

Building Reliable LLM Coding Agents for Mobile Health Data Analysis

Qiang Yang*
qiang.yang@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Alexandra Loh*
alexandralohw1@gmail.com
University of Cambridge
Cambridge, UK

Yuwei Zhang
yz798@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Yang Liu
yl25r@fsu.edu
Florida State University
Tallahassee, USA

Cecilia Mascolo
cecilia.mascolo@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Abstract

Mobile health data analysis requires end-to-end signal processing and machine learning pipelines over heterogeneous, noisy, and clinically sensitive sensor data. Although LLM coding agents show strong promise for automating software development, we find that mobile health tasks expose a different reliability problem: agents must sustain long-horizon, iterative debugging across preprocessing, feature extraction, model selection, evaluation, and interpretation, where small mistakes in data handling or evaluation design can invalidate the result. This paper presents an experience-driven study of how to design reliable LLM coding agents for mobile health data analysis and identifies practical design lessons for avoiding pollution from long conversations and remaining trapped in unsuccessful refinement loops. Across multimodal mobile health tasks, these design choices improve task success and robustness compared with more generic prompting and coding-assistant baselines. Our findings suggest that reliability in mobile health coding agents depends on carefully managing how agents remember, revise, and recover during iterative analysis.

CCS Concepts

• **Computing methodologies** → **Intelligent agents; Machine learning**; • **Applied computing** → **Health informatics**; • **Human-centered computing** → *Ubiquitous computing*.

Keywords

Mobile health; LLM; Coding agents

ACM Reference Format:

Qiang Yang, Alexandra Loh, Yuwei Zhang, Yang Liu, and Cecilia Mascolo. 2026. Building Reliable LLM Coding Agents for Mobile Health Data Analysis. In *Companion of the 2026 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp Companion '26)*, October 11–15, 2026, Shanghai, China. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3798063.3839193>

*Co-primary authors.



This work is licensed under a Creative Commons Attribution 4.0 International License. *UbiComp Companion '26, Shanghai, China*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2533-3/2026/10
<https://doi.org/10.1145/3798063.3839193>

1 Introduction

Mobile health applications increasingly rely on wearable and mobile sensors to support health monitoring outside clinical settings [4]. Signals such as electrocardiogram (ECG), photoplethysmography (PPG), inertial measurement unit (IMU) streams, audio, and other passive sensing data can support tasks ranging from activity recognition to screening and longitudinal health assessment [24]. This shift is attractive because it makes health monitoring more continuous and less dependent on specialized clinical equipment.

At the same time, it makes data analysis substantially harder. Mobile health datasets are often heterogeneous, multimodal, noisy, and incomplete [2]. Building a useful analysis pipeline therefore requires many coupled decisions: how to parse unfamiliar data formats, clean signals, segment time series, extract effective features, handle class imbalance, split data without leakage, select models, choose clinically meaningful metrics, and explain predictions [11, 19]. As a result, developing mobile health analysis pipelines often depends on people who combine programming expertise with domain knowledge in signal processing, machine learning, and health evaluation.

Large language models (LLMs) appear to offer a natural way to reduce this development burden. They can translate high-level goals into code and explain intermediate results. Recent work has explored LLMs for wearable and mobile health data, including foundation models for wearable signals [19], sensor-language models [32] and personal health agents [3, 11]. However, many of these systems focus on representing or interpreting sensor data rather than generating complete executable analysis pipelines. Directly asking an LLM to reason over raw sensor streams is also problematic because time-series data are long, continuous, and expensive to tokenize, and language models still struggle with zero-shot reasoning over such data [16]. In health settings, sending raw sensor data to remote model APIs also raises privacy concerns [12].

An alternative is to use the LLM as a coding agent [29]. Commercial coding agents and LLM-based coding systems such as Codex [20] and Claude Code [1] make it increasingly practical to generate executable analysis code from natural language instructions. But generic coding agents do not necessarily have the domain knowledge needed for mobile health. Generated programs may be syntactically valid while still making invalid analysis choices, such as leaking subjects across train and test splits, optimizing accuracy under severe class imbalance, or ignoring sensitivity and AUC in a screening task [14].

