

Introducing Shift: A Personal AI Workspace for macOS

Jackson Liu

13 October 2026

Abstract

Shift is a native macOS application that combines a desktop agent with conversation, browsing, and editable documents. Its design addresses a practical gap in AI assistance: an answer must often be transferred into another application, checked against its source, and revised before it becomes useful work. Shift connects four surfaces: task chats for execution, Neo for personal conversation, Studio for documents, spreadsheets and presentations, and Surf for browsing with contextual assistance. This report describes that interaction model and the execution contracts behind it: task-scoped observations, bounded background workers, validated tool arguments, progress-aware stopping, and typed execution receipts. It separates implemented mechanisms and interface evidence from end-to-end effectiveness, which remains an evaluation priority for the public beta. The contribution is a documented systems design and an inspectable approach to desktop assistance.

1 Introduction

Preparing a project update involves more than generating prose. A user must gather facts, resolve missing information, write for an audience, edit the result, and retain a useful file. Similarly, building a budget involves entering values and checking formulas; learning an unfamiliar application involves finding a control and understanding what it changes. In each case, conversation can help, but it is only one part of the work.

Shift places AI assistance alongside the applications and artifacts involved in a task. A person can ask a question about the current screen, request a bounded operation, discuss priorities, or continue editing a result directly. These are related interactions with different consequences. Explaining a control should not silently become clicking it, and generating a document should not remove the opportunity to revise it.

This report is for prospective users who want to understand what Shift does and technical readers who want to examine how it constrains execution. It answers three questions: how a request enters the workspace; how the system relates proposed actions to current observations; and what evidence is needed to call a task complete. It is a technical introduction to the public beta, not a product roadmap or a benchmark result.

Design position. Shift combines direct manipulation and delegated work, following the broader mixed-initiative tradition [1]. The engineering hypothesis is that explicit handoffs, editable artifacts, and inspectable outcomes can reduce the effort between intent and result. This hypothesis has not yet been established by a controlled productivity study. Human–AI interaction guidance provides a basis for assessing capability disclosure, correction, and control [2].

2 A workspace for conversation, action and artifacts

The application has four complementary surfaces (Figure 1), accessed through a shared workspace (Figure 2). They share navigation and the user’s configured AI connection, while retaining different responsibilities. Model availability and capabilities depend on that connection; Shift is not itself a foundation model.

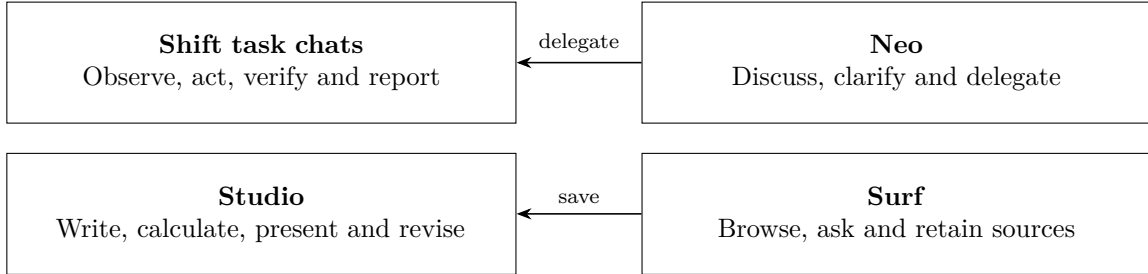


Figure 1: Product responsibilities and two implemented handoffs. The diagram does not imply that every surface automatically shares all content or authority.

