Indie at Scale: How Large Language Models Make Community-Centered Game Development Economically Viable

Jusuf Qarkaxhija
Faculty of Computer Science, AAB College, Republic of Kosovo

***Abstract*—Large language models are now widely used in software engineering, yet most evidence of their practical effect comes from corporate productivity studies or synthetic benchmarks. This study offers a complementary view from the solo Indie developer's perspective: a reflective case study of two iOS games developed in continuous pair programming with a large-language-model coding assistant and subsequently released on the App Store. Both games target communities that mainstream consumer software has long underserved. The Listening Maze is a spatial-audio echolocation maze for blind and low-vision players. Trace Memory Game is a memory-recall game built around brief-flash visual encoding and freehand redraw, positioned for daily cognitive engagement rather than cognitive training or clinical intervention. We describe the development methodology, the four-step human-LLM loop used throughout, the resulting codebases, an indicative break-even analysis, and what the experience implies for socially oriented software development by very small teams. We propose the crossover thesis as a design heuristic and label every claim in the discussion as observation, interpretation, or hypothesis. The evidence is an existence proof from a single developer on a single platform, not a population estimate.**

***Keywords*—Large language models; AI-assisted software development; mobile accessibility; spatial audio; cognitive engagement; human-AI collaboration; Indie games**

I. Introduction

In the United States, roughly one million adults live with blindness. Globally, more than fifty-five million people live with dementia, a figure projected to nearly triple by mid-century [1]. Both populations are described in the accessibility and digital-health literature as underserved by mainstream consumer software. A survey of the App Store for polished, contemporary games addressed to these audiences returns a sparse inventory: a small number of audio-only titles from specialist studios, a long tail of exercise-style brain trainers whose evidence base is weak [2], and little between the two.

The reason is, in the first instance, economic. The App Store exhibits a steep power-law revenue distribution: empirical work by Garg and Telang [3] shows that the rank-to-demand relationship in the iTunes App Store follows a Pareto curve, with the top-ranked paid app generating roughly 150 times the downloads of the app at rank 200. A practitioner study of accessibility in software development further reports the trade-off between benefits and costs as a recurring concern, and that retrofitting accessibility into a system designed without it tends to require costly design rework that earlier integration would have avoided [4]. For a solo developer aiming squarely at an underserved community such as blind players, memory-impaired adults, color-blind users, or the tremor-affected, the arithmetic has historically not worked: small audience, high cost of quality, no publisher to absorb the risk.

Large language models (LLMs) are beginning to change that arithmetic. A controlled experiment reports productivity gains of roughly 56 % on an isolated coding task [5], and a multi-company field experiment aggregating across 4,867 developers reports a 26 % increase in completed tasks, with the largest gains concentrated among less-experienced developers [6]. A large-scale survey study similarly finds perceived productivity strongly associated with suggestion-acceptance rate [7]. LLM-assisted development is not a panacea. A meta-analysis of human-AI collaboration shows that combinations sometimes underperform either party in isolation [8], and benchmarks of end-to-end issue resolution show that agents remain well short of human rates on real repositories [9].

This study documents what that shift looks like in one practitioner's practice. We report on two iOS games developed by a single author in continuous pair-programming collaboration with an LLM-based coding assistant over roughly six months, both of which have since been approved by App Review and are publicly downloadable. The Listening Maze is a spatial-audio maze game: the player navigates pitch-black mazes using head-related-transfer-function (HRTF) rendered echoes, with full VoiceOver compatibility and toggleable haptics. Trace Memory Game is a memory-recall game: a shape is flashed for a sub-second interval and the player must redraw it from memory at the same position and scale. Both were built with no external dependencies and no outsourced asset work.

A. Terminology of Product Status

Because the distinction matters for how much weight the reader should place on our claims, we fix the following vocabulary and use it consistently throughout:

- *Feature-complete* means every planned feature is implemented and the app builds without known blocking defects.
- *Archived* means a signed, distributable build has been produced in Xcode.

*Corresponding author.

- *Submitted* means the archive has been uploaded to App Store Connect and entered App Review.
- *Released* means App Review has approved the build and it is publicly downloadable.

Both apps are *released* in this sense. Release attests to technical completion and platform approval only; it attests to nothing about user adoption, retention, revenue, or commercial success. No download, retention, rating, or revenue figures are reported in this study, and no argument advanced here depends on them.

B. The Crossover Thesis

A central design commitment runs through both products and frames our contribution. We call it the *crossover thesis*: the disability or condition that motivates a game should not be a sidecar feature; it should shape the core mechanic, and that mechanic should be interesting on its own to a player who has never heard of the target community. In The Listening Maze, echolocation is not an accessibility add-on. It is the entire puzzle. In Trace Memory Game, brief-flash memory recall is not exercise-flavored therapy. It is a game whose elegance is meant to be visible to anyone. The crossover thesis is both an ethical commitment, in that the community gets a real product rather than a token, and a sustainability commitment, in that the mainstream audience underwrites the niche. We state it formally in Section VI, and we are explicit there that two case studies cannot establish it as a necessary or sufficient condition for anything.

C. Contributions

This study makes the following contributions:

- A description of the human-LLM pair-programming methodology used to develop two iOS applications, including the role of a persistent project-memory artifact and explicit audit cycles, together with a statement of what the method did *not* instrument.
- Two end-to-end case studies The Listening Maze and Trace Memory Game, each with its architecture, scoring model, accessibility design, and defect-severity criteria.
- A statement of the crossover thesis as a portable design heuristic for community-centered game development, with the two case studies as illustrations rather than as proof.
- A parametric break-even analysis that makes the economic argument checkable, and a reflective evaluation of LLM-assisted development in which every claim is labeled as observation, interpretation, or hypothesis.

D. Scope and Limitations

We are not running a controlled experiment that compares LLM-assisted to non-LLM development. The contribution is descriptive and reflective, in the tradition of HCI case-study research [10], [11]. We do not make clinical claims for either game. Trace Memory Game is designed for cognitive engagement, not cognitive training and not clinical intervention; Section V-A defines those three constructs and states which of them the application addresses. We report on two released products developed by a single author, and we treat the resulting account as an existence proof rather than a population estimate. Section II situates this work among the LLM-coding, accessibility, cognitive-health, co-creation, and Indie-economics literatures. Section III describes the human-LLM development loop. Sections IV and V present the two case studies. Section VI states the crossover thesis. Section VII discusses implications, the epistemic status of each claim, threats to validity, and ethics. Section VIII concludes.

II. Related Work

This study sits at the intersection of five strands of work: empirical evidence on LLM-assisted software development, audio-first accessibility games, cognitive-health games, human-AI co-creation, and the economics of Indie mobile development. We review each in turn, with attention to what is not yet documented: longitudinal evidence from a single developer shipping community-targeted commercial products with an LLM collaborator in the loop.

A. LLMs for Software Engineering

Empirical evidence that LLM-based coding assistants change developer productivity has accumulated rapidly. In a controlled experiment with 95 professional developers implementing an HTTP server in JavaScript, the group with access to GitHub Copilot completed the task 55.8 % faster than the control group [5]. Larger-scale field evidence yields a more conservative estimate. A three-firm field experiment at Microsoft, Accenture, and a Fortune 100 electronics manufacturer, covering 4,867 developers, reports a 26.08 % increase in completed tasks, with the strongest effects accruing to short-tenured and junior developers [6]. A survey of 2,047 matched Copilot users found that perceived productivity tracked suggestion-acceptance rate more closely than it tracked any other usage measure [7].