We therefore built an end-to-end mobile health coding agent that uses an LLM to generate and refine executable analysis pipelines over raw sensor data. During the development of the mobile health agent, we repeatedly observed three failure patterns. First, agents often repeated similar mistakes even after receiving execution feedback. A runtime error, failed validation check, or poor model output was not enough unless the relevant information was carried forward in a structured form. Second, longer context was not always beneficial. Persistent multi-turn conversations accumulated old prompts, code, explanations, and outputs from earlier phases, many of which were no longer relevant to the current subtask. This made the agent more expensive to run and sometimes made it less focused. Third, agents could become trapped in unsuccessful refinement loops: after several small edits around the same flawed approach, the model would continue patching symptoms rather than changing strategy. These observations led us to treat reliability as a context management problem.

We study three design choices for reliable mobile health coding agents. (1) Structured execution feedback preserves the information that is most useful for the next attempt, such as the previous code, execution result, and error messages, rather than relying on the model to remember an unstructured conversation. (2) Selective context management avoids sending the full interaction history to every prompt and instead attaches only the context needed for the current phase. (3) Rollback-based recovery detects repeated failure patterns and asks the agent to restart from a cleaner state, preventing local failure loops from consuming the entire attempt budget. These mechanisms are simple, but our experience suggests that they matter more than adding more open-ended autonomy.

We provide empirical evidence across ECG, IMU, audio, and multimodal mobile health tasks, comparing our design choices with more generic prompting and coding-assistant baselines. The insights from this work provide practical guidance for building reliable LLM coding agents for sensor analytics and digital health applications.

2 Lessons from Building the Agent

The system is implemented as a phase-based coding agent. We use LangGraph to orchestrate the workflow and conditional transitions between generation, execution, validation, and revision steps [13]. The LLM backend is Gemini 2.5 Pro accessed through the Gemini API [8]. For each code-producing phase, the agent prompts the model to generate Python code, executes the code in a Modal sandbox [17] with a copied version of the dataset, validates the output, and then either accepts the artifact or asks the LLM to revise it.

The agent takes a natural-language task description and a data description, then decomposes the work into phases: data exploration, data loading, optional web search for domain context, model exploration, model refinement, evaluation, and report generation. This decomposition is useful because mobile health pipelines have dependencies that generic coding prompts often leave implicit [31]. For example, the loading phase must preserve subject identifiers for later train-test splitting; the model phase must account for class imbalance; and the evaluation phase must report metrics such as sensitivity, specificity, AUC, and confidence where appropriate. The workflow therefore acts as a lightweight form of domain guidance

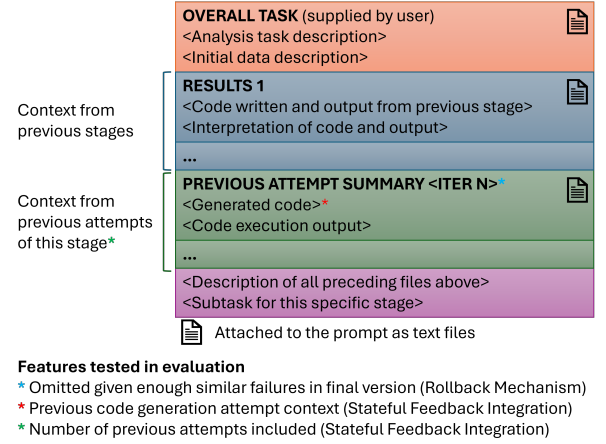


Figure 1: Task-state-oriented prompt structure. Instead of relying on the full chat history, each prompt is reconstructed from selected task state, phase outputs, previous-attempt feedback, and the current subtask.

rather than a fully autonomous open-ended agent. We omit implementation details here and instead focus the rest of the paper on the lessons that emerged from building and evaluating the agent.

