TissueCodePilot: A Code-Action Agent for AI-Assisted Spatial Tissue Analysis

Hung Q. Vo[1*], Huy Q. Vo[1*], Hong Zhao[2], Stephen T. C. Wong[2†], and Hien V. Nguyen[1†]

[1] Department of Electrical and Computer Engineering, University of Houston, USA
[2] Systems Medicine and Biomedical Engineering, Houston Methodist Hospital, USA
hqvo2@uh.edu
* Co-first authors, † Co-senior authors

Abstract. Spatial tissue image analysis is essential for quantifying cellular microenvironments and tissue architecture. Yet existing software typically exposes a fixed feature set, which may limit biomarker discovery. When required features are missing, bioscientists may require collaboration with computational experts, which can introduce delays and limit scalability. To address these limitations, we propose **TissueCodePilot**, a coding-and-reasoning agent that observes relevant information from the analysis environment and autonomously plans and executes actions to support flexible, on-demand spatial tissue analysis. Unlike prior coding agents that rely on detailed, iterative prompting, TissueCodePilot requires only a single minimal user prompt, making it accessible to bioscientists without extensive programming experience. To evaluate TissueCodePilot, we curate an expert-annotated dataset spanning three tissue types with multiple fields of view per tissue. Each field of view is paired with 50 questions that provide minimal prompt context and cover diverse spatial feature categories. In total, the dataset comprises 1,500 image–question pairs with corresponding ground-truth outputs. We will publicly release this benchmark to support future research as the first coding-agent benchmark for spatial tissue image analysis. On this new benchmark, TissueCodePilot substantially outperforms prompt-instruction coding agent baselines, boosting Success Rate from ∼0-2.4% to 18.5-35.4%, increasing pass@5 from at most 11.61% to 54.82-75.05%, and increasing pass@10 from at most 22.48% to 64.00–86.00%. For reproducibility, our code will be made publicly available at `https://github.com/hula-ai/TissueCodePilot`.

Keywords: Multiplexed Microscopy Imaging · Autonomous Coding Agents · Large Language Models · Spatial Tissue Analysis · Microscopy Image Analysis · Code Generation · Computational Pathology

1 Introduction

Understanding the spatial arrangement of cells and their microenvironments is central to many biomedical questions, including cancer and aging. However, most

spatial analysis tools offer a narrow set of predefined features, which can narrow biomarker exploration and hinder the discovery of alternative analyses. When needed analyses fall outside these capabilities, researchers often collaborate with computational experts to develop custom solutions—a process that can be time-consuming and difficult to scale. Moreover, extracting meaningful spatial features often requires combining classical image processing with modern AI and machine learning, adding further complexity.

Recent advances in large language models (LLMs), particularly code-generating models [10], offer a promising avenue for extending analysis pipelines via on-demand code synthesis. This can enable flexible, customized feature extraction beyond the built-in toolbox. A growing body of work has explored LLM-based agents that integrate code generation and tool use across domains [21], [9], [11], [17], including systems such as PaL [6], Voyager [19], and Data Interpreter [8]. While these approaches demonstrate the potential of programmatic reasoning, many operate in a largely single-pass or weakly iterative manner, with limited incorporation of intermediate outputs and execution feedback. When the initial attempt fails, the workflow is often treated as a failure rather than a signal for iterative debugging, limiting the ability to refine and recover from errors.

In contrast, human experts solve such tasks through iterative workflows—writing code, executing it, inspecting intermediate results, and refining their approach step by step.

To address these challenges, we propose **TissueCodePilot**, a multi-step code-and-reasoning agent for spatial tissue image analysis that operates through iterative interaction with the analysis environment. Given a single natural-language query, TissueCodePilot integrates task context—including tissue images, user-specified feature requests, intermediate outputs, and execution feedback—to progressively refine its plan, self-correct, and select subsequent actions, following an expert-like workflow to reliably complete the requested analysis.

Across three datasets and multiple LLM backbones, TissueCodePilot achieves approximately 3–10× higher pass@k and an order-of-magnitude improvement in Success Rate over the strongest prompt-based coding-agent baselines. Rather than proposing a new general-purpose agent architecture, this work identifies and validates environment-interactive coding as a necessary paradigm for spatial tissue analysis under minimal natural-language supervision.