Time-on-task is not the only relevant outcome. A within-subjects CHI study found that participants preferred Copilot even though it did not reliably improve task success, citing benefits in "getting started" that were partially offset by difficulties in understanding and debugging the generated code [12].

Beyond productivity studies, the SWE-bench benchmark exposed a substantial gap between snippet completion and end-to-end issue resolution. In the original release, the best contemporaneous model resolved only 1.96 % of real GitHub issues drawn from twelve mature Python repositories [9]. After the benchmark was hardened into the 500-task SWE-bench Verified set, the strongest contemporaneous model at launch resolved roughly one third of tasks [13]; the frontier has moved upward since, but real-codebase software work still leaves headroom on harder subsets.

What is missing in this literature is longitudinal evidence. Most existing studies are either short controlled tasks (hours), short field experiments (weeks), or static-snapshot benchmarks. Little has been published on what an entire single-developer product cycle looks like when an LLM is the constant pair

programmer from architectural sketch through bug audits to App Store release. Our case studies are an attempt to fill that gap from the Indie end of the cost curve.

B. Accessible and Audio-First Games

Audio-first computing for blind and low-vision users has a long research lineage. The *AudioDoom* work of Lumbreras and Sánchez [14] introduced 3D-aural interactive hyperstories for blind children and showed that spatial mental representations can be constructed from spatialized sound alone, without vision. Subsequent work extended these ideas to navigation training: audio-only games such as the Audio-based Environment Simulator were shown to support the transfer of spatial knowledge from virtual exploration to real-world wayfinding in blind children [15]. The use of HRTFs to render distance and azimuth cues is now standard in this design space.

Two observations motivate our first case study. First, almost all of the cited work lives in research prototypes, museum installations, or specialized desktop software, not in App-Store-quality consumer products distributed under a normal commercial business model. Second, where commercial audio games do exist (for example, *Papa Sangre* and *A Blind Legend*), the developer team typically counts dozens of people, not one. Our work documents a polished, released spatial-audio maze game built by a single developer with an LLM coding assistant in the loop.

C. Games for Cognitive Health

The 2024 update of the Lancet standing Commission on dementia identified fourteen modifiable risk factors and estimated that around 45 % of dementia cases worldwide could in principle be delayed or prevented through interventions that act on those factors across the life course [1]. This sits alongside the well-known critique by Simons et al. [2], who reviewed the brain-training literature and concluded that there is extensive evidence of improvement on the trained task, weaker evidence of near transfer, and very little evidence of far transfer to everyday cognitive performance. A pilot randomized study of tablet-based puzzle games in healthy adults and older people reported feasibility for cognitive training, alongside small gains in visual-attention and visuospatial measures, while remaining cautious about broader clinical claims [16].

We cite this literature to explain why the design space is crowded with weakly evidenced products, not to position our own product within it as an intervention. Section V-A draws that boundary explicitly.

D. Human-AI Collaboration in Creative Work

A growing body of HCI research treats human-AI interaction as co-creation rather than as dictation. Wan et al. [10] described prewriting with LLMs as a three-stage iterative process of ideation, illumination, and implementation. The meta-analysis of Vaccaro et al. [8] synthesizes 106 experimental studies on human-AI collaboration and finds that, on average, combinations underperform the best of human-alone or AI-alone. Combinations help in content-creation tasks; in decision-making tasks they often hurt.

Two implications of this literature are directly relevant. First, the development of a released iOS application is dominantly a content-creation task, which is the regime in which combinations have been shown to help. Second, the documented benefit is contingent on humans staying in the driving seat. Our method section formalizes that constraint: the LLM proposes; the developer audits, accepts, rejects, and persists decisions.

E. Indie Software Economics and Digital Inclusion

Two structural facts shape the economics of community-targeted mobile development. First, the App Store and equivalent storefronts exhibit a steep power-law revenue distribution. Using a rank-to-demand inference method on public ranking data, Garg and Telang [3] estimate that the top-ranked paid iPhone app generates roughly 150 times the downloads of the app at rank 200, with downloads falling off according to a Pareto distribution across the catalogue. For a developer aiming at a niche underserved community, the long-tail position implied by this curve makes the expected commercial return on the community-only audience, in isolation, rarely sufficient to fund the development effort.

Second, a mixed-methods study of accessibility in software practice, drawing on 15 practitioner interviews and 365 survey responses from 26 countries, groups 44 practitioner statements into eight recurring themes covering both the benefits and the costs of incorporating accessibility, and reports that the cleanest fixes for many late-stage accessibility issues require revisiting the overall design, with significant impact on timelines and processes [4]. Together, these facts imply a tight squeeze: small audience, high cost of quality, no publisher to absorb the risk.

Recent HCI work has begun to document how generative AI changes this calculus. Panchanadikar and Freeman [11] analyzed 3,091 online posts and comments from Indie-dev subreddits and Facebook groups and reported that Indie developers see generative AI in tension, as both supporting and threatening their creative practice; one participant gave the study its title by calling generative AI "my new ill-informed co-worker." A preprint analysis of Steam catalogue data and AI-disclosure statements reports that relatively more independent developers entered the market once generative AI became publicly accessible [17]; this result is treated here as suggestive rather than established. To our knowledge, no published work yet documents, end-to-end, how this changed calculus plays out for a solo developer specifically targeting underserved accessibility and health-adjacent communities. This study is that account.

III. Method: A Human-LLM Pair-Programming Loop

A. Setting and Tooling

The working environment is intentionally minimal. All development took place on macOS using Xcode and SwiftUI, targeting iOS 17 and above. No external dependencies were introduced beyond first-party Apple frameworks (`AVFoundation`, `CoreHaptics`, `StoreKit`, `SwiftUI`, `UIKit`). Source control was Git with a single `main` branch.

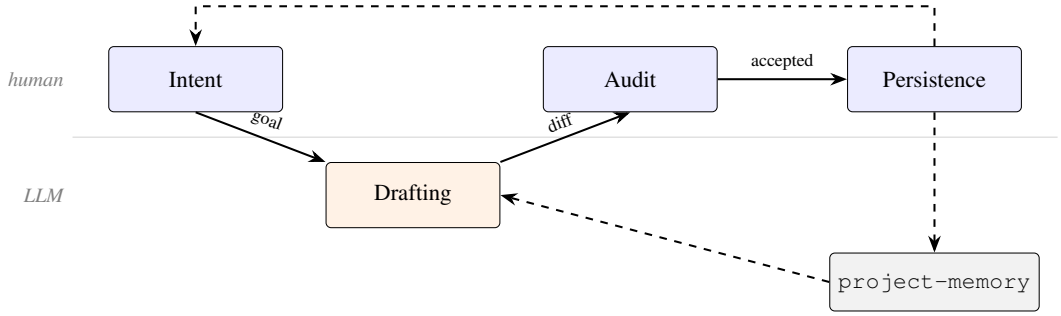


Fig. 1. The four-step pair-programming loop. The human states intent; the LLM drafts an implementation; the human audits the diff; accepted decisions persist to a project-memory file that informs the LLM's drafting on the next pass. Dashed arrows are the two return paths: the short cycle (persistence to intent) and the long cycle (project memory to drafting).

The LLM collaborator was a commercial code-aware assistant accessed through a terminal client running directly in the working directory. Two properties of this setup matter for the methodology. First, the assistant could read, write, and execute against the live project tree; it was not confined to a chat window detached from the code. Second, the assistant was instructed, on every session, to load and respect a persistent project-memory file (`project-memory.md`) committed alongside the source. We return to this artifact in Section III-C.

