

LMM Labs: An Early-Stage Case Study in Human-Orchestrated AI Acceleration

Lee Farretta and Matthew Farretta

LMM Labs | October 4, 2026

Prepared in collaboration with ChatGPT and Codex

Abstract

This case study examines how a two-person LMM Labs team used human orchestration, generative AI, and Codex agents to develop and refine an application portfolio. Over roughly a month of recent work, from September through October 4, 2026, the portfolio reached fourteen applications, twelve with cloud-deployment evidence and two working locally. This work built on applications and shared infrastructure established in August. The central achievement was a repeatable cycle: people defined the problem, agents implemented and tested, people reviewed and corrected, and an approved pipeline delivered releases to Azure. Revisions were part of developing both the applications and the method. Repository and deployment records support the practical value of this approach for a small team. Published studies provide context but cannot establish an LMM-specific percentage time saving. The case offers a promising starting baseline for a planned 4GL-style accelerator that will automate more of the cycle while preserving human direction and release approval.

Keywords: agentic AI; human orchestration; software prototyping; Codex; CI/CD; Azure; fourth-generation tools



Figure 1. Human orchestration from ideas to cloud applications. Note. Original AI-generated conceptual artwork; fictional collaborators are not portraits of the authors. It depicts a process, not measured findings.

A two-person effort across fourteen applications

The achievement and the question

LMM Labs is the collaborative effort of Lee Farretta and Matthew Farretta. Its mission connects human creativity and judgment with AI-assisted research and useful applications (LMM Labs, n.d.-a). Over roughly a month of recent development, from September through October 4, 2026, the team expanded and refined a portfolio that reached fourteen applications. Twelve had cloud-deployment evidence and two had local implementations. The central question is how human direction, chatbots, and coding agents helped this small team turn requirements into working software and deliver it through Azure.

The result is broader than a count of new apps. The work included new implementations, improvements to existing applications, tests, shared interface components, deployment automation, and a local UI Sandbox. These activities formed a cycle of building, reviewing, correcting, and releasing. Each cycle improved the applications and helped the team learn which steps could be standardized. LMM Labs' published method combines requirements analysis, prototyping, agile revision, and DevOps (LMM Labs, n.d.-b). The intended accelerator grows from this practical experience.

The starting point must be clear. Eight applications and shared cloud infrastructure were already established by late August. September through October 4 added six application implementations and further work on the suite. The fourteen-app total is the portfolio at the end of the recent period; it is not fourteen fresh builds begun on the same date. For date-based reporting, September 1–October 4 spans 33 elapsed days. Table 1 shows the earlier foundation and later additions together. The Dashboard, infrastructure, and UI Sandbox are counted separately from the applications.

How the evidence was checked

The analysis uses repository checkpoint 123df42, dated October 4. Commits, dated specifications and histories, retained human requests and agent responses, successful GitHub workflows, and read-only Azure records provide the evidence (LMM Labs, 2026). The broader records extend into August, including work on an existing Stock notebook before the first Git commit. Those earlier inputs belong to the project's history. Their presence prevents treating the recent month as the lifetime of the whole effort.

The evidence ledger records each application's purpose, scope, milestones, and maturity. It separates a reported runnable version, a tested local checkpoint, and a verified cloud release. It also distinguishes calendar time, deployment-job time, and human effort. Human hours were not recorded consistently. External studies were selected for original empirical methods and a clear metric; favorable and unfavorable results were included. The article focuses on the portfolio and working method, with detailed timestamps and calculations kept in the ledger.

A portfolio built through iteration

Breadth and shared foundations

The portfolio crosses several domains: financial analysis, research, digital art, tax scenarios, document processing, opportunity screening, food, encrypted communication, music, golf, wine, elections, household budgeting, and travel. That variety is a meaningful outcome for a two-person effort. The August foundation included Stock Analyzer, Research Studio, Digital Art Studio, Tax Scenario Lab, Document Scraper, Asymmetric Opportunity Screener, Palate, and Cipher. Shared components and a familiar stack made subsequent development easier. Individual apps varied in complexity and maturity, so their count describes breadth rather than equal units of effort.