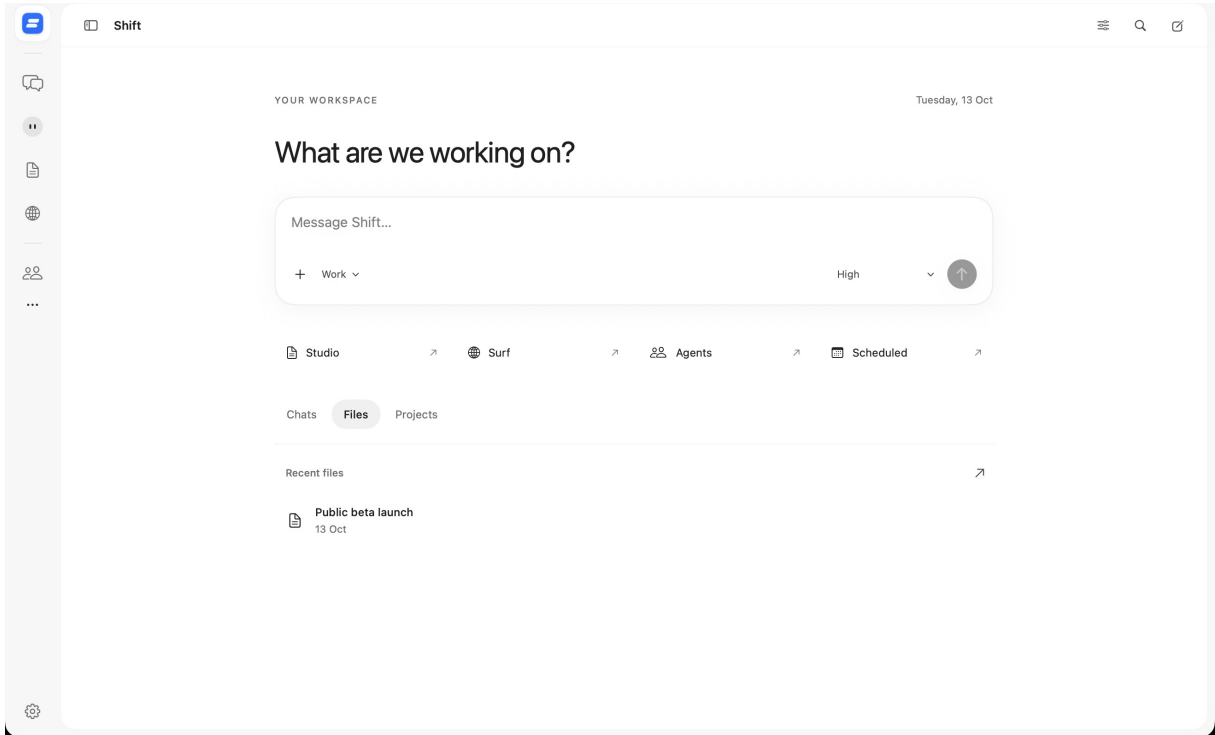


Figure 2: Shift workspace captured from the running macOS app on 13 October 2026. The navigation rail and composer provide access to task chats, Neo, Studio and Surf. Only desktop margins and the operating-system sharing strip are cropped.

2.1 Shift task chats

A task chat owns a request and its execution history. The desktop agent can observe a target application, reason about the next operation, and dispatch supported input. Visual information is combined with macOS Accessibility information when available [3]. Guidance remains a useful alternative to autonomous input: the agent can indicate a control while the user performs the action.

Requests are most useful when their outcome is checkable. “Identify the sign-in control” describes a narrower contract than “handle this website.” A task’s final message is a report from the system, not independent evidence that the external application reached the intended state. That distinction motivates the observation and receipt mechanisms in Section 5.

2.2 Neo: an ongoing conversation

Neo provides a persistent conversational surface for questions, explanations and planning. When a desktop work request is delegated, the host creates a separate Shift task chat using the existing runner. Neo can report the worker’s state and link back to the task so that the user can inspect progress, respond to a question or stop it.

The conversation and worker have separate responsibilities. Acknowledging a delegation means that work has been handed off; completion comes from the worker’s terminal state. Ordinary conversation does not execute mouse or keyboard input inside the conversational model call. This separation helps keep a conversational reply from being mistaken for an action receipt.

2.3 Studio: results that remain editable

Studio is the integrated editor for documents, spreadsheets and presentations. Its library supports file creation, naming, search, filtering, export and recoverable deletion. Document controls include formatting, an outline and selected-text revision. Spreadsheets expose cell addresses, ranges and formulas; presentation controls support slide creation and reordering. Inline `/shift` and `/visualise` requests connect assistance to the content being edited.

Review matters as much as generation. A suggested edit can be inspected before it is applied, and the user can continue direct editing afterward. Studio is suited to bounded writing, planning and structured-content tasks. Figure 3 shows a manually authored document in the actual editor.