The main contributions of this work are:

1. **TissueCodePilot.** We introduce and validate TissueCodePilot, an environment-interactive code-action agent that iteratively plans, executes, and debugs Python-based spatial tissue analyses from minimal natural-language queries, demonstrating the importance of execution-driven feedback in this domain.
2. **Benchmark.** We curate and will release the first coding-agent benchmark for spatial tissue analysis: **1,500** expert-annotated image–question pairs across three tissues with ground-truth outputs.
3. **Evaluation.** We conduct a systematic evaluation across multiple LLM backbones and tissue types, showing that enabling environment interaction leads

to substantial and consistent improvements over prompt-based coding agents in both Success Rate and pass@k.

2 Methodology

2.1 Task definition

Each instance consists of an image x (field of view) and a natural-language request q by user specifying the desired spatial analysis. The system must produce an output $\hat{y} \in \mathcal{Y}$ (e.g., scalar, string, or list) matching the ground-truth $y^\star$.

2.2 Inputs and perception artifacts

In addition to (x, q), methods may have access to cell-level perception artifacts: a segmentation result m and a cell classification result c. These artifacts are accessed as variables within the execution environment, enabling the agent to directly interact with segmentation maps and classification outputs through code. Depending on the method, (m, c) are either provided directly as part of the input - for prompt-instruction coding baselines, or obtained by calling tools during execution - for our proposed TissueCodePilot agent.

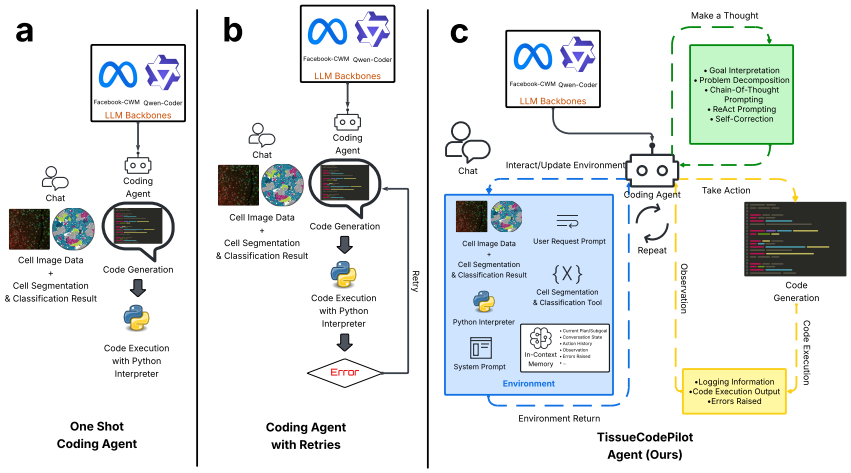

Fig. 1: Comparison of coding-agent workflows for spatial tissue analysis. (a–b) Prompt-instruction baselines that generate analysis code from the user query (optionally with retries/Chain-of-Thought/few-shot) and execute it with limited or no environment-driven revision. (c) TissueCodePilot (ours), an environment-interactive agent that iteratively observes intermediate results and interpreter feedback to re-plan, debug, and execute until producing the requested output.

2.3 Prompt-Instruction (PI) coding agent baselines

We compare our TissueCodePilot agent with two PI baselines (fig. 1): 1) **One-shot coding agent baseline:** Has access to a Python Interpreter environment and can execute code only once, to evaluate whether single-shot code execution is sufficient to solve tasks; 2) **Retry-on-error coding agent baseline:** Can additionally refine its code over multiple steps, correcting errors along the way.

1. **One-shot coding agent (prompt-only; provided m, c).** We make a single model call conditioned on (q, m, c) (and optionally x if included in the prompt). The model generates Python code that is executed in a Python interpreter and returns an output $\hat{y}$ in the required format.
2. **Retry-on-error coding agent (prompt-only; provided m, c).** We first run the one-shot procedure above. If the produced output is invalid (e.g., fails to execute, cannot be parsed, or violates the required output schema), we re-call the model with the previous attempt and the corresponding textual error message, up to a fixed retry budget R.