2.1 Task State Matters More Than Chat History

Many coding agents are organized around a persistent conversation [7]. The model receives previous user messages, generated code, tool outputs, and explanations, then decides for itself which parts of the history matter. This design is natural for interactive programming, but mobile health analysis behaves less like an open-ended conversation and more like an iterative development workflow with phase-specific state. Treating this process as a single long conversation leaves the model to recover the relevant state from an increasingly noisy transcript.

Therefore, the information that matters for the next generation step is usually not the whole transcript. For example, the model refinement phase needs the loaded dataset summary and model exploration results; it does not necessarily need old failed attempts from data exploration. Similarly, when repairing code, the previous code and execution output are more useful than a vague conversational memory that something failed.

Our design reconstructs the prompt for each phase from structured state rather than appending to a global chat. As shown in Figure 1, each prompt contains four components: the overall task, selected context from previous phases, selected feedback from previous attempts in the current phase, and the current subtask instruction. Each attached context file is explicitly labeled so that the LLM can identify what role the information plays. This makes the agent less dependent on the LLM’s ability to search a long conversation and gives the system designer control over what information should influence the next step.

Table 1: Evaluation tasks.

Task	Modality	Main challenge
R-peak detection	ECG	Signal preprocessing
Activity recognition	IMU	Multiclass imbalance
COVID-19 classification	Audio	Noisy files and loading
Activity recognition	Multimodal	Heterogeneous sensors

2.2 More Context Is Not Always Better

Once context is represented as state, the next question is how much of it to include. Our experience was that more context was often worse. Persistent multi-turn conversations accumulated prompts, code, explanations, and outputs from earlier phases. Much of this information was stale or unrelated to the current subtask, but it was still sent to the model. This increased token usage and sometimes made the model less focused.

We therefore use selective context management [9]. For each phase, the agent attaches only the artifacts that are relevant to the current subtask. Outputs from accepted previous phases are included when they provide necessary information, such as data-loading summaries for model exploration or model-exploration metrics for model refinement. For iterative code repair within a phase, the agent includes a bounded number of previous attempts rather than the entire attempt history.

This principle of stateful feedback integration also applies to the type of feedback included. Each previous attempt may contain code, execution output, error messages, and the LLM’s explanation of its own changes. These are not equally useful. In our final design, the prompt emphasizes the previous code and execution output because they provide concrete evidence about what happened. Natural-language explanation comments are useful for interpretation, but they are less reliable as the only source of repair context.

2.3 Rollback Prevents Failure Loops

Selective context helps the model repair local errors, but it does not solve every failure. In some runs, the agent became trapped in a poor refinement trajectory. The model repeatedly patched symptoms of the same underlying problem, such as an incorrect loading assumption or an unsuitable processing strategy, without changing approach. Adding more execution feedback in these cases did not help; it often reinforced the same local solution path.

We therefore implemented rollback as a recovery mechanism. If the agent encounters the same type of failure repeatedly, such as syntax error, runtime error, timeout, or successful execution with invalid output, it clears the previous attempt history for that phase and asks the LLM to regenerate the solution from a cleaner state. The task description and relevant accepted phase outputs remain available, but the local failed trajectory is removed.

Rollback is not a replacement for execution feedback. It is a guardrail around iterative repair. The agent first attempts to improve code using structured feedback; rollback is triggered only when repeated failures suggest that local repair is no longer productive. Long-horizon mobile health analysis gives the agent many opportunities to enter a bad local trajectory, so the system should support both incremental repair and strategic restart.

Table 2: Model performance for successful runs. Sensitivity and specificity are omitted for multiclass tasks where they are less directly comparable.