B. The Loop

The development cycle settled, across both products, into the repeatable four-step loop shown in Fig. 1:

- Intent. The human author states a concrete goal in natural language (for example, "add a daily-challenge mode seeded by date so that all players see the same maze on the same calendar day").
- Drafting. The LLM produces a candidate implementation with rationale, taking the persistent project-memory file as durable context. The output is typically 10 to 100 lines of Swift code, a one-paragraph explanation, and an explicit list of files touched.
- Audit. The author reviews the diff and either accepts, rejects with a reason, or asks for an explicit second pass. A subset of audits are formal *rigorous audits*, in which the LLM is asked to review its own prior output against an enumerated checklist (bounds checks, `EnvironmentObject` injection, haptic deduplication, and so on).
- Persistence. Accepted decisions are written back into the project-memory file so that they survive future sessions and future LLM contexts.

Fig. 1 renders two properties of the cycle that the enumeration above leaves implicit. The first is the distribution of control: the human occupies three of the four steps, and drafting is the only stage at which the assistant acts without immediate human participation. The second is that the cycle contains two distinct return paths rather than one. The short path returns from persistence to intent and initiates the next feature; the long path returns from the project-memory store to drafting and is the mechanism by which accepted decisions survive session boundaries. In the absence of the second path, the arrangement reduces to a conventional prompt-and-paste workflow.

The loop is consistent with the three-stage ideation $\rightarrow$ illumination $\rightarrow$ implementation pattern described by Wan et al. [10] for prewriting, with one addition: an explicit fourth *persistence* step that closes the loop across sessions. The audit step is where the human carries the task-specific competence that Vaccaro et al. [8] associate with human-AI combinations that beat either party alone, namely cases where the human is already the stronger solo performer on the task.

C. Project Memory as a First-Class Artifact

The single most consequential lever in our setup is the persistent project-memory file at the repository root. It is not auto-generated documentation. It is a curated context file that records, in a flat Markdown structure, every architectural decision, bug fixed, business-model choice, and design constraint that has accumulated over the life of the project. It begins with a one-paragraph statement of *what this is*, then lists files, environment-object chains, scoring rules, edge cases handled, and known limitations.

The project-memory artifact serves three functions. First, it is the LLM's durable working memory: every session loads it on entry, so the assistant does not have to be re-briefed on the architecture. Second, it is a forcing function for the human, since writing into it requires explicit articulation of decisions that might otherwise remain tacit. Third, it is the lineage of defects already caught, which makes regressions detectable across rewrites. Across the two projects, the memory file grew to roughly 200 lines of structured prose per app and was updated on most sessions.

The project memory keeps decisions addressable long after the conversation that produced them has expired, and it is the mechanism that most plausibly accounts for the absence, in these projects, of the failure mode reported by Vaithilingam et al. [12], in which developers found LLM-generated code difficult to understand and debug. This is an interpretation rather than a finding: no condition without the memory file was run, so the absence cannot be attributed to it.

D. What this Method Did Not Measure?

The two projects were undertaken as commercial development rather than as a designed study, and the available

instrumentation reflects that origin. Three quantities that would strengthen the account are consequently unavailable, and are reported as absent rather than replaced by estimates.

1) Suggestion acceptance and rejection rates: No automated log of proposed against accepted diffs was maintained. An acceptance rate cannot be reported, and the present account therefore cannot be related to the acceptance-rate findings of Ziegler et al. [7].

2) Proportion of LLM-authored to human-authored code: Accepted drafts were routinely edited during audit prior to commit, and individual commits combine both sources; surviving line-level provenance was consequently not recorded. The only component for which an approximate figure can be offered is the shape catalogue of Trace Memory Game (Section V-E), where drafting proceeded shape by shape. That figure is a retrospective estimate rather than a value recovered from a log.

3) Time-on-task with and without assistance: No condition without LLM assistance was run. Every productivity statement in this study is therefore an interpretation informed by the literature reviewed in Section II-A rather than a measurement obtained from the two projects.

These gaps could be closed at negligible cost in a replication: it would suffice to log each proposed diff with an accept, reject, or revise verdict and a one-line justification, to tag the resulting commits, and to recover surviving provenance at release using `git blame` restricted to the tagged hunks. Their absence is the principal methodological limitation of the present account.

E. Quantitative Snapshot

Table I summarizes the two products at their release states. Both apps ship with no external dependencies, are written in pure SwiftUI, and target iOS 17 and above.

IV. Case Study I: The Listening Maze

A. Motivation and Target Community

Approximately one million US adults are estimated to live with blindness, and an order of magnitude more live with significant low vision. The dedicated community around audio-only games, centered on resources such as *AppleVis* and *AudioGames.net*, is small but active and has produced both notable commercial titles and a body of practitioner critique. Within this space, the typical product is either decades old, technically constrained, or developed by a specialist studio at a cost of quality that is inaccessible to a solo author.

The Listening Maze is an attempt to ship, as a single Indie developer, a polished spatial-audio maze game that meets modern App-Store-quality expectations while remaining playable without sight. The crossover thesis applies: the game is primarily designed for blind and low-vision players, but its mechanic of navigating a pitch-black maze using only sound is intended to be novel to sighted players who have never had to construct a mental map from audio cues alone.

B. Core Mechanic: Navigation by Echo

A maze is generated by a recursive-backtracker algorithm seeded either by level number (campaign mode), the calendar date (daily challenge), or run depth (endless mode). The player occupies a single cell. Tapping the center of the screen sends a "ping," a short procedurally generated impulse that bounces through the maze and returns to the player's virtual ears as spatialized echoes whose direction and timing encode the shape of the surrounding corridors. The player moves with four directional gestures.

The audio interface is the entire interface. The on-screen visuals exist to satisfy users who prefer to play with their eyes open, as many sighted players do, but they are strictly redundant with the audio: a player who closes their eyes loses no game-state information. This constraint is what aligns the design with the requirements of the target community, and it is also the source of the novelty of the mechanic for players outside that community. The line of research initiated by Lumbreras and Sánchez [14] and continued by Merabet et al. [15] provides the theoretical grounding: blind players can and do construct rich spatial cognitive maps from spatialized audio alone.

C. System Architecture

The application is implemented in SwiftUI with four `EnvironmentObjects` injected at the app root: `GameManager` (core game state, movement, ping, timer, and score), `SpatialAudioEngine` (audio synthesis and 3D playback), `ProgressManager` (level progress, settings, daily and endless persistence), and `StoreKitManager` (in-app purchase, restore, entitlement verification). All audio is procedurally generated. The application ships with zero audio files, which keeps the binary small and allows the ping and echo parameters to be tuned in code rather than re-recorded.

Fig. 2 relates these objects to the view layer. Its salient property is the sparsity of the dashed consumption edges: no view depends on all four objects, and the audio engine is consumed by only two. This separation permitted the ping-calibration work described below to proceed through repeated iteration without modification to persistence or purchase code, and it is the principal reason the system remained tractable for a single developer.

The spatial audio engine is built on `AVAudioEngine` with an `AVAudioEnvironmentNode` performing the HRTF rendering. The engine consumes a sequence of `(position, time, amplitude)` triples from the maze topology and schedules procedurally generated buffers to play at the correct 3D location relative to the listener's head. The technical foundations of this approach, HRTF-based binaural synthesis for virtual auditory spaces, are well established in the spatial-audio literature [18].