Our comparison is designed to isolate the effect of environment interaction. All methods operate under identical inputs, tools, and execution environments. The primary difference is whether the agent can iteratively observe execution feedback and revise its actions. This allows us to evaluate whether interaction, rather than prompt engineering, is the key factor for solving these tasks.

2.4 TissueCodePilot: A coding agent for spatial tissue analysis

TissueCodePilot is an environment-aware, multi-step coding agent that extends the Reasoning and Acting (ReAct) paradigm [23], [16], [1], [20] with domain-specific tooling to enable robust, on-demand spatial tissue analysis from minimal user input. The environment exposes domain-specific tools as callable Python functions. For example, `segment_cells(image)` returns a segmentation mask, and `classify_cells(mask)` returns cell-type labels. Outputs are stored in the workspace and can be reused in subsequent steps. The agent alternates between interpreting observations, updating a short-horizon plan, and acting through tool calls and Python execution, continuously leveraging environment feedback to make decision over successive steps. TissueCodePilot is implemented as an agent that operates within an analysis environment $\mathcal{E}$ that provides a *Python Interpreter* and callable tools for cell segmentation and cell classification (see Fig. 1). Three core novelties are:

1. Environment-aware interaction: it continuously leverages live interpreter feedback, intermediate artifacts (e.g., segmentation maps and cell classifications), and tool outputs to drive iterative plan revision.
2. Minimal-prompt capability: a concise user query yields an evolving short-horizon plan and executable Python actions that call domain tools, transform data, and perform on-the-fly sanity checks.

3. Strong zero-shot generalization: the agent robustly adapts to unseen tissue analyses and backbone models by relying on environment signals and iterative re-planning rather than pre-specified prompts.

At step t, the environment state $s_t \in \mathcal{S}$ includes the image x, the current workspace w_t (variables and intermediate artifacts), any available tool outputs (e.g., segmentation m_t and classification c_t), and execution traces (stdout/stderr and exceptions). TissueCodePilot receives an observation $o_t = \Omega(s_t)$, which includes the request q, pointers/handles to x, currently available artifacts, and interpreter feedbacks from all previously executed steps.

Action space: code as the interface. TissueCodePilot acts by emitting executable Python code snippets. Code may invoke tools, transform intermediate results, run sanity checks, and format the final output. We define the action space as $a_t \in \mathcal{A}, \mathcal{A} \triangleq \Sigma^\star$, where $\Sigma^\star$ denotes the set of finite strings over an alphabet Σ representing Python code. Executing an action updates the environment: $(s_{t+1}, \rho_t) = \mathcal{E}(s_t, a_t)$, where ρ_t contains interpreter feedback (returned values, logs, and/or an error object such as exception type and traceback). The next observation is $o_{t+1} = \Omega(s_{t+1})$. TissueCodePilot terminates by emitting a finalization command (e.g., `final(obj)`), which produces the answer $\hat{y}$.

Policy and feedback-driven revision. TissueCodePilot follows a multi-step policy π_θ with in-context memory h_t storing the request q, the expected output schema, a current plan, prior code actions $\{a_i\}_{i<t}$, tool outputs produced so far, and interpreter feedback. At each step, $a_t \sim \pi_\theta(\cdot \mid o_t, h_t)$. TissueCodePilot can revise its plan based on both *error feedback* (e.g., exceptions) and *non-error intermediate signals* (e.g., observations about previous steps, current step, and executed code output, etc.), up to a maximum interaction budget $T_{\max}$.

Tool-generated perception. Unlike methods that receive perception artifacts as fixed inputs, TissueCodePilot may call the segmentation/classification tools inside $\mathcal{E}$ to obtain (m, c) when they are not provided, and it may re-compute derived quantities as part of its iterative process.

2.5 Implementation details and reproducibility

We fix the maximum number of interaction steps for TissueCodePilot to $T_{\max}$ and the retry budget for the retry baseline to R. In this paper, for all ReAct-style experiments, each agent run is capped at 10 steps; the agent terminates either when it produces a final answer or when the step limit is reached. In this work, we define a run as a single interaction that begins with a bioscientist's question and ends when the agent outputs a final answer or reaches the step limit. For all LLM backbones, we use a sampling temperature of 0.7 and cap generation at 2048 tokens per agent step.