2.4 Surf: browsing with contextual assistance

Surf embeds a WebKit browser [4], with tabs, bookmarks, history and a contextual assistant. Asking about a page sends an excerpt of that page to the configured AI. The answer can retain a link to its source, and saving it to Studio writes the answer together with that source URL. This is an explicit content transfer, rather than an assumption that all workspace surfaces can read one another’s state.

Browser operation and page discussion are distinct. Shift’s automation can interact with Surf, while a contextual question can simply produce an answer. A source link identifies where information came from; it does not certify that the answer is correct or that the page is a reliable source.

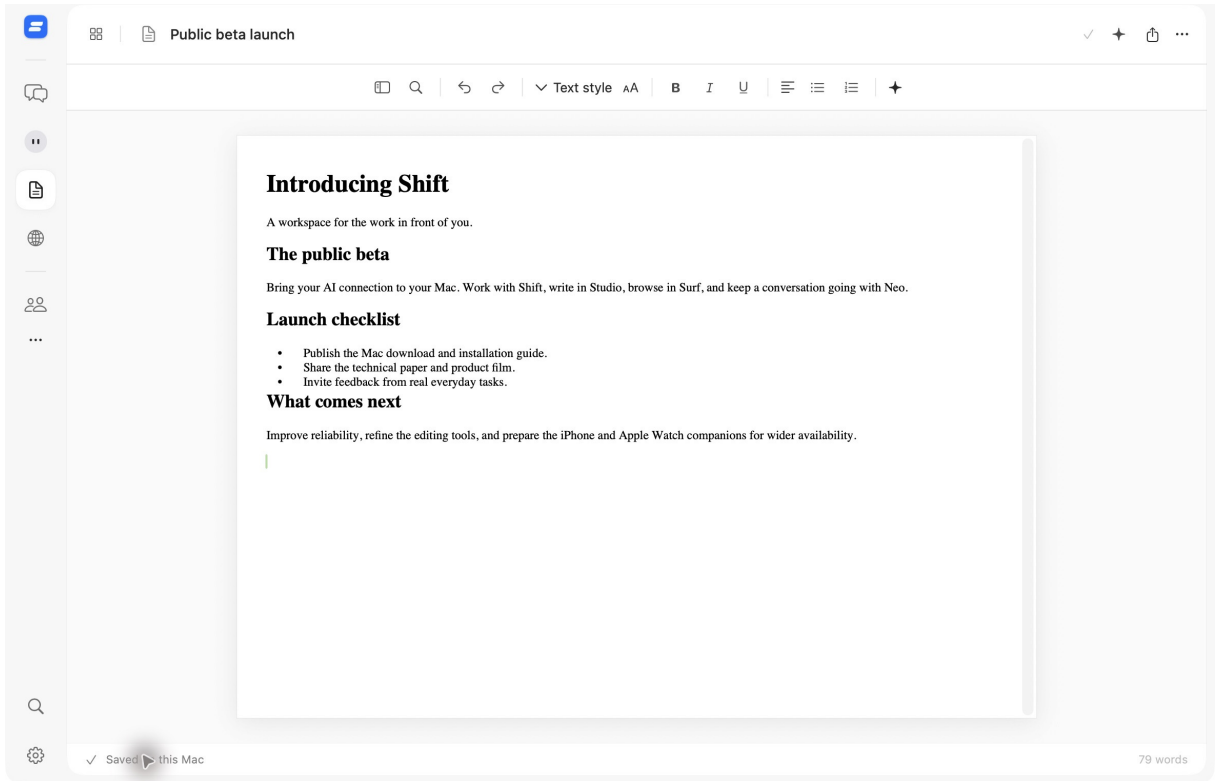


Figure 3: A launch brief created manually through Studio’s document editor, captured on 13 October 2026. The figure demonstrates an editable document and its formatting controls; it is not an AI-generated workflow result. Only the operating-system sharing strip is cropped.

3 Entering and continuing work

3.1 Access without reorganising the desktop

Different tasks call for different entry points. Shift offers deliberate cursor shaking, holding Command, the “Hey Shift” wake phrase, a compact notch surface, and the main workspace. These are configurable access paths to the same product, rather than independent agents.