Dataset	Bal. Acc.	Specificity	Sensitivity	F1	AUC
MIT-BIH	0.989	0.993	0.983	0.971	0.995
PAMAP2	0.663	–	–	0.687	0.805
Coswara	0.638	0.808	0.447	0.437	0.680
Harmony	0.579	–	–	0.599	0.865

Table 3: Execution success rate compared with general-purpose baselines.

Dataset	Our agent	Gemini prompting	GitHub Copilot
MIT-BIH	0.7	0.1	0.9
PAMAP2	0.6	0.3	0.9
Coswara	0.9	0.0	0.0
Harmony	0.7	0.0	0.8
Mean	0.725	0.100	0.650

3 Validating the Lessons

We use an exploratory feasibility evaluation to assess whether the agent can repeatedly construct executable mobile health analysis pipelines across these settings. As shown in Table 1, we evaluate the agent on four mobile health analysis tasks: ECG R-peak detection using MIT-BIH [18, 25], IMU activity recognition using PAMAP2 [23], audio-based COVID-19 classification using Coswara [26], and multimodal activity recognition using Harmony [21]. These tasks vary in modality, data format, preprocessing difficulty, and evaluation requirements. Each reported success rate is computed over ten runs per task unless otherwise stated, and this number is intended to expose recurring execution and functional failure modes rather than provide a high-precision estimate of small differences between systems; transient failures caused by external LLM service unavailability were excluded.

3.1 Overall Reliability

Table 2 summarizes the predictive performance across tasks. Performance is strongest on the ECG task and lower on the more heterogeneous activity-recognition and audio tasks. This is expected: the later tasks require more assumptions about noisy files, heterogeneous modalities, and multiclass evaluation. The point of these results is not that the agent always finds the best possible model, but that it can repeatedly produce executable and evaluable pipelines across different mobile health settings.

Baseline Comparison. We also compare the agent with two general-purpose baselines: Gemini prompting [5] and GitHub Copilot [6] in Table 3. A run is counted as execution-successful if the generated program completes without a fatal error; this does not by itself guarantee that the resulting pipeline satisfies all task requirements. The agent succeeds on a majority of runs for all four tasks, with a mean execution success rate of 0.725. Coswara has the highest success rate, partly because the original repository provided a

Table 4: Functional success rate compared with general-purpose baselines.

Dataset	Our agent	Gemini prompting	GitHub Copilot
MIT-BIH	0.7	0.1	0.5
PAMAP2	0.5	0.3	0.9
Coswara	0.7	0.0	0.0
Harmony	0.6	0.0	0.6
Mean	0.625	0.100	0.500

Table 5: Success rate for stateful feedback ablations.

Configuration	Success rate
Code with two previous iterations	0.9
Comments only, without previous code	0.6
One previous iteration included	0.9
Three previous iterations included	0.5

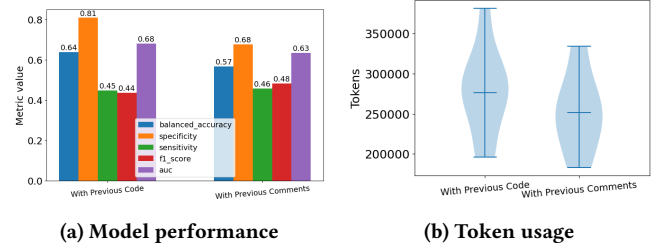
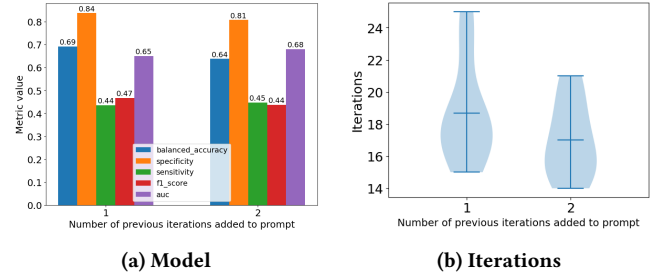
loading script that the agent could incorporate into its pipeline [26]; the remaining tasks still require the agent to infer signal structure, split data, train models, and report appropriate metrics.

