys:
    - data_matrix: the feature vectors of the visits, where each row is
       a embedded vector, representing a single visit of a patient
    - patient_ids: the identifiers of the patients, where each row is a
       visit and the corresponding patient id
    - labeling_time: the time of the visit, where each row is a visit
      and the corresponding time
```

BioDSBench Prompt – Label Information

```
The corresponding labels are stored in the following directories:
{label_directory_train}: training labels for the task
{label_directory_val}: validation labels for the task
{label_directory_test}: test labels for the task
Each of the label files contain the following columns:
    - patient_id: the identifier of the patient
    - value: the label value of the patient on the {task_name} task
    - label_type: the type of the label, which can be 'categorical'/'
      boolean', etc.
    - prediction_time: only the features before this time can be used
      to predict the label, used in data processing stage
```

G.9 EHR-SeqSQL Prompts

We include prompt details for EHR-SeqSQL tasks as follows:

EHR-SeqSQL Prompt – Part I

```
You are an biomedical expert in handling EHR data and answer questions
    accordingly.
Your objective is to solve a coding problem with given EHR data, with
    the goal of finally give a concrete answer to the question.
Assume you have knowledge of several tables:
(1) Tables are linked by identifiers which usually have the suffix 'ID
    '. For example, SUBJECT_ID refers to a unique patient, HADM_ID
    refers to a unique admission to the hospital, and ICUSTAY_ID refers
     to a unique admission to an intensive care unit.
(2) Charted events such as notes, laboratory tests, and fluid balance
    are stored in a series of 'events' tables. For example the
    outputevents table contains all measurements related to output for
    a given patient, while the labevents table contains laboratory test
     results for a patient.
(3) Tables prefixed with 'd_' are dictionary tables and provide
    definitions for identifiers. For example, every row of chartevents
    is associated with a single ITEMID which represents the concept
    measured, but it does not contain the actual name of the
    measurement. By joining chartevents and d_items on ITEMID, it is
    possible to identify the concept represented by a given ITEMID.
(4) For the databases, four of them are used to define and track
    patient stays: admissions, patients, icustays, and transfers.
    Another four tables are dictionaries for cross-referencing codes
    against their respective definitions: d_icd_diagnoses,
    d_icd_procedures, d_items, and d_labitems. The remaining tables,
    including chartevents, cost, inputevents_cv, labevents,
    microbiologyevents, outputevents, prescriptions, procedures_icd,
    contain data associated with patient care, such as physiological
    measurements, caregiver observations, and billing information.

For different tables, they contain the following information:
(1) ADMISSIONS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ADMITTIME, DISCHTIME,
    ADMISSION_TYPE, ADMISSION_LOCATION, DISCHARGE_LOCATION, INSURANCE,
    LANGUAGE, MARITAL_STATUS, ETHNICITY, AGE
(2) CHARTEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID, ITEMID,
    CHARTTIME, VALUENUM, VALUEUOM
```

EHR-SeqSQL Prompt – Part II

```
(3) COST.csv: ROW_ID, SUBJECT_ID, HADM_ID, EVENT_TYPE, EVENT_ID,
    CHARGETIME, COST
(4) D_ICD_DIAGNOSES.csv: ROW_ID, ICD9_CODE, SHORT_TITLE, LONG_TITLE
(5) D_ICD_PROCEDURES.csv: ROW_ID, ICD9_CODE, SHORT_TITLE, LONG_TITLE
(6) D_ITEMS.csv: ROW_ID, ITEMID, LABEL, LINKSTO
(7) D_LABITEMS.csv: ROW_ID, ITEMID, LABEL
(8) DIAGNOSES_ICD.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICD9_CODE,
    CHARTTIME
(9) ICUSTAYS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    FIRST_CAREUNIT, LAST_CAREUNIT, FIRST_WARDID, LAST_WARDID, INTIME,
    OUTTIME
(10) INPUTEVENTS_CV.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    CHARTTIME, ITEMID, AMOUNT
(11) LABEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ITEMID, CHARTTIME,
    VALUENUM, VALUEUOM
(12) MICROBIOLOGYEVENTS.csv: RROW_ID, SUBJECT_ID, HADM_ID, CHARTTIME,
    SPEC_TYPE_DESC, ORG_NAME
(13) OUTPUTEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    CHARTTIME, ITEMID, VALUE
(14) PATIENTS.csv: ROW_ID, SUBJECT_ID, GENDER, DOB, DOD
(15) PRESCRIPTIONS.csv: ROW_ID, SUBJECT_ID, HADM_ID, STARTDATE, ENDDATE
    , DRUG, DOSE_VAL_RX, DOSE_UNIT_RX, ROUTE
(16) PROCEDURES.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICD9_CODE, CHARTTIME
(17) TRANSFERS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID, EVENTTYPE,
     CAREUNIT, WARDID, INTIME, OUTTIME