Entry point	Interaction	Intended use
Cursor shake	Invoke Shift with a deliberate gesture.	Ask about something already on screen.
Hold Command	Hold the key for voice input when enabled.	Make a request without switching windows.
“Hey Shift”	Speak the enabled wake phrase.	Start a voice interaction hands-free.
Notch	Open compact task controls and status.	Inspect or control work alongside another app.
Workspace	Open chats, Neo, Studio, Surf or scheduled work.	Develop requests and return to saved work.

Table 1: Invocation methods. Voice features depend on their configuration and permissions; these paths have not been compared in a usability study.

Whichever entry point is used, a person needs to know whether Shift is listening, thinking, acting or waiting for them. They also need a visible way to stop and correct a task. Keyboard navigation and command search complement the compact, icon-led interface; icons require labels for accessibility and discoverable help even when the visible layout is minimal.

3.2 Setup with a purpose

Onboarding separates account sign-in, AI connection and operating-system permissions from personalisation. After connection, a brief three-question interview collects work context and communication preferences. The resulting context can inform subsequent assistance and optional agent suggestions, which the user can review before saving. Surface-specific introductions explain the roles of Neo, Studio and Surf rather than assuming those names are self-explanatory.

Personalisation supplies prompt context; it does not train a private model. It should be possible to correct preferences later. Credentials should be entered through their designated connection controls, not as interview answers. Permissions remain explicit macOS decisions: Accessibility supports app control, Screen Recording supports visual observation, and voice functions may need microphone and speech permissions.

3.3 Guidance before action

A development capture illustrates a narrow, observable interaction (Figure 4). The request is paraphrased as:

“Show me where to sign in to this website.”

Shift’s pointer identifies the *Log in* control. This establishes an indicated target in a captured interface state. It does not establish a completed click, authentication or a general target-identification success rate. The capture is retained as historical evidence, not as a representation of the current website or all current Shift surfaces.