Three of the resulting screens are shown in Fig. 3. Fig. 3a presents the in-maze view, whose visual sparseness follows directly from the redundancy constraint, since every element drawn there is also available through the audio channel. Fig. 3b presents the composite score, which combines pings issued, moves, elapsed time, wall collisions, and collectibles, so that

TABLE I. CODEBASE SUMMARY FOR THE TWO CASE-STUDY APPS AT RELEASE

	The Listening Maze	Trace Memory Game
Platform	iOS 17+ (SwiftUI)	iOS 17+ (SwiftUI)
Swift files	14	13
External dependencies	0	0
Bundled media assets	0 (audio synthesized in code)	0 (shapes drawn in code)
Core mechanic	spatial-audio echolocation	flash and recall drawing
Content shipped	120 levels (6 difficulties)	141 shapes (8 categories)
Free / paid split	20 levels free, 100 paid ($1.99)	15 shapes free, 126 paid ($2.99)
Target communities	blind and low-vision; sighted puzzle players	cognitive engagement; memory-curious players
Critical defects caught in audit	12 (see Section IV-E)	audit log not classified to the same criteria

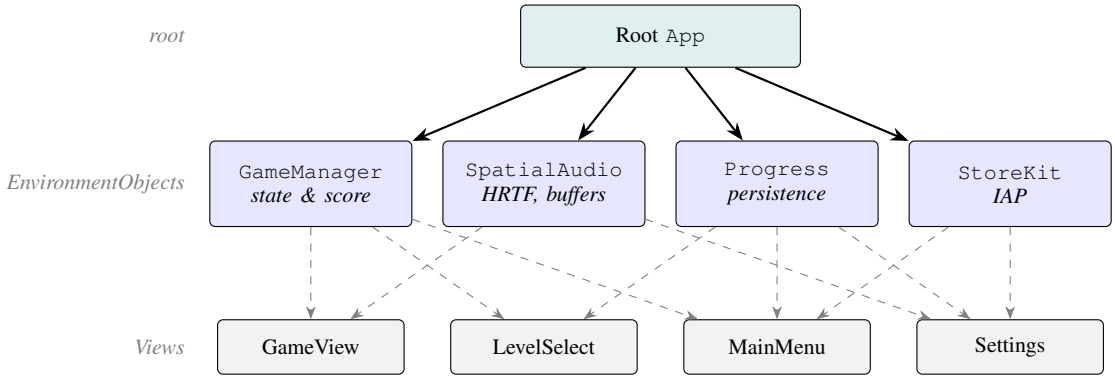
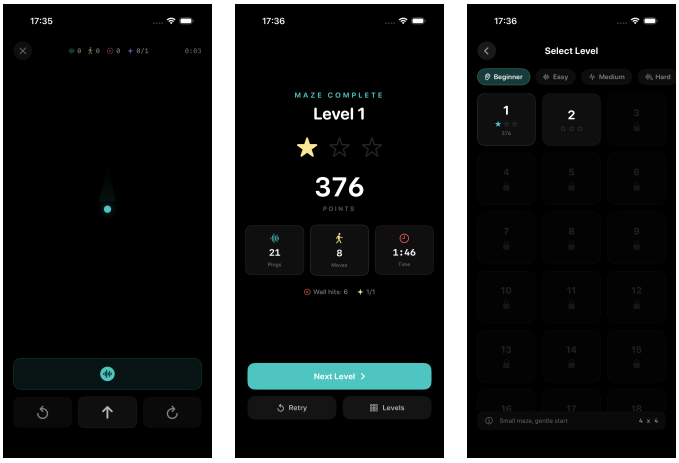


Fig. 2. Architecture of The Listening Maze. The app root injects four `EnvironmentObject` singletons; views consume the subset of objects they need (dashed lines). Audio synthesis, game state, persistence, and in-app purchase are kept separable so that a change to one rarely forces a change to another.



(a) In-maze view. The compass triangle marks heading; the ping button issues an audible probe.

(b) Result screen. Pings, moves, time, wall hits, and collectibles combine into a composite score.

(c) Level select. Difficulty tabs span Beginner through God; locked levels carry a padlock.

Fig. 3. Three screens from The Listening Maze: in-maze gameplay, the result screen, and level select.

economical use of the ping is rewarded over exhaustive search. Fig. 3c presents the six difficulty bands, from Beginner to God, across which the 120 shipped levels are distributed.

D. Accessibility Engineering

Accessibility is not an additional layer but a constraint on the core architecture. Full VoiceOver compatibility is implemented on all controls, with dynamic announcement of game-state changes (level start, wall hit, exit reached). Haptic feedback through `CoreHaptics` provides a non-auditory parallel channel for key events. The player can play with sound, with haptics, or with both. Dark mode is forced for high-contrast presentation, and there is no mandatory time-pressure mode. The campaign can be played without a timer for players who use screen readers.

E. Defect Severity Criteria and Audit Outcomes

Table I reports twelve critical defects identified during audit. A raw count is uninterpretable in the absence of an explicit severity rule; the rule applied here is therefore stated in full. A defect recorded in project memory was classified as *critical* if it was reproducible from a written sequence of steps and satisfied at least one of the following conditions:

- Unwinnable or unrecoverable state The defect could leave a level impossible to complete, or the app in a state from which the player could not return to the menu without terminating it.
- Loss of the primary feedback channel The defect could deprive a blind player of information available to a sighted player, thereby breaking the redundancy constraint that the whole design rests on.
- Data or entitlement loss The defect could discard saved progress or fail to restore a purchased entitlement.
- Crash or hang on a supported device and OS version.

Defects that were cosmetic, that affected only visual presentation while leaving the audio channel intact, or that could

not be reproduced deterministically were recorded but not counted. Two consequences follow. The rule is one of severity rather than of origin, so a defect is counted irrespective of whether the assistant or the author introduced it; and criterion 2 is specific to this product, since a different accessibility target would require a different second criterion. The count is therefore not offered as a portable benchmark.

Two examples illustrate the classes. Under criterion 2, an early version issued the haptic for each game event twice, once from the `GameManager` that owned the game state and once from the `SpatialAudioEngine` that observed the corresponding audio event. Each invocation was correct in isolation, but for a player whose sole feedback channel was haptic, the duplication rendered two distinct events indistinguishable from one. The remedy was a strict ownership rule under which haptics are triggered only from `GameManager`; the rule was recorded in project memory so that subsequent edits preserve it. Under criterion 1, a connectivity defect in the maze generator could, for certain seeds, produce a layout in which the exit cell was unreachable from the start cell, and a bounds-check defect in the same generator could index outside the grid on the outermost ring. Both were identified during rigorous-audit passes rather than through play.

The complete classified log resides in the project-memory file rather than in this study. The criteria are reported so that the count is auditable in principle, and comparison with defect counts obtained under different severity rules should be avoided.

F. LLM Contributions and Caveats

The LLM accelerated several categories of work: SwiftUI and `AVAudioEngine` boilerplate, edge-case enumeration during audits (the bounds-check, sort-order, and connectivity defects were all surfaced by the LLM during rigorous-audit passes), and the mechanical work of writing App-Store-listing copy, privacy policies, and marketing materials. The LLM also handled the SwiftUI-to-Cocoa interop required to convert a captured drawing into a share image.

Several things did not delegate well and required the human author throughout. The first was the design decision that visuals must be redundant with audio. The LLM could implement this constraint, but it could not have arrived at it on its own. The second was the ergonomic question of how loud, how long, and how distance-attenuated the ping had to be for a blind player to feel oriented. This was iterative testing with human listeners, not code generation. The third was business-model calibration: pricing, free-tier scope, and what to gate behind the $1.99 unlock.