September through October 4

Songsmith, Playable, and Oeno added music, golf instruction, and wine discovery in early September. Election Predictor, Budget, and Travel followed in the later records. Songsmith, Playable, Oeno, and Election have cloud evidence; Budget and Travel remain local in the reviewed snapshot. The resulting count is straightforward: eight earlier applications plus six additions equals fourteen catalog applications; eight earlier cloud prototypes plus four later ones equals twelve with cloud evidence. Table 1 sets out this sequence. The ledger provides each application's scope, dates, and evidence limits.

The team also revised existing behavior. Document Scraper gained clearer save and reopen actions, and Oeno's price information was updated. Election work separated forecast issuance from ordinary startup and introduced durable persistence. Budget implemented household imports, reconciliation, cash flow, targets, and reporting; Travel combined source acquisition with normalized data and retrieval. The local UI Sandbox added reusable presentation families and layouts. This is the cyclical approach in practice: working software exposes the next requirement, and the next implementation improves both the product and the reusable method.

Feedback turns prototypes into better applications

Retained sessions show that agents returned runnable versions quickly after receiving supplied requirements. Those versions gave the people something concrete to inspect and change. Songsmith required revisions to instrument behavior and documents. Earlier Tax work expanded into source-record, persistence, and account workflows. Playable moved from a reported local build to refinements and a successful release roughly a day after the request. The useful pattern is rapid feedback followed by correction. The case evaluates that pattern, rather than expecting the first generated version to be perfect.

The October 4 checkpoint report records 574 passing tests across the applications and shared contracts included in that review, together with relevant lint and builds. This supports the reviewed local code. The ledger identifies cloud releases separately and records maturity limits for each app. The evidence is appropriate to an evolving prototype portfolio, with quality checks strengthened as the method develops.

Human orchestration and iterative delivery

Direction, execution, and correction

The working sequence was ideation, specification, implementation, testing, human review, Azure deployment, and refinement (Figure 2). People chose the problems, priorities, intended behavior, and release conditions. ChatGPT supported discussion, analysis, and drafting. Codex inspected the repository, edited files, executed commands, ran tests, diagnosed failures, and prepared artifacts. This agentic capability connected discussion to actions in the actual development environment. The people then assessed the result and directed the next cycle.

Human review shaped the software. Requests to correct Songsmith's behavior, expand Tax's account and data handling, make Document saves explicit, and preserve Election forecasts established the behavior that the implementation had to satisfy. Agents translated those decisions into code, tests, and documentation. Test failures led to repairs; review revealed requirements that needed clarification. Corrections are evidence of the process doing its work. They also reveal opportunities to improve reusable contracts and checks so later applications avoid repeating the same mistakes.

Reuse reduced repeated setup

The team reused interface components, application catalog conventions, React/Vite frontends, Python services, containers, and deployment scripts. The August cloud foundation made it easier to add later applications without creating a new platform each time. Agents worked within those existing conventions. This combination plausibly reduced repeated setup and coordination work. The records cannot separate the benefit of the agents from the benefit of reuse, developer experience, and controlled prototype scope. AI features inside an application are also different from AI assistance used to build it.

Automation after release approval

GitHub Actions validates the requested components, authenticates to Azure, packages the selected application, deploys it, and checks health. September changes established manual-only application deployment. The current workflow requires an explicit dispatch, and repository policy requires human release authorization. The agent can prepare the release and its evidence, but a checkpoint push is not permission to deploy. This approval boundary is part of the working method and should remain in the future accelerator.

Across August and September, four recovered successful deployment jobs took between 2 minutes 9 seconds and 9 minutes 41 seconds. Those measurements demonstrate that a prepared release can run quickly through the pipeline. They exclude requirements work, coding, review, waiting outside the job, and failed attempts. Election's October 2 records include failed frontend deployments before a successful deployment and health check. The fair conclusion is that automation shortened the repeated release operation while development and troubleshooting still required work. Pipeline minutes are not an application's total development time.

Comparisons with published research

Coding assistance: useful, but variable

Peng et al. (2023) randomly assigned professional programmers to a JavaScript HTTP-server task with or without Copilot. Among completers, the assisted group averaged 71.17 minutes and the control group 160.89 minutes; the reported reduction was 55.8%. This is evidence that AI assistance can speed up a specific coding task. It does not establish the time needed to specify, test, review, and deploy an entire application. LMM's applications are not equivalent to that task, so no application-wide multiplier is calculated from it.