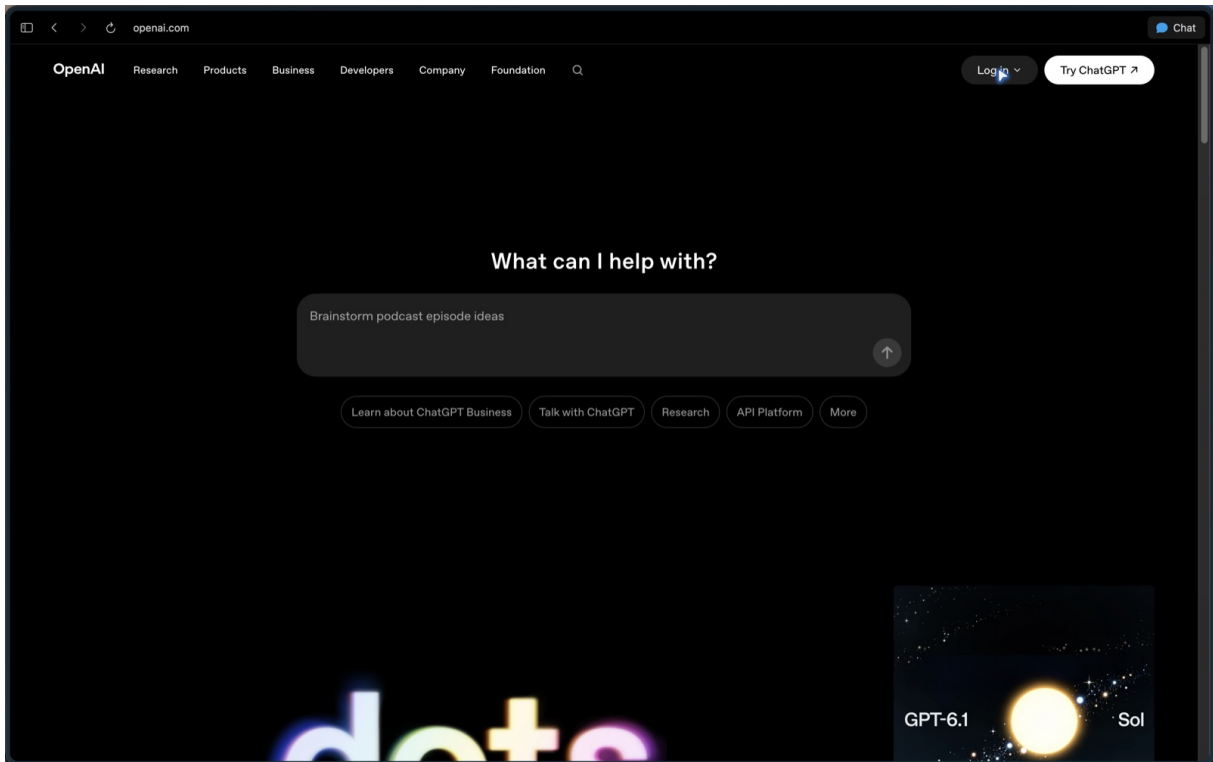


Figure 4: Contextual guidance on OpenAI.com, captured by the author on 7 October 2026. The blue pointer indicates the Log in control in the upper-right navigation. The example prompt is edited for clarity.

4 A concrete workflow: from a source to an editable plan

A representative workflow connects the product’s surfaces without requiring them to become interchangeable. The following sequence is an evaluation scenario grounded in implemented controls; it is not a timed or successful trial reported by this paper.

1. In Surf, open a source page and request a short brief covering the audience, key facts, open questions and next steps. Require missing facts to remain explicit rather than be filled in.
2. Inspect the response and its source. Save the useful answer to Studio, retaining the source URL.
3. In the document, select the next steps and request a seven-day checklist. Review the proposed revision and apply it only if it preserves the facts.
4. Discuss priorities with Neo, explicitly supplying the relevant passage. Do not assume Neo has automatically read the document.
5. Export or retain the edited file and check that its content and structure survived the transfer.

The acceptance criteria concern the result: facts must agree with the source; unknown owners and dates must stay unknown; the revision must affect the intended passage; and the final artifact must remain editable. Time saved matters only after these conditions are met.

This sequence also clarifies the division of labour. Surf supplies page context, Studio supports durable editing, Neo helps reason about priorities, and a Shift task is appropriate when an external application must be operated. The design should reduce transfers and reconstruction of context without hiding which information was actually provided to each operation.

5 Execution contracts

The desktop execution loop observes, plans, validates, acts and verifies. This pattern is related to language-model systems that interleave reasoning with actions [5] and to GUI-agent architectures [6]. Shift’s engineering emphasis is the boundary between a probabilistic action proposal and stateful desktop input.

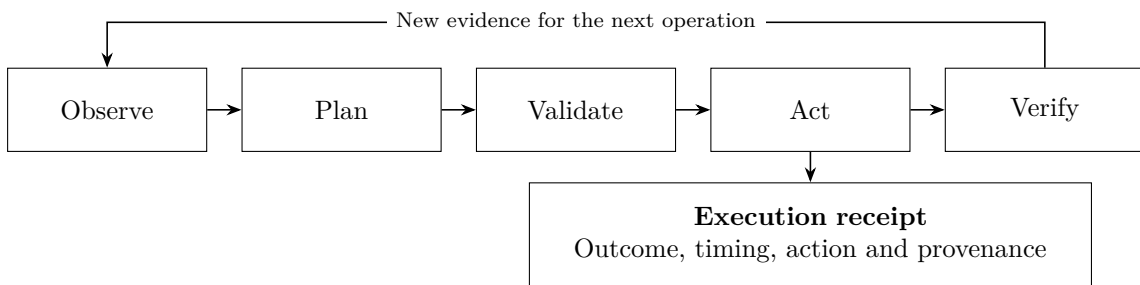


Figure 5: Simplified desktop loop. Execution diagnostics are retained separately from the bounded context used for the next model request.

5.1 Keep expensive observations off the UI path

Accessibility discovery and traversal, readiness checks, and synchronous frame acquisition run through bounded observation workers. The current shared pool permits two active jobs and four pending jobs; excess work is rejected explicitly. Shared capacity does not imply shared task observations: each job receives immutable task provenance, and mutable accepted state belongs to its task scope.

Cancellation of an awaiting Swift task cannot necessarily interrupt a synchronous operating-system call [7]. The pool therefore retains a capacity slot until the underlying operation exits, even if the caller has already timed out. Releasing it early would allow a succession of replacement jobs to defeat the concurrency bound. A completed background result still has to pass a freshness check before it can affect active state.