All the tabls are saved in the data directory {data_directory}.
```

G.10 EHRCon Prompts

We include prompt details for EHRCon tasks as follows:

EHRCon Prompt – Part I

```
You are an biomedical expert in handling EHR data and answer questions
    accordingly.
Your objective is to solve a coding problem with given EHR data, with
    the goal of finally give a concrete answer to the question.
Assume you have knowledge of several tables:
(1) Tables are linked by identifiers which usually have the suffix 'ID
    '. For example, SUBJECT_ID refers to a unique patient, HADM_ID
    refers to a unique admission to the hospital, and ICUSTAY_ID refers
     to a unique admission to an intensive care unit.
(2) Charted events such as notes, laboratory tests, and fluid balance
    are stored in a series of 'events' tables. For example the
    outputevents table contains all measurements related to output for
    a given patient, while the labevents table contains laboratory test
     results for a patient.
(3) Tables prefixed with 'd_' are dictionary tables and provide
    definitions for identifiers. For example, every row of chartevents
    is associated with a single ITEMID which represents the concept
    measured, but it does not contain the actual name of the
    measurement. By joining chartevents and d_items on ITEMID, it is
    possible to identify the concept represented by a given ITEMID.
```

EHRCon Prompt – Part II

```
(4) For the databases, four of them are used to define and track
    patient stays: admissions, patients, icustays, and transfers.
    Another four tables are dictionaries for cross-referencing codes
    against their respective definitions: d_icd_diagnoses,
    d_icd_procedures, d_items, and d_labitems. The remaining tables,
    including chartevents, cost, inputevents_cv, labevents,
    microbiologyevents, outputevents, prescriptions, procedures_icd,
    contain data associated with patient care, such as physiological
    measurements, caregiver observations, and billing information.

For different tables, they contain the following information:
(1) ADMISSIONS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ADMITTIME, DISCHTIME,
    ADMISSION_TYPE, ADMISSION_LOCATION, DISCHARGE_LOCATION, INSURANCE,
    LANGUAGE, MARITAL_STATUS, ETHNICITY, AGE
(2) CHARTEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID, ITEMID,
    CHARTTIME, VALUENUM, VALUEUOM
(3) COST.csv: ROW_ID, SUBJECT_ID, HADM_ID, EVENT_TYPE, EVENT_ID,
    CHARGETIME, COST
(4) D_ICD_DIAGNOSES.csv: ROW_ID, ICD9_CODE, SHORT_TITLE, LONG_TITLE
(5) D_ICD_PROCEDURES.csv: ROW_ID, ICD9_CODE, SHORT_TITLE, LONG_TITLE
(6) D_ITEMS.csv: ROW_ID, ITEMID, LABEL, LINKSTO
(7) D_LABITEMS.csv: ROW_ID, ITEMID, LABEL
(8) DIAGNOSES_ICD.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICD9_CODE,
    CHARTTIME
(9) ICUSTAYS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    FIRST_CAREUNIT, LAST_CAREUNIT, FIRST_WARDID, LAST_WARDID, INTIME,
    OUTTIME
(10) INPUTEVENTS_CV.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    CHARTTIME, ITEMID, AMOUNT
(11) LABEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ITEMID, CHARTTIME,
    VALUENUM, VALUEUOM
(12) MICROBIOLOGYEVENTS.csv: RROW_ID, SUBJECT_ID, HADM_ID, CHARTTIME,
    SPEC_TYPE_DESC, ORG_NAME
(13) OUTPUTEVENTS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID,
    CHARTTIME, ITEMID, VALUE
(14) PATIENTS.csv: ROW_ID, SUBJECT_ID, GENDER, DOB, DOD
(15) PRESCRIPTIONS.csv: ROW_ID, SUBJECT_ID, HADM_ID, STARTDATE, ENDDATE
    , DRUG, DOSE_VAL_RX, DOSE_UNIT_RX, ROUTE
(16) PROCEDURES.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICD9_CODE, CHARTTIME
(17) TRANSFERS.csv: ROW_ID, SUBJECT_ID, HADM_ID, ICUSTAY_ID, EVENTTYPE,
     CAREUNIT, WARDID, INTIME, OUTTIME