Cui et al. (2026) studied 4,867 developers across three companies and estimated 26.08% more completed tasks with assistant access. That is a measure of output, not a reduction in end-to-end delivery time. Becker et al. (2025), by contrast, found that 16 experienced open-source developers took 19% longer across 246 tasks when early-2025 AI tools were available. Their later update explains why selection and concurrent agent use make newer timing estimates difficult to interpret (Becker et al., 2026). The studies show that benefits depend on the work, the developer, and the tools.

Agents and delivery automation measure different things

Yang et al. (2024) evaluated an agent that navigates files, edits code, and runs tests. Its 12.5% first-attempt resolution rate on SWE-bench concerns software-issue capability. It is neither a measurement of current Codex nor a benchmark for building a cloud app. Its relevance here is the use of execution and test feedback, operations visible in the LMM sessions. Table 2 identifies the population and metric for each source so that unlike results are not presented as equivalent.

Forsgren et al. (2019) found that high-performing delivery organizations could move committed changes to production rapidly: less than a day for the elite group, and one day to one week for the high group. LMM's successful workflow jobs show a fast automated release process. However, job duration omits development and review, and one prototype release is not comparable to a team's sustained production delivery record. The article therefore makes no numerical acceleration ratio against those survey categories. Conventional tooling and CI/CD already account for part of the delivery benefit.

Quality is part of the comparison

Gordon and Bieman (1993) found requirements and usability benefits in reported industrial prototyping cases, together with design and maintainability risks. DeBellis et al. (2025) associated greater AI adoption with lower delivery throughput and stability in survey data. These findings do not negate LMM's results; they explain why quick builds must be followed by testing and review. Overall, the external evidence is consistent with the team's experience of rapid implementation and continuing correction. It does not provide a sufficiently matched traditional-development portfolio to quantify the total time saved at LMM.

From working method to automated accelerator

The supported finding

The big picture is a two-person team using human-orchestrated AI to expand and refine a fourteen-application portfolio during roughly a month of recent work. Alongside the apps, the team developed shared components, tested behavior, a release process, and a local Sandbox. The value is the capacity to work across this scope through repeated delivery cycles, while progressively making the method reusable. The portfolio provides concrete evidence of what the combined approach enabled the team to deliver.

The approach combines complementary contributions. People supply domain knowledge, requirements, priorities, corrections, and release decisions. Agents connect those decisions to repository changes and verification. Shared components reduce repeated setup, and CI/CD handles repeatable release operations. The records show these parts working together. They support a practical conclusion: a small team can use this combined method to carry a varied portfolio forward quickly. The available evidence cannot assign a percentage of total time saved to any one contribution.

Limits that affect the conclusion

This is an early-stage insider case study. Human effort records are incomplete, some first requirements and release records are missing, and applications differ in scope and quality needs. Earlier notebooks and research supplied inputs. Cloud prototypes have not all received independent security or correctness audits. These limits constrain a numerical comparison with a hypothetical traditional project, but do not erase the delivered software and successful releases. Transfer is most plausible for similar prototypes on a familiar stack; more demanding reliability or regulatory requirements require their own validation.

The next step: a planned accelerator

The planned 4GL-style accelerator is the next step in the same cycle. The team is identifying repeatable requirements, component contracts, scaffolds, tests, and release preparation so these can be automated together. The first local module, UI Sandbox, already supports presentation families and layouts, configuration import/export, preserved UI state, and a reusable Dashboard adapter. It uses fixtures. Complete application generation and broader automation remain planned capabilities. Human direction, review, and release approval remain part of the intended design.