V. Case Study II: Trace Memory Game

A. Three Constructs, and Which One this App Addresses

The literature reviewed in Section II-C spans three constructs that are frequently conflated in product marketing and, on occasion, in research writing. They are distinguished explicitly here, since the distinction determines what this study is entitled to claim.

1) Clinical intervention: A treatment intended to prevent, delay, diagnose, or treat a disease or its symptoms, evaluated against clinical endpoints in controlled trials and, in most jurisdictions, regulated as a medical device when delivered as software.

2) Cognitive training: Structured, repeated practice on tasks designed to improve one or more targeted cognitive abilities, carrying an explicit expectation of transfer beyond the trained task. This is the construct that the brain-training critique of Simons et al. [2] addresses, and the one on which far-transfer evidence is weakest.

3) Cognitive engagement: Voluntary participation in a cognitively demanding activity, valued for the activity itself, with no claim that performance gains transfer beyond it.

Trace Memory Game addresses *cognitive engagement* only. It is not a clinical intervention: it is not regulated as a medical device, was not evaluated against any clinical endpoint, and its store listing employs no medical-device language. Neither is it cognitive training: no transfer claim, near or far, is advanced, and no cognitive outcome has been measured. The dementia epidemiology cited in Section II-C serves to explain why this design space is commercially crowded and why the positioning of a product within it required care. It functions as market context rather than as a premise in any argument concerning the application, and no causal relation between play and modified dementia risk is asserted or implied anywhere in this study. The appropriate reference class is the crossword puzzle: a cognitively demanding recreational activity that carries no health claim.

B. Motivation and Target Community

The App Store's response to the dementia evidence base is dominated by exercise-style brain trainers whose far-transfer claims have been forcefully critiqued [2]. Trace Memory Game occupies a narrower position. It is a short, visually elegant memory-recall game designed to be played daily, and it sits in the same broad category of casual puzzle games as recent tablet-based pilots in healthy adults and older people [16], without inheriting their outcome claims. The crossover thesis applies again: the game is designed primarily for players with an interest in their own memory performance, but its core challenge, the reproduction of a shape glimpsed for a fraction of a second, is intended to engage players without that interest.

C. Core Mechanic: Flash, Recall, Score

A round walks through four phases, governed by a `RoundPhase` state machine and illustrated on a common time axis in Fig. 4:

- Ready (0.8 s countdown).
- Showing. A shape is rendered at a difficulty-dependent position and scale for $\Delta_{\text{flash}} \in [0.3, 3.0]$ s.
- Recall. The shape disappears, the canvas becomes a drawing surface, and the player has $\Delta_{\text{draw}} \in [4, 12]$ s to redraw the shape from memory in the same place at the same size.
- Scored. A composite accuracy is computed and shown.

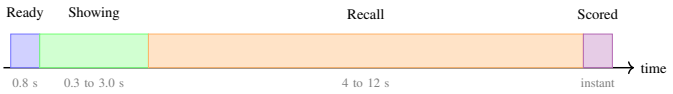


Fig. 4. A single round of Trace Memory Game walks through four phases: a brief Ready countdown, a Showing window in which the target shape is rendered, a Recall window in which the player redraws the shape on a blank canvas, and a Scored phase that displays the composite accuracy. Bar widths are drawn proportional to the durations at the easiest setting.

The proportions in Fig. 4 substantiate the design rationale. The figure is drawn to the easiest setting, at which the shape remains visible for 3.0 s and the recall window extends to 12 s, so that the recall window exceeds the encoding window by a factor of four. Over the admissible parameter ranges $\Delta_{\text{flash}} \in [0.3, 3.0]$ s and $\Delta_{\text{draw}} \in [4, 12]$ s, the ratio $\Delta_{\text{draw}}/\Delta_{\text{flash}}$ varies between approximately 1.3 and 40, and the recall window is longer than the encoding window at every admissible combination. The round is therefore predominantly an act of retrieval rather than of copying, and it is this property that distinguishes the task from a tracing exercise.

The choice of sub-second flash durations is deliberate. The visual-memory literature, surveyed by Brady et al. [19], treats visual memory as capacity- and fidelity-limited, with performance depending on how stimuli are represented as well as on their complexity. The difficulty knobs of Trace Memory Game are designed to expose this curve. At the easiest setting (3.0 s flash, low position variance, generous similarity tolerance) almost any adult can score well; at the hardest setting (0.3 s flash, high variance, tight tolerance) the task pushes against the limits of visual working memory.

D. Scoring Model

The composite accuracy is

$$A = A_{\text{shape}} \cdot A_{\text{position}} \cdot A_{\text{scale}}, \tag{1}$$

where, A_{shape} is curve similarity after aligning centroids and scales, A_{position} is a smooth fall-off in centroid distance, and A_{scale} is a bounding-box size ratio mapped through a smoothstep. The multiplicative form in (1) ensures all three components matter: a player who reproduces the shape perfectly in the wrong place scores near zero.

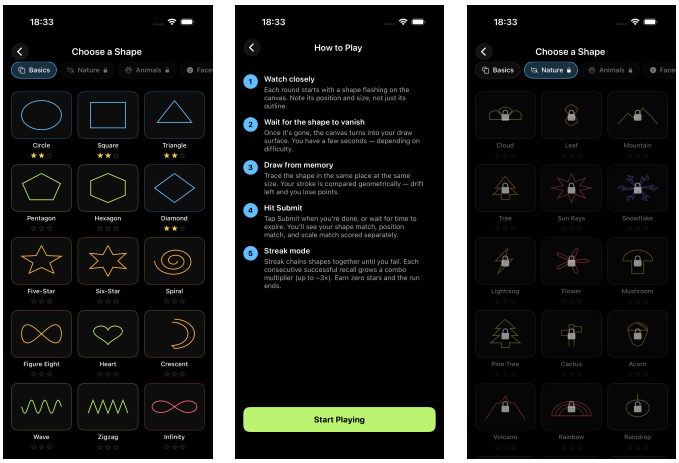
The displayed total is

$$S = \big(S_{\text{accuracy}}(A) + S_{\text{speed}} + S_{\text{stroke}}\big) \cdot m, \tag{2}$$

with $S_{\text{accuracy}} \in [0, 600]$ smoothstep-curved on A, $S_{\text{speed}} \in [0, 300]$ proportional to remaining time, $S_{\text{stroke}} \in \{0, 100\}$ rewarding a single unbroken stroke, and $m \in [1.0, \approx 3.0]$ a combo multiplier in chained modes.

E. Shape Catalogue

The shipped catalogue contains 141 shapes across eight categories: Basics (15, the free-tier pack), Nature (18), Animals (17), Faces (18), Zig-zags (20), Constellations (13), Symbols (14), and the full Alphabet (26 letters). Every shape is implemented in code, as a sequence of normalized `CGPoints` rendered through SwiftUI's `Path` API. There are no PNG or SVG assets shipped with the game.



(a) Shape catalogue, Basics tab. Star ratings record per-shape best performance.

(b) How-to-play tutorial. Five steps describe a round: watch, wait, draw, submit, streak.

(c) Shape selector, Nature category. Padlocks mark shapes that require the Pro purchase.

Fig. 5. Three screens from Trace Memory Game: the free Basics catalogue, the how-to-play tutorial, and a paid category in the shape selector.