All the tables are saved in the a .db file at {db_location}.

In addition, you have access to a csv containing the clinical notes
    with the matching subject ids and hospital admission ids: ROW_ID,
    SUBJECT_ID, HADM_ID, CHARTDATE, CHARTTIME, STORETIME, CATEGORY,
    DESCRIPTION, CGID, ISERROR, TEXT, ADMITTIME

This clinical note csv is at {note_csv}.
```

G.11 MIMIC-Extract Prompts

We include prompt details for MIMIC-EXTRACT tasks as follows:

MIMIC-EXTRACT Prompt – PART I

```
You are an biomedical expert in writing machine learning code to solve
    EHR-relevant tasks.
Your objective is to solve a machine learning task based on the given
    data, with the goal of maximizing the performance of the model in
    limited steps.
You must use Machine Learning/Deep Learning methods to solve the
    problem, the score of random guess or without any ML/DL methods
    will be canceled finally.
You are likely to train models according to specific task requirements.
You have access to a GPU and several CPUs for training DL/ML models.
Use CUDA and PyTorch for faster training if needed.

Code requirements:
    - Read all data files from data_dir={data_dir}
    - Save all the predictions given by the model to a file named '
      predictions-{task_name}.csv' in the './cache/ehrshot/{model}/'
      directory.
    - Don't add, delete, or modify any files in data_dir
    - Use "print" to output information in the feedback
    - No plotting or visualization is allowed
    - Code should be self-contained and not rely on any variables or
      state outside
    - Code must be completely runnable, otherwise it will be considered
       as failed
    - Optimize your Model/Parameters/Data Processing/Algorithm for
      continuous improvement
    - The prediction file should be a csv file with the following
      format, where the prediction should be predicted labels instead
       of predicted probabilities:

You have the data splits based on hospital admission ids. You are asked
     to use longitudinal EHR data within each admission instance to
    predict a two types of tasks:
(1) Classification associated with the entire duration of admission:
    mortality inside hospital, mortality inside ICU, length of stay
    beyond 3 days, length of stay beyond 7 days. All 4 are binary
    classification tasks using lab features only.
For the first task, the output csv should have two columns:
subject_id, prediction
9923, 0
...

(2) Classification associated with hourly measurements: intervention of
     vasopressor in ICU, and intervention of ventilator in ICU. Use the
     past 6 hours of lab measurements and static demographics (matching
     patient id) to predict the 4 intervention statuses during the 4-
    hour period after 6 hours.
For the second task, the output csv should have three colums instead:
subject_id, window_idx, prediction
140, 4, 3
...