5.2 An observation is a versioned claim about state

Each observation ticket identifies a task, revision and request, together with monotonic request time. The controller verifies these identities on acceptance. Finishing an execution advances the revision and invalidates the previous accepted observation. The current input policy also imposes a 30-second age limit; this is a ceiling, not a guarantee that a younger observation is still correct.

```
func check(_ ticket: ObservationProvenance) throws {
    guard !ended,
          ticket.taskID == id,
          ticket.revision == revision,
          pending == ticket,
          !executing else {
        throw ObservationFailure.superseded
    }
}
```

Listing 1: Acceptance guard from `ObservationTaskState.check`.

The invariant in Listing 1 is enforced outside model-generated reasoning. A late response from a cancelled or superseded request cannot become a fresh observation for a successor task. It does not establish that a target was interpreted correctly; geometry and application state still require verification.

5.3 Validate proposals and bound working memory

Provider-side schemas help structure native tool calls, but local validation remains necessary. The native file-task loop rejects unknown actions, unexpected keys and invalid field types before dispatch. Supported multi-file reads are bounded to eight paths. Batching is useful when operations are independent; it must not bypass a needed observation after an action changes the target state.

The native loop’s textual working history is bounded to eight entries and approximately 64,000 UTF-8 bytes, with a 24,000-byte cap per entry. Truncated and omitted material is marked explicitly so it cannot silently become assumed evidence. These are bounds for that context component, not a universal limit on all model inputs or a measured token-saving claim. The durable diagnostic log is kept separately.

5.4 Recover a target without replaying input

A window may move, lose focus briefly or change geometry while a task is executing. Shift’s window-recovery policy makes up to three eligible verification or capture attempts, separated by asynchronous delays of 80 and 160 milliseconds. It checks cancellation and scope around the attempts and can re-query target identity and geometry.

Re-observing is different from repeating a click. A click may already have submitted a form or changed a file when verification fails. The recovery contract does not replay dispatched input, and receipts preserve whether input may already have occurred. Bounded recovery can handle transient interruptions; it cannot make an uncertain external transaction safe to repeat.

5.5 Stop activity that is not progress

A development failure involved repeated searches for country-attributed AI-agent statistics, followed by revisiting results, waiting and scrolling without extracting records. The important defect was a run that remained active while acquiring no useful evidence.

Browser progress tracking normalises actions and queries and compares page and evidence state. Search-result pages are navigation state, not source evidence. The default policy permits two identical attempts without new evidence and three bounded pivots; additional guards cover 12 actions without progress, eight unchanged page states and 120 seconds without meaningful progress. Limits lead to a strategy pivot or a typed terminal outcome, rather than unlimited retries.

Source validation is task-specific. Counting individual AI-agent products by country requires product-level records with an explicit country field and enough structure to count. Company statistics, generic articles and search snippets do not satisfy that unit of analysis. Validation can reject an unsuitable source, but it cannot establish a publisher’s accuracy, completeness or definition of country.

5.6 Record outcomes, including uncertainty

A receipt describes what the executor knows about one operation. Its outcome is discriminated rather than inferred from a string (Listing 2). Failure and timeout can retain uncertainty about whether input was delivered.

```
enum Outcome: Codable, Sendable {
    case success(String)
    case failure(message: String, mayHaveActed: Bool)
    case timedOut(mayHaveActed: Bool)
    case cancelled
    case noProgress(String)
    case needsReview(String)
    case rejected(ComputerExecutionRejection)
}
```

Listing 2: Outcome type from `ComputerExecutionReceipt`.

Receipts can include the action, completed-action count, frame identity, observation provenance, errors, browser-progress diagnostics and window-recovery events. A task-local recorder prevents late completion from appending to a different run. Export serialises a value snapshot to versioned JSON, with UTC ISO-8601 timestamps, fractional seconds, explicit duration values and atomic file replacement. Monotonic clocks determine elapsed time; wall-clock timestamps support inspection.

Success means the executor fulfilled its dispatch or verification contract. It does not certify the user’s whole objective. Conversely, a clear no-progress outcome is more useful than an indefinitely active run: it exposes where human review or a different strategy is needed.