Gemini prompting performs poorly because it cannot inspect execution feedback or repair code after incorrect assumptions about data format. GitHub Copilot is stronger, but its success is less consistent across datasets. In particular, both baselines struggle on Coswara, where compressed audio files and repository-specific loading steps make prompting or generic coding workflows fragile. Therefore, the following mechanism-level analyses are based on the Coswara task as it exposes several reliability issues that motivated our design: compressed audio files, repository-specific loading code, noisy user-contributed recordings, and repeated opportunities for the agent to make incorrect data-loading assumptions.

Execution alone does not show whether a generated pipeline satisfies the analysis task. We therefore also report functional success rate in Table 4. A run is functionally successful only if it executes, correctly loads all relevant data modalities, trains a classifier, and produces predictions with confidence values. Our agent has the highest mean functional success rate (0.625), compared with 0.100 for Gemini prompting and 0.500 for GitHub Copilot. The gap between execution and functional success is particularly visible for Copilot: several runs executed but did not fully process the data or produce a valid prediction report. This distinction matters in mobile health, where apparently successful code can still yield an incomplete or misleading analysis.

3.2 Stateful Feedback Integration

We next evaluate the feedback design used during iterative code generation. The key question is what information from previous attempts should be carried into the next prompt. Table 5 shows that including previous code and execution output is more effective than including only natural-language comments: success increases from 0.6 to 0.9. The code provides concrete repair context, such as exact


Figure 2: Token usage and model performance when stateful feedback includes previous code rather than only previous summary.

Figure 3: Effect of including one or two previous attempts in stateful feedback.

implementation choices and failure locations, while comments often summarize the previous attempt too vaguely.

Figure 2 gives the corresponding token and performance comparison. Including previous code costs slightly more tokens, but the successful runs produce better models overall, including higher balanced accuracy and specificity on the Coswara task. In this setting, the additional executable context is worth the token cost because it gives the model a more precise basis for repair.

The amount of feedback history also matters. Figure 3 shows that one and two previous attempts lead to similar model performance, while two attempts reduce the mean number of iterations from 19 to 17 and slightly reduce total token usage. However, including three previous attempts reduces success to 0.5 (Table 5), suggesting that older failed attempts can become distracting rather than helpful.

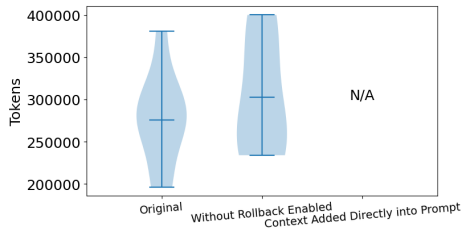
3.3 Rollback Mechanism

Finally, we evaluate rollback as a recovery mechanism for repeated local failures. In our implementation, rollback clears the previous attempt history for a phase when the agent repeatedly encounters the same class of error, then asks the LLM to regenerate from the task state and accepted phase context. Table 6 shows that rollback improves success on Coswara from 0.5 to 0.9. This supports the observation that some failures cannot be fixed by incremental patching: the agent sometimes needs to abandon a poor local trajectory and try a different implementation strategy.

Figure 4 reports token usage for the same ablation. Rollback slightly reduces token usage because it removes unproductive local history after repeated failures. The file-attachment result is also informative: inserting all background context directly into the

Table 6: Success rate for rollback and context-attachment ablations.

Configuration	Success rate
Rollback enabled, context attached as files	0.9
Rollback disabled	0.5
Context inserted directly into the prompt	0.0

**Figure 4: Token usage in the rollback and context-attachment ablation.**

prompt produced no successful runs in this ablation. For mobile health tasks, large context is necessary, but it must be structured and separated so that the model can distinguish stable task state from local repair feedback.