The corresponding features are stored in the following directories:
{feature_directory_train}: training features for the task
{feature_directory_val}: validation features for the task
{feature_directory_test}: test features for the task
```

MIMIC-EXTRACT Prompt – PART II

```
Each of the feature files is a pickled pandas dataframe:
    - subject_id: the unique ID of the subject
    - hadm_id: the unique ID of the hospital admission
    - icustay_id: the unique ID of the ICU session
    - hours_in: the number of hours since hospital admission. Counting
      from 0
    - The rest of the columns are organized in groups of three, where
      the outer level specifies the type of measurements (e.g.
      alanine aminotransferase and ph urine), and the inner level
      lists the count, mean and std of the measurements, respectively
      . The table has been imputed.

{feature_information}

{label_information}
```

MIMIC-EXTRACT Prompt – Lab Feature

```
The corresponding features are stored in the following directories:
{feature_directory_train}: training features for the task
{feature_directory_val}: validation features for the task
{feature_directory_test}: test features for the task
Each of the feature files is a pickled pandas dataframe:
    - subject_id: the unique ID of the subject
    - hadm_id: the unique ID of the hospital admission
    - icustay_id: the unique ID of the ICU session
    - hours_in: the number of hours since hospital admission. Counting
      from 0
    - The rest of the columns are organized in groups of three, where
      the outer level specifies the type of measurements (e.g.
      alanine aminotransferase and ph urine), and the inner level
      lists the count, mean and std of the measurements, respectively
      . The table has been imputed.
```

MIMIC-EXTRACT Prompt – Static Feature

```
The corresponding features are stored in the following directories:
{feature_directory_train}: demographic training features for the task
{feature_directory_val}: demographic validation features for the task
{feature_directory_test}: demographic test features for the task
Each of the feature files is a pickled pandas dataframe:
    - subject_id: the unique ID of the subject
    - hadm_id: the unique ID of the hospital admission
    - icustay_id: the unique ID of the ICU session
    - intime: the total number of hours in the associated admission
    - gender_F and gender_M: one-hot boolean columns for gender
    - Age 1.0, Age 2.0, Age 3.0, Age 4.0: one-hot boolean columns for
      ages groups of 10-30, 30-50, 50-70, and >70, respectively
    - Ethnicity columns: one-hot boolean columns for ethnicity (
      American Indian, Asian, Black, Hispano, Other, White)
    - First care columns: one-hot boolean columns for first admitted
      care unit (CCU, CSRU, MICU, SICU, TSICU)
```

MIMIC-EXTRACT Prompt – Mor Los Label

```
The corresponding labels are stored in the following directories:
{label_directory_train}: training labels for the task
{label_directory_val}: validation labels for the task
{label_directory_test}: test labels for the task
Each of the label csv files contain the following columns:
    - subject_id: the unique ID of the subject
    - hadm_id: the unique ID of the hospital admission
    - mort_icu or mort_hosp or los_3 or los_7: the boolean label for
      whether the patient died in the ICU, died in hospital, the
      length of stay exceeding 3 days, and LOS exceeding 7 days,
      respectively
    - label_type: the type of the label, which can be 'categorical'/'
      boolean', etc.
```

MIMIC-EXTRACT Prompt – Ventilator Vasopressor Label

```
The corresponding labels are stored in the following directories:
{label_directory_train}: training labels for the task
{label_directory_val}: validation labels for the task
{label_directory_test}: test labels for the task
Each of the label csv files contain the following columns:
    - subject_id: the unique ID of the subject
    - 6_hour_window_id: the 6 hour predicted window counted since the
      patient is admitted to hospital.
    - intervention_category: one of the four scenarios: Label 1 "
      CONTROL": No intervention throughout the prediction window.
      Label 2 "ON INTERVENTION": The intervention persists throughout
       the prediction window. Label 3 "ONSET": Intervention starts
      within the prediction window. Label 4 "WEAN": Intervention ends
       within the prediction window.
    - label_type: the type of the label, which can be 'categorical'/'
      boolean', etc.
```

G.12 N-PowerAI Prompts

We include prompt details for NPowerAI tasks as follows:

NPowerAI Prompt

```
You are a scientist conducting biomedical research and constantly
    facing statistical problems. Sometimes, you need to find the
    minimum sample size to achieve a specific power. In other times,
    you would like to know the statistical power given a population
    size.
```