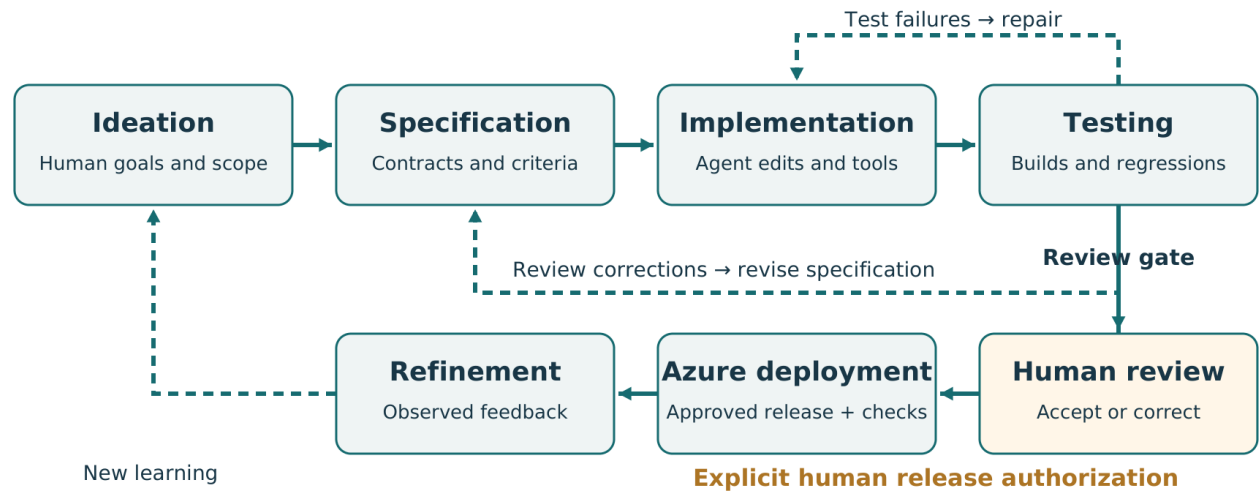
The next evaluation should measure complete cycles: accepted requirements, a usable prototype, completed human review, release approval, and verified cloud success. Record active human time separately from agent execution, waiting, and parallel work, together with reuse, scope changes, defects, rework, costs, and tool versions. Compare similar increments using conventional tools, chatbot assistance, and agents. The resulting measures will show whether the accelerator improves on this first working baseline in speed, quality, and effort.

Conclusion

LMM Labs' early findings are promising because they show a working approach across a broad portfolio, not just an isolated coding exercise. Human orchestration, agentic execution, reuse, and automated delivery made a small team's work concrete and reviewable. Iteration improved the applications and exposed the building blocks of a future accelerator. The next objective is to automate more of that working cycle while preserving human judgment and making its gains easier to measure.

Lifecycle and verified milestones

Human direction and technical judgment throughout



Reusable foundation: shell • contracts • containers • delivery scripts

Solid arrows: delivery sequence. Dashed arrows: feedback. No stage is timed by this figure.

Figure 2. Observed iterative delivery lifecycle and explicit release gate. Note. Original synthesis of repository sessions, workflow configuration and deployment policy (LMM Labs, 2026). Loops illustrate correction paths, not measured frequencies.

Portfolio milestones

Table 1
Verified portfolio milestones

Period	Milestone	Interpretation	Sources
Aug 1-3	Stock + Research	Requests precede first Git checkpoint; Stock reuses notebook.	Sessions; ce03112
Aug 7-11	Pipeline + Art	Cloud foundation; Stock/Research Aug 8; Art Aug 11 successful jobs.	31282383452; 31568558812
Aug 13-26	Tax, Document, Asymmetric, Palate, Cipher	Local builds, expanded Tax scope; eight original apps cloud-validated by Aug 26.	c79c41f; a3b379b; fb8cbcd; deployment report
Sep 2-4	Songsmith	First version, human-directed correction, then commissioning.	Session; a77ff6c; Azure creation
Sep 7-8	Playable + Oeno	Eleven original apps commissioned; Playable release succeeds.	db628f5; 34200690450; Azure creation
Sep 25-Oct 4	Travel, Election, Budget, Sandbox	Fourteen catalog apps; twelve have cloud evidence; Sandbox remains local.	96b6f68; 36999244868; 53fd417; 123df42

Note. All dates are 2026 in America/Los_Angeles. See the ledger for per-app scope, host-versus-release distinctions and evidence limitations.