3 Experiment Results and Discussion

3.1 Dataset Construction and Ground Truth

We use publicly released datasets generated with the CosMx Spatial Molecular Imager (SMI) [13], spanning **Frontal Cortex**, **NSCLC**, and **Pancreas**. For

each tissue type, we evaluate **10** fields of view (FOVs) paired with approximately **50** expert-validated questions each, resulting in **1500** image–question pairs.

We first define a panel of spatial cellular features (e.g., nearest-neighbor distances, neighborhood composition, geometric descriptors, and graph-based measures) and use an LLM to convert each feature into a concise text question *without additional guidance or task instructions*, enforcing a minimal-prompt setting. We then manually audit each question to ensure a one-to-one correspondence with its target feature, followed by expert-biologist curation and validation. This design isolates the core challenge of mapping minimal natural-language queries to executable spatial analyses, all generated queries were manually reviewed and refined with a bioscientist to ensure domain relevance

Ground-truth answers are obtained through a human analysis workflow: ML engineers write and execute code for each image–question pair, including data loading and preprocessing, segmentation and cell-type labeling, and computation of the requested feature from a biologist. The resulting output is used as the reference ground-truth for evaluation.

Sample queries of spatial cellular features from our expert-curated dataset

1. What is the mean nearest-neighbor distance from epithelial cells to the closest other immune cell (non-T) within the FOV?
2. What fraction of T-cell centroids lie within merged epithelial-cluster polygons?
3. What is the astrocyte-to-non-astrocyte ratio within the entire FOV?
4. What is the coefficient of variation (CV) of astrocyte nearest-neighbor distances?
5. What is the spatial Pielou's evenness of astrocytes across a 10x10 grid over the FOV?
6. What is the mean nearest-neighbor distance between macrophages (macrophage-to-macrophage) within the FOV?

3.2 Quantitative Results

We benchmark TissueCodePilot against prompt-instruction (PI) coding-agent baselines. Note that, the agents have no prior exposure to the benchmark queries; all queries are novel to the system, and features are calculated on the fly.

Table 1 reports Success Rate and pass@k [3] across three tissue types and three backbone models. We report Success Rate (SR) as the fraction of instances where the predicted output matches the ground truth. Predictions are considered correct if they fall within a relative tolerance of ϵ (set to $1e-3$). Due to stochasticity, we also report pass@k, defined as the fraction of instances for which at least one of k independent runs produces a correct output. Despite never having seen the benchmark queries during development, TissueCodePilot attains the top performance on all dataset–model combinations, with substantial gains in both success rate and pass@k over prompt-instruction baselines.

TissueCodePilot dominates coding agent baselines. Across all backbones, PI-based agents (1-shot and retry-on-error) remain near zero Success Rate ($\approx 0\%$ to 2.4%) even when provided segmentation and cell-type artifacts (m, c). In contrast, TissueCodePilot attains $\approx 18.5\%$ to 35.4% Success Rate, indicating that closed-loop interaction with the environment (iterative execution, inspection, and revision) is crucial for reliably completing spatial analysis requests.

Large gains in pass@5 and pass@10. TissueCodePilot increases pass@5 to 54.82%−75.05% and pass@10 to 64%−86%, compared to at most 11.61% (pass@5) and 22.48% (pass@10) for the strongest PI baseline variants. The widening gap at higher k suggests that TissueCodePilot's step-wise feedback utilization converts additional context information into meaningful progress (debugging and re-planning), whereas prompt-only baselines saturate quickly because failures are often due to structural issues (mis-specified computations, incorrect intermediate assumptions, or brittle code) that are not resolved by minor re-sampling.

Table 1: Quantitative comparison of TissueCodePilot with prompt-instruction (PI) coding-agent baselines across three tissue datasets and three LLM backbones.