6 Evidence and evaluation

Four kinds of evidence should be kept separate: code establishes an implemented mechanism; component tests exercise selected contracts; captures show interface states; and completed workflows establish a checked end result. A screenshot cannot prove task success, and a passing cancellation test cannot prove everyday usefulness.

The 12 October design pass reports 127 passing macOS test methods, representing 142 cases, across 26 selected suites. The 13 October naming and fresh-install pass reports 51 methods and 52 cases across eight selected suites. A macOS Release build and disk-image checks also passed. These differently scoped checks are development evidence, not a benchmark, a full-suite

Claim	Available evidence	Remaining evaluation
Task isolation	Task/revision checks and controlled regression tests.	Adversarial interruption during real applications.
Editable outputs	Studio editing paths and rendering/workflow tests.	Fidelity across representative imported and exported files.
Inspectability	Typed receipts and JSON export.	Whether users diagnose failures using these records.
Convenient access	Implemented entry points and interface captures.	Discoverability, accidental activation and interruption cost.
Useful overall workflow	Connected product surfaces.	Correct end states, user effort and sustained use.

Table 2: Evidence boundaries. These rows do not imply that the listed mechanisms eliminate every failure in their category.

certification or additive counts of unique tests. Appendix A records the scope and outstanding uncertainty.

6.1 A useful next experiment

Evaluation should use a fixed task set with explicit end-state checks, following the execution-based emphasis of OSWorld [8]. Shift has not been evaluated on that benchmark. Suitable local scenarios include the source-to-plan workflow, a budget with known totals, a slide revision with required facts, and guidance for an unfamiliar control.

For each scenario, preserve the initial files and application state. Record the app build, model, provider, device, network conditions and permissions. Compare manual and assisted completion under similar conditions, counterbalancing order where possible. Include unsuccessful and interrupted runs.

Report correctness first, then elapsed time, time to first useful feedback, user corrections and terminal outcome. Artifact checks should test facts, formulas or exported structure rather than the plausibility of a final message. Median time among correct completions should be accompanied by failures; otherwise an agent that stops early with wrong output can appear fast. No comparative speed or productivity result is claimed here.

7 Limits, permissions and release scope

Shift remains a beta. Models can misinterpret targets, fabricate facts or choose unhelpful strategies. Accessibility coverage varies, web pages change, and exports may need manual inspection. Background activity is also constrained by the Mac’s availability and the selected provider. A fast acknowledgement does not remove model or tool latency.

Content sent to a connected provider depends on the task: it may include a prompt, selected text, a page excerpt or a screen observation. The fact that Shift is a native Mac application does not make every AI request local. Users should review the provider and data involved, particularly before using sensitive material. Diagnostic exports can contain action details and should be inspected before sharing.

The October 2026 public-beta release targets Apple-silicon Macs, with distribution through meetshift.app. The 13 October package is development-signed and not notarised. If macOS blocks opening it, users who choose to proceed can use *System Settings* → *Privacy & Security* → *Open Anyway* after attempting to open the app, following Apple’s guidance [9]. Installation approval is separate from Accessibility and Screen Recording permissions used by desktop tasks.

The iPhone and Apple Watch companion apps are being prepared for broader availability and are not yet publicly distributed. Simulator build success does not establish physical-device

call reliability, background ringing or remote-control reliability. These capabilities require their own device and operational checks.

The repository is private at publication. This report includes selected implementation excerpts, not a complete reproduction package. Distribution and source availability remain subject to the applicable component licences. References explain the design context; they do not imply affiliation with, endorsement by, or measured equivalence to the cited systems.

8 Conclusion

Shift explores a personal workspace in which conversation, desktop execution, browsing and editing can support the same task. Its central systems choice is to treat observation freshness, cancellation, progress and outcome provenance as execution contracts rather than conversational promises. The public beta should test whether those mechanisms and the connected interface produce work that people can verify, revise and use. Future editions should add reproducible workflow results; routine feature changes belong in release notes.

Acknowledgements

AI tools assisted implementation and manuscript preparation. Third-party software is credited in the project’s attribution and licence records.

A Development validation record

The following results were recorded in the private repository’s dated development reports. Test methods and parameterised cases are distinguished, and overlapping runs must not be summed.