External benchmark comparison

Table 2
Benchmarks and corresponding LMM observations

Comparison	Population/source	Metric/result	Relevance/limits	LMM observation
Human-led prototyping	Gordon & Bieman (1993): 34 reported cases	Requirements/usability benefits; design and maintenance risks.	Qualitative; heterogeneous scope and reporting bias.	Rapid prototypes expose corrections; not production certification.
Human-only vs assistant coding	Peng et al. (2023): 95 recruited professionals	160.89 vs 71.17 min among completers; 55.8% less time.	Bounded HTTP server, not cloud app delivery.	Runnable builds followed by review and correction.
Conventional CI/CD	Forsgren et al. (2019): almost 1,000 survey respondents	Elite <1 day; high 1 day –1 week commit-to-production.	Category ranges, self-report; established production services.	Reusable release pipeline; successful jobs took minutes.
AI field experiments	Cui et al. (2026): 4,867 developers, three firms	26.08% more completed tasks; SE 10.3 percentage points.	Task output; completion assistance; variable experience.	Fourteen-app portfolio and revisions; no comparable task count.
Experienced maintenance	Becker et al. (2025): 16 developers, 246 tasks	19% longer with early-2025 AI.	Mature familiar repositories; not greenfield prototypes.	Corrections and integration remain substantial.
Updated agent use	Becker et al. (2026): 57 developers, 800+ tasks	Later signal unreliable because of selection/time measurement.	Concurrent agents complicate task timing.	Separate active human time from agent waiting/overlap.
Agent capability/testing	Yang et al. (2024): SWE-bench issue evaluation	12.5% first-attempt resolution; file/tool/test feedback.	Capability, not delivery speed or a current Codex score.	Observed repository editing, testing and repair loop.
AI and delivery quality	DeBellis et al. (2025): DORA survey analysis	AI adoption associated with lower throughput/stability.	Observational, 2024 data; not a randomized causal estimate.	Keep review, bounded releases and regression checks.

Note. Distinct populations, units and completion stages prevent a portfolio-wide causal multiplier. Detailed source records and comparison limits are in the ledger.

Acknowledgment and author note

ChatGPT and Codex assisted with inspection, research, calculations, drafting, artwork, formatting and artifact checks. AI is not an author. The human byline follows LMM Labs' established article convention. Both named authors are case participants; this is an insider evaluation rather than an independent audit.

References

- Becker, J., Rush, N., Barnes, B., & Rein, D. (2025). *Measuring the impact of early-2025 AI on experienced open-source developer productivity*. arXiv. <https://doi.org/10.48550/arXiv.2507.09089>
- Becker, J., Rush, N., Cunningham, T., Rein, D., & Mahamud, K. (2026, February 24). *We are changing our developer productivity experiment design*. METR. <https://metr.org/blog/2026-02-24-uplift-update/>
- Cui, K. Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2026). The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers. *Management Science*. Advance online publication. <https://doi.org/10.1287/mnsc.2025.00535>
- DeBellis, D., Storer, K. M., Villalba, D., Harvey, N., D'Angelo, S., & Brown, A. (2025). *Impact of generative AI in software development* (Version 2025.2). DORA, Google. <https://dora.dev/ai/gen-ai-report/dora-impact-of-generative-ai-in-software-development.pdf>
- Forsgren, N., Humble, J., Smith, D., & Frazelle, J. (2019). *Accelerate state of DevOps 2019*. DORA, Google Cloud. <https://dora.dev/research/2019/dora-report/2019-dora-accelerate-state-of-devops-report.pdf>
- Gordon, V. S., & Bieman, J. M. (1993). Reported effects of rapid prototyping on industrial software quality. *Software Quality Journal*, 2(2), 93–108. <https://doi.org/10.1007/BF00590438>
- LMM Labs. (n.d.-a). *About us*. Retrieved October 4, 2026, from <https://lmmllabs.com/about-us/>
- LMM Labs. (n.d.-b). *Development approach—Methodology*. Retrieved October 4, 2026, from <https://lmmllabs.com/development-methodology/>
- LMM Labs. (2026). *Application delivery evidence, August–October 2026* [Unpublished internal records]. Repository checkpoint 123df42; accompanying case-study evidence ledger.
- Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). *The impact of AI on developer productivity: Evidence from GitHub Copilot*. arXiv. <https://doi.org/10.48550/arXiv.2302.06590>
- Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). *SWE-agent: Agent-computer interfaces enable automated software engineering*. arXiv. <https://doi.org/10.48550/arXiv.2405.15793>