			Coding Agent with PI		**Coding Agent with PI - CoT**		**Coding Agent with PI - CoT Few-Shot**		**TissueCodePilot Agent (Ours)**
Model	**Dataset**	**Metric**	**1-shot**	**w/ Retries**	**1-shot**	**w/ Retries**	**1-shot**	**w/ Retries**	
Qwen3-Coder-30B -A3B-Instruct (≈30B params)	Frontal Cortex	Success Rate	1.60	1.70	0.83	1.28	2.20	2.40	**28.20**
		pass@5/10	6.83/12.00	8.29/16.07	5.83/10.00	6.40/12.61	9.39/16.00	11.61/22.48	**60.35/70.00**
	NSCLC	Success Rate	1.40	1.47	2.00	1.89	1.40	1.50	**35.40**
		pass@5/10	6.56/12.00	7.39/14.60	8.39/14.00	9.35/18.20	6.56/12.00	7.43/14.52	**75.05/86.00**
	Pancreas	Success Rate	1.25	1.51	0.50	0.89	1.25	1.49	**30.19**
		pass@5/10	4.82/6.00	7.45/14.40	2.50/4.00	4.53/8.94	5.54/8.00	7.39/14.56	**59.37/64.00**
GLM-4.5-Air (≈106B params)	Frontal Cortex	Success Rate	0.20	0.50	0.00	0.31	0.00	0.30	**19.20**
		pass@5/10	1.00/2.00	2.48/4.89	0.00/0.00	1.49/3.09	0.00/0.00	1.55/3.02	**61.07/80.00**
	NSCLC	Success Rate	0.40	0.70	0.20	0.70	0.40	0.70	**20.20**
		pass@5/10	2.00/4.00	3.57/7.13	1.00/2.00	3.57/7.12	2.00/4.00	3.45/6.95	**61.36/80.00**
	Pancreas	Success Rate	0.25	0.50	0.25	0.50	0.25	0.51	**24.00**
		pass@5/10	1.25/2.00	2.57/5.08	1.25/2.00	2.52/5.01	1.25/2.00	2.52/5.03	**64.82/74.00**
Facebook-CWM (≈32B params)	Frontal Cortex	Success Rate	1.00	1.30	1.00	1.31	0.60	0.90	**21.80**
		pass@5/10	5.00/10.00	6.48/12.42	4.56/8.00	6.43/12.64	3.00/6.00	4.47/9.00	**61.52/78.00**
	NSCLC	Success Rate	1.40	1.51	0.80	1.09	1.00	1.29	**26.20**
		pass@5/10	6.56/12.00	7.42/14.52	4.00/8.00	5.43/10.45	5.00/10.00	6.36/12.33	**68.18/78.00**
	Pancreas	Success Rate	1.25	1.48	1.50	1.49	0.50	0.80	**18.50**
		pass@5/10	6.25/10.00	7.45/14.62	6.79/10.00	7.51/14.50	2.50/4.00	4.01/8.04	**54.82/66.00**

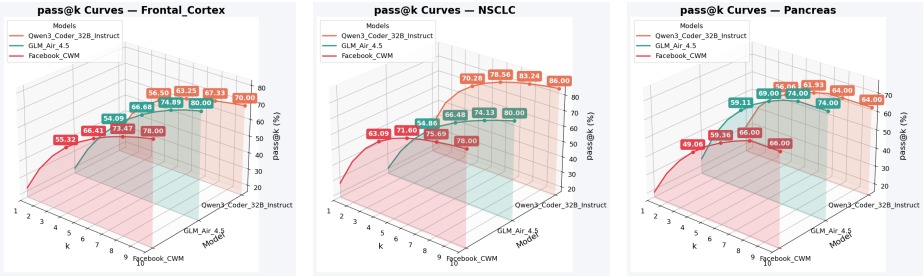

Fig. 2: Pass@k curves for various coding LLM backbones evaluated on three tissue types. Panels: Left-Frontal Cortex; Center-NSCLC; Right-Pancreas.

Retries, Chain-of-Thought (CoT), and few-shot provide limited benefit without interaction. Within the PI family, adding retries yields small improvements, while CoT [22] and few-shot prompting [2] are inconsistent and sometimes degrade performance. This pattern indicates that, performance is bottlenecked less by insufficient high-level task specification and more by successfully executing and validating multi-step computations with iterative environment feedback.