This is more than a binary-size optimization. Every shape can be regenerated at any scale, jittered procedurally per round, and rotated or warped in future game modes without the asset-pipeline overhead that a graphical-asset workflow would impose. It also allowed the assistant to contribute to the catalogue: roughly a third of the shipped shapes were drafted in code by the LLM from a prose specification such as "a stylized treble clef" or "Cassiopeia as five connected points," then audited and adjusted by the human. This proportion is the author's retrospective estimate rather than a figure recovered from a log, for the reason given in Section III-D, and it should be read with the tolerance that implies.

The catalogue and tutorial surfaces produced by this code-only approach are shown in Fig. 5. Fig. 5a presents the free Basics tab, in which per-shape star ratings provide an incentive to revisit individual shapes rather than to exhaust the catalogue once. Fig. 5b presents the five-step tutorial, which carries the entire explanation of the mechanic, the game having no tutorial level. Fig. 5c presents a paid category, in which padlocks mark the boundary between the 15 free shapes and the 126 available under the $2.99 unlock.

F. LLM Contributions and Caveats

As with The Listening Maze, the LLM accelerated boilerplate, edge-case enumeration, and asset-style code (procedural shapes here, procedural audio buffers there). The `ScoringEngine`, where the composite accuracy of (1) and its sub-scores are computed, was largely LLM-drafted from a mathematical specification provided by the author, then iterated through several rounds of perceptual calibration on real drawings.

Several decisions could not be delegated. The most consequential was the revision of the chained mode from "Endless"

to "Streak" and the introduction of a one-strike fail rule, under which the mode terminates on the first zero-star round rather than continuing indefinitely. The assistant had implemented the original "Endless" specification correctly, but playtesting established that the absence of a fail state removed the tension on which the mode depended. The revision was a design decision taken by the author; the assistant served as an interlocutor for examining its implications but was not its source. The division illustrated here is general: the specification of what the game is remains with the developer, and the assistant contributes to its implementation.

VI. The Crossover Thesis

This section states, in a form intended to be portable across products, the design principle that runs through both case studies. We call it the *crossover thesis*, and we offer it as a design heuristic and a testable conjecture, not as a validated criterion.

A. Statement

A community-centered game *passes the crossover test* when it answers *yes* to both of the following questions:

- Does a specific underserved community get a real, useful product?
- Would a mainstream player who has never heard of that community still find this game interesting?

The conjecture attached to the test is that passing it contributes to the commercial defensibility of a community-centered product developed by a very small team. The limits of this formulation require emphasis. Passing the test is not claimed to be *sufficient* for commercial viability, since distribution, marketing, and timing intervene independently of design. Neither is it claimed to be *necessary*: a sufficiently large or sufficiently well-funded niche may sustain a product that fails gate 2, and a specialist studio operating under institutional funding is subject to different constraints altogether. An earlier formulation of this work stated the test as a biconditional. Two case studies cannot support a claim of that strength, and the biconditional formulation is withdrawn. The claim advanced here is correspondingly narrower: the two gates articulate a distinction that proved decisive in the design practice reported below, and they can be evaluated at negligible cost prior to implementation.

B. Why Both Gates Matter?

The first gate is the ethical commitment. It excludes products in which the underserved community functions as a marketing adjunct to an otherwise generic design, of which the addition of a high-contrast theme to an otherwise unmodified application is the characteristic instance. In a crossover-compliant design the community's condition shapes the core mechanic itself. Echolocation is not an accessibility setting in The Listening Maze; it is the entire puzzle. Brief-flash memory recall is not therapy-flavored exercise in Trace Memory Game; it is the game.

The second gate is the sustainability commitment. Recall, from Section II-E, that the App Store revenue curve is steeply power-law: a solo developer cannot generally cover production costs out of a niche audience alone. The mainstream audience underwrites the niche. Section VII-C makes that dependence quantitative. The crossover test is what makes the cross-subsidy defensible without exploitation: if the same mechanic that serves the niche community also engages a mainstream audience, then the subsidy flows in a direction that the niche community would itself endorse.

C. Operationalizing the Test

In practice we have found three concrete checks useful when proposing a new concept:

- Identify the underserved community and the prevalence figure that defines it.
- Identify the design constraint imposed by that community and describe it in one sentence (for example, "no information may be conveyed by color alone").
- Restate the resulting mechanic without mentioning the community at all. If what remains is interesting, the crossover test is passed; if what remains needs the accessibility framing to be interesting, it is not.

D. What the Two Cases Do and Do Not Establish?

Both case-study products survive step (3) in the author's judgment. That judgment constitutes the principal weakness of the argument, and it is stated here without qualification. Step (3) asks whether a mainstream player would find the mechanic interesting; it was answered by design reasoning rather than by consulting mainstream players. No study was conducted with participants outside the target communities, no preference or engagement data were collected, and no store metric is reported that might serve as a proxy. The claim that either game possesses mainstream appeal is therefore treated throughout this study as a *hypothesis*, and it is recorded as such in Table II.

The hypothesis admits a direct test, and one is planned: a between-subjects study in which participants with no connection to either community complete a short session and report interest, perceived difficulty, and intention to continue would resolve gate 2 for these two products. Establishing the broader conjecture would require a substantially larger design, namely a corpus of community-centered products coded for crossover compliance and commercial outcome across multiple developers and platforms. Pending such evidence, the thesis is advanced as a heuristic and offered to other investigators for evaluation or refutation.

VII. Discussion

A. Epistemic Status of the Claims in this Study

The evidential value of a case study depends on a separation between what was observed, what is inferred from those observations, and what is proposed for subsequent testing. Table II classifies each substantive claim advanced in this study accordingly. *Observations* are properties of the two projects recorded in the codebase, the project-memory file, or the App Store listing, and verifiable by any party with access to those artifacts. *Interpretations* are readings of those observations

TABLE II. EPISTEMIC STATUS OF THE PRINCIPAL CLAIMS IN THIS STUDY

Claim	Status	Basis, and what would be needed to establish it
The four-step loop of Section III-B was used throughout both projects, with a persistent project-memory file.	Observation	Development record; committed `project-memory.md` files.
Twelve defects meeting the criteria of Section IV-E were caught during audit passes in The Listening Maze.	Observation	Classified defect log in project memory; criteria in Section IV-E.
Both apps ship with zero external dependencies and zero bundled media assets.	Observation	Codebase and Xcode project.
Roughly a third of the 141 shapes in Trace Memory Game were LLM-drafted.	Interpretation	Author's retrospective estimate; no authorship log (Section III-D).
Both apps are released and publicly downloadable on the App Store.	Observation	App Store listings; silent on adoption or revenue.
LLM assistance lowered the marginal cost of enumerating edge cases and of writing framework boilerplate.	Interpretation	Consistent with [5], [6], [7]; not instrumented here. Requires a within-developer A/B design.
Human audit is the step that keeps the combination useful, and the project-memory file is what makes audit cheap.	Interpretation	Consistent with [8], [12]; no condition without the memory file was run.
The content budget reachable by a solo developer at break-even is larger with LLM assistance than without.	Interpretation	Break-even model of Section VII-C and the field-experiment effect sizes; hours were not logged.
Both games have appeal for mainstream players outside the target communities.	Hypothesis	Design judgment only. Requires a study with non-community participants (Section VI-D).
Passing the crossover test is what makes a community-centered Indie product commercially defensible.	Hypothesis	Two illustrative cases. Requires a coded corpus across developers and platforms.
The solo-developer break-even point has moved inward across the mobile market as a whole.	Hypothesis	A market-level claim two products cannot evidence; [17] is suggestive only.
Playing Trace Memory Game produces any cognitive benefit.	Not claimed	Out of scope; would require a randomized trial with cognitive endpoints (Section V-A).

consistent with the literature reviewed in Section II but not established by the data reported here. *Hypotheses* are claims the two cases are insufficient to support, stated in a form permitting subsequent testing or refutation.