4 Related Work

LLM coding agents. LLM-based coding tools can now generate, edit, and execute code from natural language instructions, making them attractive for automating data analysis workflows [27–29]. However, code that appears plausible is not necessarily correct for mobile health [14]: a script may execute successfully while using the wrong subject split, ignoring class imbalance, or reporting metrics that are weak indicators for screening tasks. Our focus is therefore not whether LLMs can write code, but what agent design choices make iterative code generation reliable enough for mobile health analysis.

LLMs for mobile health. Recent systems have explored LLMs and foundation models for wearable sensing, mobile health interpretation, and personal health agents [11]. Much of this work focuses on representing sensor data, answering questions about health records, or generating user-facing insights [3, 32, 33]. Our setting is different: the agent must build executable signal-processing and machine-learning pipelines over raw mobile health data. This requires the agent to reason about data loading, preprocessing, validation, modeling, and evaluation as a connected workflow.

Iterative refinement. Recent work shows that LLM outputs can often be improved through iterative refinement. Self-Refine [15] and Reflexion [30] use self-critique or reflective memory to guide later attempts, while CRITIC [10] shows that external tools can provide useful signals for self-correction. Related memory-management work such as MemGPT treats long-horizon reasoning as a context-selection problem [22]. Our work is closest to this line, but focuses on a different question: for mobile health coding agents, reliability depends not on retaining more history, but on preserving the right

task state, carrying forward bounded executable feedback, and resetting local history when refinement becomes unproductive.

5 Discussion

The main lesson is that reliable mobile health coding agents should be designed around iteration management. The agent needs to maintain task state, carry forward concrete execution feedback, control how much previous context is reused, and recover when local refinement is no longer productive. These mechanisms are especially important in mobile health because data-loading, pre-processing, modeling, and evaluation decisions are tightly coupled: a script may execute successfully while still using an invalid split, ignoring class imbalance, or reporting weak metrics for a screening task. The practical design implication is that agent autonomy should be structured by the workflow and by explicit evidence from previous attempts, not by an ever-growing conversation alone.

There are several limitations. Our evaluation covers only four mobile health datasets, and all tasks are supervised machine-learning pipelines rather than broader clinical reasoning or deployment scenarios. Our ten-run evaluation per condition is intended as a feasibility study and does not support precise estimates of small performance differences; larger repeated studies are needed to quantify variance and statistical significance. We also did not use the strongest possible LLM or the most advanced coding-agent framework. More recent coding agents with very long context windows, stronger tool use, and better planning may improve absolute success rates on these tasks. However, we expect the lessons to remain useful beyond this particular implementation: even with stronger models, mobile health agents still need to decide what task state to preserve, what context to exclude, how to interpret execution feedback, and when to restart from a failed trajectory.

6 Conclusion

This paper studies what makes LLM coding agents reliable for mobile health data analysis. Through building a mobile health coding agent, we found that the core challenge is not simply generating code, but managing long-horizon iterative context across noisy data, domain-specific evaluation, and dependent workflow phases. Structured execution feedback, stateful feedback integration, and rollback recovery provide simple but practical mechanisms for improving reliability. These lessons can inform future LLM-powered agents for sensor analytics and digital health.

Acknowledgments

This work was supported by EPSRC grant EP/Z53447X/1.

References

- [1] Anthropic. [n. d.]. Claude Code. <https://claude.com/product/claude-code>
- [2] Luca Cossu, Giacomo Cappon, and Andrea Facchinetti. 2024. Adaptive and self-learning Bayesian filtering algorithm to statistically characterize and improve signal-to-noise ratio of heart-rate data in wearable devices. *Journal of The Royal Society Interface* 21, 218 (09 2024), 20240222. doi:10.1098/rsif.2024.0222
- [3] Zachary Enghardt, Chengqian Ma, Margaret E. Morris, Chun-Cheng Chang, Xuhai Xu, Lianhui Qin, Daniel McDuff, Xin Liu, Shwetak Patel, and Vikram Iyer. 2024. From Classification to Clinical Insights: Towards Analyzing and Reasoning About Mobile and Behavioral Health Data With Large Language Models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 2 (May 2024), 1–25. doi:10.1145/3659604