Consistency across tissues and models. The relative improvements hold across Frontal Cortex, NSCLC, and Pancreas. For example, with Qwen3-Coder-30B-Instruct [18], TissueCodePilot reaches 28.20%, 35.40%, and 30.19% Success Rate on the three tissues, respectively, while the best PI baseline stays $\leq$ 2.40%. Similar trends hold for GLM-4.5-Air [24] and Facebook-CWM [5], demonstrating that TissueCodePilot's gains are not model-specific but stem from the interactive agent design.
Pass@k curves of TissueCodePilot on three tissue-type datasets We plot the pass@k curves for TissueCodePilot across three tissue types using different Coding LLM backbones in Figure 2. Across tissues and backbones, pass@k monotonically increases with k, reflecting higher success rates as more retries or larger k-values are allowed. The curves tend to converge as k becomes large, suggesting that simply increasing retries yields diminishing returns beyond a certain threshold. Future works could retrain TissueCodePilot to improve its robustness [14], [4], [12], [15], [7].

3.3 Qualitative Results

Figure 3 shows a representative TissueCodePilot execution trace. TissueCodePilot follows a closed-loop workflow: it inspects the user request and microscopy image, checks which perception artifacts are already available, calls segmentation and cell-type classification tools when needed, and then iteratively executes Python to compute the requested spatial feature. Intermediate overlays and printed outputs serve as sanity checks, while logs and exceptions provide feedback for debugging and re-planning. In contrast to prompt-only coding baselines that generate code once (or rely on limited retries), TissueCodePilot uses these environment observations to refine subsequent actions until it produces a final answer or reaches the step limit.

4 Conclusion

We introduced **TissueCodePilot**, an environment-interactive code-action agent for on-demand spatial tissue analysis from minimal natural-language queries. Given a one-time request from a bioscientist, TissueCodePilot iteratively executes and debugs Python actions using tool outputs and interpreter feedback, enabling flexible spatial feature computation beyond fixed-function software. We also curated and will release the first benchmark for this setting, with **1,500** expert-validated image–question pairs across three tissues. Across multiple LLM backbones and tissues, TissueCodePilot consistently outperforms prompt-instruction coding baselines, improving Success Rate from $\sim$0–2.4% to **18.5–35.4%** and pass@5/pass@10 from at most 11.61%/22.48% to **54.82–75.05%/64–86%**. Our findings suggest that enabling agents to iteratively interact with execution environments is not merely beneficial but necessary for reliable spatial tissue analysis under minimal supervision.

Acknowledgements. This work is supported by the National Institutes of Health (NIH) under Grants R01DK134055 and R01AG071496; T. T. & W. F. Chao Foundation; and John S. Dunn Research Foundation.

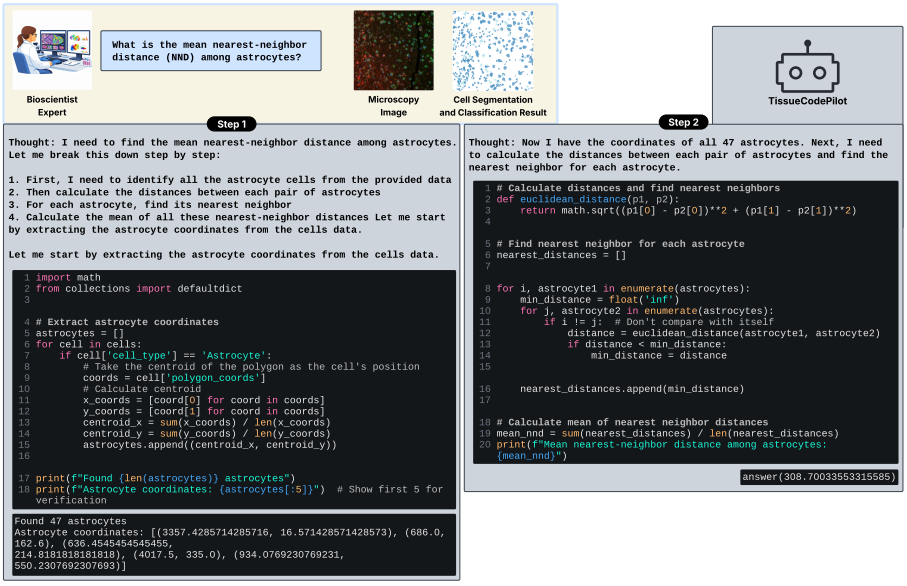

Fig. 3: Qualitative Result: Example TissueCodePilot run on a bioscientist query.