B. What LLM-Assisted Development Appears to Change?

Three changes were observable over the course of the two projects. Each is classified as an interpretation in the sense of Table II.

The first concerns content budget. The 141-shape catalogue of Trace Memory Game and the six-difficulty by twenty-level structure of The Listening Maze would, in the author's estimation, have required additional personnel, purchased assets, or a reduction in scope under a workflow without LLM assistance. Section VII-C establishes the arithmetic significance of this observation, since N is linear in development hours.

The second concerns the cost of auditing for subtle correctness defects, which appears to be lower. The twelve critical defects classified in Section IV-E were identified during rigorous-audit passes whose marginal cost was a single prompt, and an unaided solo developer would plausibly not have executed such passes with comparable regularity. The effect of the assistant cannot, however, be separated from the effect of having adopted a checklist discipline at all; a developer applying the same checklist manually might have identified the same twelve defects.

The third, and the least anticipated, concerns the cost of initiating work. This observation corresponds to a recurring theme in the qualitative study of Indie developers by Panchanadikar and Freeman [11], whose participants characterized generative AI as an "ill-informed co-worker" that is imperfect but reduces the barrier to beginning a task.

C. An Indicative Break-Even Analysis

The title of this study advances an economic claim, and qualitative reasoning is insufficient to support one. The underlying arithmetic is therefore stated explicitly. No figure for the hours actually expended on the two projects is disclosed, and no conclusion below depends on one; the model is parametric, so that alternative values may be substituted.

TABLE III. PAID UNLOCKS REQUIRED TO REACH BREAK-EVEN

Hours H	**Cost** Hw	N **at \$1.99**	N **at \$2.99**
200	\$8,000	4,730	3,148
400	\$16,000	9,459	6,295
600	\$24,000	14,189	9,443
800	\$32,000	18,918	12,591

a. Computed from (3) with $w = \$40$/h and $r = 0.85$.

Let H denote development hours, w the developer's opportunity-cost hourly rate, p the unlock price, and r the developer's revenue share. Apple's Small Business Program sets $r = 0.85$ for developers below the one-million-dollar annual threshold, which encompasses any product in the regime considered here. Disregarding the \$99 annual membership fee, which is negligible relative to labor at any realistic H, the number of paid unlocks required to recover the opportunity cost of development is:

$$N = \frac{H\,w}{r\,p}. \tag{3}$$

Table III evaluates (3) at $w = \$40$/h for the two price points actually used by our products.

Three implications follow, the third of which qualifies the argument advanced elsewhere in this study.

First, N is linear in H. Any intervention reducing the hours required for a fixed feature scope reduces the required unlock count in exact proportion. The estimate of Cui et al. [6], a 26.08 % increase in completed tasks, corresponds under a fixed-scope assumption to completion in a fraction $1/1.2608 \approx 0.79$ of the hours otherwise required, and therefore to a reduction of approximately 21 % in both H and N. The larger figure reported by Peng et al. [5] is set aside here, having been obtained on a single self-contained task rather than on a multi-month product cycle. A reduction of the order of one

fifth in required volume constitutes the whole of the economic mechanism described in this study, and is a modest effect when stated in these terms.

Second, the required unlock counts are small in absolute terms. Across the range tabulated, break-even is reached at between roughly three thousand and nineteen thousand unlocks, volumes that characterize a long-tail product rather than a commercially prominent one. It is in this sense that such a product becomes economically defensible for a single developer.

Third, and in tension with the framing adopted above, N enumerates *paid unlocks* rather than downloads. Under a freemium structure with a free tier, unlock conversion is typically in the low single digits. At a conversion rate of 2 %, the 4,730 unlocks in the first row of Table III correspond to approximately 236,000 downloads. Evaluated against the Pareto rank-to-demand relationship estimated by Garg and Telang [3], a download volume of that order is not attainable from a niche community audience in isolation. This result is interpreted here as the strongest quantitative support the study offers for the crossover thesis rather than as evidence against it: the arithmetic establishes that gate 2 is not an optional refinement but a necessary condition of the business case. At these price points, a product addressed exclusively to the community audience does not reach break-even, irrespective of the reduction in development cost that LLM assistance affords.

The limits of this exercise require statement. Eq. (3) is an opportunity-cost model rather than an account of realized finances; it excludes marketing expenditure, localization, ongoing maintenance, and the value of expertise acquired; and the conversion rate invoked in the third implication is an industry rule of thumb rather than a measurement obtained from either product. The model is offered as a checkable structure for the argument, not as an empirical result.

D. What it Does Not Change?

Three requirements are unaffected by the availability of an LLM assistant. The first is direct contact with the domain. The assistant has no access to the lived experience of blindness and cannot determine whether the amplitude and duration of the ping are adequate for orientation; that determination was obtained only through iterative testing with human listeners. The second is human design judgment at the level of mode structure, exemplified by the "Endless" to "Streak" revision in Trace Memory Game. The third is the question of standing: the entitlement to develop a product for a community is established through testing, consultation, and revision, and cannot be conferred by a tool. Responsibility for everything the released system contains likewise remains with the developer. The meta-analytic literature on human-AI complementarity reports that combinations underperform the stronger party on average and confer benefit principally in tasks on which the human is already competent [8]; the methodology described in Section III is constructed around that finding.

E. Threats to Validity

This study is a case study of two released products by a single author, with no controlled comparison against a non-LLM baseline. Several threats follow.

1) Self-report bias: The account of what the LLM contributed is the author's, and, as set out in Section III-D, no instrumented log of accepted or rejected suggestions was kept. The one authorship proportion we give, for the shape catalogue, is a retrospective estimate and inherits the same bias.

2) Selection bias: We report on products that were released; we do not report on abandoned projects, where the LLM-assisted method may have contributed to failure as well as to success.

3) Single developer: The methodology may not generalize to teams, where project memory has to be co-curated and where audit cycles interact with code review.

4) Single platform: Both products are SwiftUI and iOS; we do not speak to Android, web, or console contexts.

5) No user evaluation: No study was conducted with players from either target community or from outside it. Claims about mainstream appeal are hypotheses (Section VI-D).

6) No commercial outcome data: Release is not adoption. We report no downloads, conversion, retention, or revenue, so the break-even analysis of Section VII-C is a model, not a result.

7) No cognitive outcome: We make no claim about whether Trace Memory Game produces measurable cognitive benefit; per Section V-A, it is not designed to.

F. Ethical Considerations

1) Testers were not research participants: During development, both products were shown to fewer than ten friends and acquaintances, who offered verbal reactions of the kind customarily solicited before a commercial release. The boundary is drawn explicitly here because the distinction is material for an academic venue. The activity constituted product quality assurance rather than human-subjects research: no protocol was followed, no research question was posed to those individuals, no instrument was administered, no systematic or comparative design was employed, and nothing said was recorded, transcribed, measured, or retained. No datum originating from a tester appears anywhere in this study, and no claim in Table II rests on tester feedback. Because no human-subjects data were generated or analyzed, no ethics review was sought, and none was required under the applicable institutional policy. Should a future version of either application be evaluated as an intervention or as a study instrument, whether with respect to accessibility outcomes, mainstream appeal (Section VI-D), or cognitive endpoints, that work will be designed as human-subjects research and submitted for ethics approval prior to the enrolment of any participant.