- [4] Ty Ferguson, Timothy Olds, Rachel Curtis, Henry Blake, Alyson Crozier, Kylie Dankiw, Dorothea Dumuid, Daiki Kasai, Edward O'Connor, Rosa Virgara, and Carol Maher. 2022. Effectiveness of wearable activity trackers to increase physical activity and improve health: a systematic review of systematic reviews and meta-analyses. *The Lancet Digital Health* 4 (08 2022), e615–e626. doi:10.1016/S2589-7500(22)00111-X
- [5] Gemini Team. 2025. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL] <https://arxiv.org/abs/2312.11805>
- [6] GitHub. [n. d.]. GitHub Copilot. <https://github.com/features/copilot>
- [7] Google. 2025. Gemini API Multi-turn Conversations. <https://ai.google.dev/gemini-api/docs/text-generation#multi-turn-conversations>
- [8] Google. 2025. Google Gen AI SDK Documentation. <https://googleapis.github.io/python-genai/>
- [9] Google. 2025. Prompting with Contextual Information. <https://docs.cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/contextual-information>
- [10] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujie Yang, Nan Duan, and Weizhu Chen. 2023. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. arXiv:2305.11738 [cs.CL] <https://arxiv.org/abs/2305.11738>
- [11] A. Ali Heydari, Ken Gu, Vidya Srinivas, Hong Yu, Zhihan Zhang, Yuwei Zhang, Akshay Paruchuri, Qian He, Hamid Palangi, Nova Hammerquist, Ahmed A. Metwally, Brent Winslow, Yubin Kim, Kumar Ayush, Yuzhe Yang, Girish Narayanswamy, Maxwell A. Xu, Jake Garrison, Amy Armento Lee, Jenny Vafeiadou, Ben Graef, Isaac R. Galatzer-Levy, Erik Schenck, Andrew Barakat, Javier Perez, Jacqueline Shreibati, John Hernandez, Anthony Z. Faranesh, Javier L. Prieto, Connor Heneghan, Yun Liu, Jiening Zhan, Mark Malhotra, Shwetak Patel, Tim Althoff, Xin Liu, Daniel McDuff, and Xuhai Xu. 2025. The Anatomy of a Personal Health Agent. arXiv:2508.20148 [cs.AI] <https://arxiv.org/abs/2508.20148>
- [12] Jie Huang, Hanyin Shao, and Kevin Chen-Chuan Chang. 2022. Are Large Pre-Trained Language Models Leaking Your Personal Information?. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2038–2047. doi:10.18653/v1/2022.findings-emnlp.148
- [13] LangChain. [n. d.]. LangGraph: Agent Orchestration Framework for Reliable AI Agents. <https://www.langchain.com/langgraph>
- [14] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. arXiv:2305.01210 [cs.SE] <https://arxiv.org/abs/2305.01210>
- [15] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651 [cs.CL] <https://arxiv.org/abs/2303.17651>
- [16] Mike A. Merrill, Mingtian Tan, Vinayak Gupta, Tom Hartvigsen, and Tim Althoff. 2024. Language Models Still Struggle to Zero-shot Reason about Time Series. arXiv:2404.11757 [cs.CL] <https://arxiv.org/abs/2404.11757>
- [17] Modal. [n. d.]. Modal. <https://modal.com/>
- [18] G. B. Moody and R. G. Mark. 2001. The Impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine* 20, 3 (2001), 45–50. doi:10.1109/51.932724
- [19] Girish Narayanswamy, Xin Liu, Kumar Ayush, Yuzhe Yang, Xuhai Xu, Shun Liao, Jake Garrison, Shyam Tailor, Jake Sunshine, Yun Liu, Tim Althoff, Shrikanth Narayanan, Pushmeet Kohli, Jiening Zhan, Mark Malhotra, Shwetak Patel, Samy Abdel-Ghaffar, and Daniel McDuff. 2024. Scaling Wearable Foundation Models. arXiv:2410.13638 [cs.LG] <https://arxiv.org/abs/2410.13638>