Disclosure of Interests. The authors declare that they have no competing interests.

References

1. Acharya, D.B., Kuppan, K., Divya, B.: Agentic ai: Autonomous intelligence for complex goals—a comprehensive survey. IEEE Access **13**, 18912–18936 (2025)
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
3. Chen, M.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
4. Cheng, M., Ouyang, J., Yu, S., Yan, R., Luo, Y., Liu, Z., Wang, D., Liu, Q., Chen, E.: Agent-r1: Training powerful llm agents with end-to-end reinforcement learning (2025), `https://arxiv.org/abs/2511.14460`
5. Copet, J., Carbonneaux, Q., Cohen, G., Gehring, J., Kahn, J., Kossen, J., Kreuk, F., McMilin, E., Meyer, M., Wei, Y., et al.: Cwm: An open-weights llm for research on code generation with world models. arXiv preprint arXiv:2510.02387 (2025)
6. Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., Neubig, G.: Pal: Program-aided language models. In: International conference on machine learning. pp. 10764–10799. PMLR (2023)
7. Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al.: Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948 (2025)

8. Hong, S., Lin, Y., Liu, B., Liu, B., Wu, B., Zhang, C., Li, D., Chen, J., Zhang, J., Wang, J., et al.: Data interpreter: An llm agent for data science. In: Findings of the Association for Computational Linguistics: ACL 2025. pp. 19796–19821 (2025)
9. Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S.K.S., Lin, Z., et al.: Metagpt: Meta programming for a multi-agent collaborative framework. In: The twelfth international conference on learning representations (2023)
10. Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S.: A survey on large language models for code generation. ACM Transactions on Software Engineering and Methodology **35**(2), 1–72 (2026)
11. Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., Zeng, A.: Code as policies: Language model programs for embodied control. In: 2023 IEEE International conference on robotics and automation (ICRA). pp. 9493–9500. IEEE (2023)
12. Luo, M., Jain, N., Singh, J., Tan, S., Patel, A., Wu, Q., Ariyak, A., Cai, C., Venkat, T., Zhu, S., Athiwaratkun, B., Roongta, M., Zhang, C., Li, L.E., Popa, R.A., Sen, K., Stoica, I.: Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl (2025), notion Blog
13. NanoString Technologies: Nanostring. `https://nanostring.com/`, accessed: 2026-01-20
14. NVIDIA: Nemo gym: An open source library for scaling reinforcement learning environments for llm. `https://github.com/NVIDIA-NeMo/Gym` (2025), gitHub repository
15. Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. Advances in neural information processing systems **36**, 53728–53741 (2023)
16. Sahoo, P., Singh, A.K., Saha, S., Jain, V., Mondal, S., Chadha, A.: A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927 **1** (2024)
17. Surís, D., Menon, S., Vondrick, C.: Vipergpt: Visual inference via python execution for reasoning. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 11888–11898 (2023)
18. Team, Q.: Qwen3 technical report (2025), `https://arxiv.org/abs/2505.09388`
19. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291 (2023)
20. Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., Ji, H.: Executable code actions elicit better llm agents. In: Forty-first International Conference on Machine Learning (2024)
21. Wang, X., Li, S., Ji, H.: Code4struct: Code generation for few-shot event structure prediction. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 3640–3663 (2023)
22. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems **35**, 24824–24837 (2022)
23. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K.R., Cao, Y.: React: Synergizing reasoning and acting in language models. In: The eleventh international conference on learning representations (2022)
24. Zeng, A., Lv, X., Zheng, Q., Hou, Z., Chen, B., Xie, C., Wang, C., Yin, D., Zeng, H., Zhang, J., et al.: Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. arXiv preprint arXiv:2508.06471 (2025)