Date	Scope	Methods	Cases
12 Oct. 2026	Design pass, 26 focused macOS suites; zero failures.	127	142
12 Oct. 2026	Follow-up navigation check, three overlapping suites; zero failures.	14	19
13 Oct. 2026	Studio naming, onboarding, keyboard, editor and migration checks, eight suites; zero failures.	51	52

Table 3: Recent regression evidence. Counts are not task-success rates, and the runs do not cover identical scopes.

The 12 October pass also reports an unsigned iOS Simulator build, including companion targets. The 13 October pass reports a successful arm64 macOS Release build, strict nested-code-signature checks, valid disk-image verification and a tested Applications-folder installation layout. Gatekeeper assessment rejected the non-notarised package; passing a build or signature check does not change that distribution status.

An older 6 October full-suite report recorded 872 passing and 21 failing cases. The later focused runs do not by themselves resolve that entire failure set. Existing compiler warnings and unverified physical-device behaviour are also not erased by focused regression success. There is no claim of a clean full-suite rerun or an end-to-end live AI workflow in these reports.

This edition describes a changing working tree based on 7d893310, with uncommitted changes; that commit alone does not reproduce the release. Published binaries should retain an independent checksum and build record. The current edition is dated 13 October 2026 and should be revised when the execution architecture, release scope or measured evidence changes materially.

B Implementation map

The following are private-repository references for the mechanisms discussed. Historical internal filenames retain earlier product names for compatibility; Studio is the user-facing editor name.

- `Shift/Agent/ObservationWorker.swift`: bounded workers, immutable tickets and task-scoped observation state.
- `Shift/Agent/ComputerExecutionReceipt.swift` and `ExecutionReceiptLog.swift`: outcomes, provenance, task-local recording and JSON export.
- `Shift/Agent/ShiftLoopContext.swift`: bounded text history and local native-tool validation.
- `Shift/Agent/WindowRecoveryPolicy.swift` and `BrowserResearchProgress.swift`: recovery and progress policies.
- `Shift/ShiftWorkspaceOnboarding.swift`: personalisation and setup; `Shift/PipChatView.swift`: Neo conversation.
- `Shift/ShiftStageView.swift`, `StageDocumentEditor.swift`, `StageSpreadsheetEditor.swift` and `StagePresentationEditor.swift`: Studio library and editors.
- `Shift/ShiftBrowserView.swift`: Surf assistant and the explicit source-preserving save to Studio.
- `docs/Public-beta-design-pass-October-12.txt` and `docs/Studio-unboxing-October-13.txt`: recent development validation records.

References

- [1] E. Horvitz. *Principles of Mixed-Initiative User Interfaces*. Proceedings of CHI, 1999, pp. 159–166. <https://www.microsoft.com/en-us/research/publication/principles-mixed-initiative-user-interfaces/>.
- [2] S. Amershi et al. *Guidelines for Human-AI Interaction*. Proceedings of CHI, 2019. doi:10.1145/3290605.3300233. <https://www.microsoft.com/en-us/research/publication/guidelines-for-human-ai-interaction/>.
- [3] Apple. *AXUIElement*. Application Services documentation. Accessed 13 October 2026. <https://developer.apple.com/documentation/applicationservices/axuielement>.
- [4] Apple. *WKWebView*. WebKit documentation. Accessed 13 October 2026. <https://developer.apple.com/documentation/webkit/wkwebview>.
- [5] S. Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. ICLR, 2023. arXiv:2210.03629v3. <https://arxiv.org/abs/2210.03629v3>.
- [6] S. Agashe, J. Han, S. Gan, J. Yang, A. Li, and X. E. Wang. *Agent S: An Open Agentic Framework that Uses Computers Like a Human*. arXiv:2410.08164, 2024. <https://arxiv.org/abs/2410.08164>.
- [7] Swift.org. *Concurrency*. The Swift Programming Language. Accessed 13 October 2026. <https://docs.swift.org/latest/documentation/the-swift-programming-language/concurrency/>.
- [8] T. Xie et al. *OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments*. arXiv:2404.07972v2, 2024. <https://arxiv.org/abs/2404.07972v2>.
- [9] Apple. *Safely open apps on your Mac*. Apple Support, updated 1 June 2026. Accessed 13 October 2026. <https://support.apple.com/en-nz/102445>.