2) Clinical claims: Trace Memory Game is positioned as cognitive engagement, in the sense defined in Section V-A. It is neither cognitive training nor a clinical intervention. The app does not use medical-device language, and the listing materials are correspondingly conservative.

3) Data privacy: Both apps store all state locally in `UserDefaults` and ship with the "Data Not Collected" privacy nutrition label. No telemetry, analytics, or third-party SDKs are present in either binary. The privacy policies are

hosted as static HTML on GitHub Pages. One consequence of this design choice should be noted: the absence of data collection is the reason the adoption figures discussed in Section I-A cannot be reported.

VIII. Conclusion

This syudy has documented, end-to-end, what LLM-assisted Indie development looks like when it is pointed at communities that the App Store routinely underserves. Two iOS products, a spatial-audio maze game for blind and low-vision players and a memory-recall game positioned for cognitive engagement, were built by a single author in pair-programming collaboration with an LLM coding assistant and released on the App Store. We have described the human-LLM loop that produced them, the role of a curated project-memory artifact in keeping the loop coherent across sessions, the severity criteria behind our defect counts, and a parametric break-even model that makes the economic argument checkable.

The empirical contribution is narrow: two released products, one author, one LLM coding assistant, no control condition, and no user study. Table II records which of the study's claims this evidence supports and which it does not. The observation is that a single developer, with an LLM assistant in the loop, brought two accessibility- and cognition-oriented products to public release without external dependencies, outsourced assets, or additional personnel. The interpretation is that the marginal cost of the components for which the assistant is well suited, namely framework boilerplate, edge-case enumeration, and asset-style code, has fallen sufficiently to be consequential at this scale. The hypothesis, which the present evidence cannot establish, is that this shift applies to community-centered products generally, and that the crossover test identifies the design property determining whether a given product benefits from it.

The break-even arithmetic of Section VII-C qualifies that argument in one important respect: reduced development cost is not by itself sufficient. At realistic conversion rates a product addressed exclusively to the community audience does not reach break-even at these price points, however rapidly the code is produced. Gate 2 of the crossover test is therefore a necessary condition of the economic case rather than an ancillary consideration. The evidence offered for the position named in the title consists of two applications on the App Store and a set of claims this study has sought to classify accurately.

Declaration on Generative AI

In accordance with SAI policy on the use of generative AI tools, the author discloses the following. The two software artifacts that are the subject of this study were implemented in continuous pair-programming collaboration with a commercial large-language-model coding assistant, as described in Section III; that collaboration is itself part of the reported subject matter. In the preparation of the manuscript, a generative AI tool was additionally used for language refinement and formatting. All scientific ideas, arguments, design decisions, analysis, and conclusions are the author's own. The author reviewed and edited all AI-assisted text and takes full responsibility for the accuracy, originality, and integrity of the manuscript. No generative AI tool is listed as an author.

Acknowledgment

The author thanks the early testers of both applications, whose critique shaped the products, and the anonymous reviewers, whose comments improved the framing, the evidential standards, and the economic analysis of this study.

References

[1] G. Livingston, J. Huntley, K. Y. Liu, S. G. Costafreda, G. Selbæk, S. Alladi, D. Ames, S. Banerjee, A. Burns, C. Brayne, N. C. Fox, C. P. Ferri, L. N. Gitlin, R. Howard, H. C. Kales, M. Kivimäki, E. B. Larson, N. Nakasujja, K. Rockwood, Q. Samus, K. Shirai, A. Singh-Manoux, L. S. Schneider, S. Walsh, Y. Yao, A. Sommerlad, and N. Mukadam, "Dementia prevention, intervention, and care: 2024 report of the Lancet standing Commission," *The Lancet*, vol. 404, no. 10452, pp. 572–628, Aug. 2024.

[2] D. J. Simons, W. R. Boot, N. Charness, S. E. Gathercole, C. F. Chabris, D. Z. Hambrick, and E. A. L. Stine-Morrow, "Do 'brain-training' programs work?" *Psychological Science in the Public Interest*, vol. 17, no. 3, pp. 103–186, Oct. 2016.

[3] R. Garg and R. Telang, "Inferring app demand from publicly available data," *MIS Quarterly*, vol. 37, no. 4, pp. 1253–1264, Dec. 2013.

[4] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford, "Accessibility in software practice: A practitioner's perspective," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 4, pp. 1–26, Oct. 2022, article 66.

[5] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of AI on developer productivity: Evidence from GitHub Copilot," arXiv:2302.06590 [cs.SE], Feb. 2023.

[6] K. Z. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz, "The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers," *Management Science*, 2026, published online 27 Feb. 2026 (Articles in Advance).

[7] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Measuring GitHub Copilot's impact on productivity," *Communications of the ACM*, vol. 67, no. 3, pp. 54–63, Mar. 2024.

[8] M. Vaccaro, A. Almaatouq, and T. Malone, "When combinations of humans and AI are useful: A systematic review and meta-analysis," *Nature Human Behaviour*, vol. 8, pp. 2293–2303, Dec. 2024.

[9] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can language models resolve real-world GitHub issues?" in *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, Vienna, Austria, May 2024.

[10] Q. Wan, S. Hu, Y. Zhang, P. Wang, B. Wen, and Z. Lu, "'It felt like having a second mind': Investigating human–AI co-creativity in prewriting with large language models," *Proceedings of the ACM on Human-Computer Interaction*, vol. 8, no. CSCW1, pp. 1–26, Apr. 2024, article 84.

[11] R. Panchanadikar and G. Freeman, "'I'm a solo developer but AI is my new ill-informed co-worker': Envisioning and designing generative AI to support indie game development," *Proceedings of the ACM on Human-Computer Interaction*, vol. 8, no. CHI PLAY, pp. 1–26, Oct. 2024, article 317.

[12] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22)*, New Orleans, LA, USA, Apr. 2022, pp. 1–7.

[13] OpenAI, "Introducing SWE-bench Verified," OpenAI Blog, Aug. 2024, accessed: Aug. 17, 2026. [Online]. Available: https://openai.com/index/introducing-swe-bench-verified/

[14] M. Lumbreras and J. Sánchez, "3D aural interactive hyperstories for blind children," in *Proceedings of the 2nd European Conference on Disability, Virtual Reality and Associated Technologies (ECDVRAT)*, Skövde, Sweden, Sep. 1998, pp. 119–128.

[15] L. B. Merabet, E. C. Connors, M. A. Halko, and J. Sánchez, "Teaching the blind to find their way by playing video games," *PLOS ONE*, vol. 7, no. 9, p. e44958, Sep. 2012.

[16] P. Urwyler, R. K. Gupta, M. Falkner, J. Niklaus, R. M. Müri, and T. Nef, "Tablet-based puzzle game intervention for cognitive function and well-being in healthy adults: Pilot feasibility randomized controlled trial," *JMIR Aging*, vol. 6, p. e46177, Nov. 2023.

[17] S. Jo, W.-S. Jung, J. Yoon, and H. Kim, "Artificial intelligence and market entrant game developers," arXiv:2509.14907 [econ.GN], Sep. 2025.

[18] D. R. Begault, *3-D Sound for Virtual Reality and Multimedia*. Boston, MA, USA: AP Professional, 1994.

[19] T. F. Brady, T. Konkle, and G. A. Alvarez, "A review of visual memory capacity: Beyond individual items and toward structured representations," *Journal of Vision*, vol. 11, no. 5, p. 4, 2011.