- [20] OpenAI. [n. d.]. OpenAI Codex. <https://openai.com/codex/>
- [21] Xiaomin Ouyang, Zhiyuan Xie, Heming Fu, Sitong Cheng, Li Pan, Niwen Ling, Guoliang Xing, Jiayu Zhou, and Jianwei Huang. 2023. Harmony: Heterogeneous Multi-Modal Federated Learning through Disentangled Model Training. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 530–543. doi:10.1145/3581791.3596844
- [22] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560 [cs.AI] <https://arxiv.org/abs/2310.08560>
- [23] Attila Reiss. 2012. PAMAP2 Physical Activity Monitoring. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5NW2H>
- [24] Mickaël Ringeval, Gerit Wagner, James Denford, Guy Pare, and Spyros Kitsiou. 2020. Fitbit-Based Interventions for Healthy Lifestyle Outcomes: Systematic Review and Meta-Analysis. *Journal of Medical Internet Research* 22 (10 2020), e23954. doi:10.2196/23954
- [25] Andrew Roman, Harvard-MIT Division of Health Sciences, Technology, Roger G. Mark, and George B. Moody. 2025. MIT-BIH Arrhythmia Database (Simple CSVs). doi:10.34740/KAGGLE/DSV/12526755
- [26] Neeraj Sharma, Prashant Krishnan, Rohit Kumar, Shreyas Ramoji, Srikanth Raj Chetupalli, Nirmala R., Prasanta Kumar Ghosh, and Sriram Ganapathy. 2020. Coswara: A Database of Breathing, Cough, and Voice Sounds for COVID-19 Diagnosis. In *Interspeech 2020*. ISCA, 4811–4815. doi:10.21437/Interspeech.2020-2768
- [27] Leming Shen, Qiang Yang, Xinyu Huang, Zijing Ma, and Yuanqing Zheng. 2025. Gpiot: Tailoring small language models for iot program synthesis and development. In *Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems*. 199–212.
- [28] Leming Shen, Qiang Yang, Xinyu Huang, Zijing Ma, Yuanqing Zheng, and Chris Xiaoxuan Lu. 2026. Automating Model Deployment on Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services Workshops*. 188–194.
- [29] Leming Shen, Qiang Yang, Yuanqing Zheng, and Mo Li. 2025. Autoiot: Llm-driven automated natural language programming for iot applications. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking*. 468–482.
- [30] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] <https://arxiv.org/abs/2303.11366>
- [31] Shichao Sun, Ruifeng Yuan, Ziqiang Cao, Wenjie Li, and Pengfei Liu. 2024. Prompt Chaining or Stepwise Prompt? Refinement in Text Summarization. arXiv:2406.00507 [cs.CL] <https://arxiv.org/abs/2406.00507>
- [32] Yuwei Zhang, Kumar Ayush, Siyuan Qiao, A. Ali Heydari, Girish Narayanswamy, Maxwell A. Xu, Ahmed A. Metwally, Shawn Xu, Jake Garrison, Xuhai Xu, Tim Althoff, Yun Liu, Pushmeet Kohli, Jiening Zhan, Mark Malhotra, Shwetak Patel, Cecilia Mascolo, Xin Liu, Daniel McDuff, and Yuzhe Yang. 2025. SensorLM: Learning the Language of Wearable Sensors. arXiv:2506.09108 [cs.LG] <https://arxiv.org/abs/2506.09108>
- [33] Yuwei Zhang, Tong Xia, Aaqib Saeed, and Cecilia Mascolo. 2024. Respllm: Unifying audio and text with multimodal llms for generalized respiratory health prediction. *arXiv preprint arXiv:2410.05361* (2024).