Telethryve: A Local Multimodal Agent Stack with Governed Embodiment

A Public-Safe Scholarly White Paper

Charles Morgan
Independent Researcher, Charles Morgan Software
Seattle, Washington

August 2026, revised edition

Abstract

Telethryve is a desktop-and-mobile software system for connecting people to coding agents, local language models, and governed computer-assisted work through a single conversational experience. Its local stack combines an Ollama-served Gemma profile with the Consciousness sidecar for governed context and continuity, and can expose a native local coding terminal with a bounded, approval-mediated tool surface. Configured sensory and action layers allow that local stack to request bounded camera observations, receive spoken input, synthesize spoken output, and use permissioned computer tools and applications. A separate robotics adapter can connect the same interaction model to bounded physical motion when compatible hardware, calibration, explicit arming, and independent safety controls are present. This revised paper presents the public design rationale for those multimodal and embodied capabilities alongside explicit backend routing, coding-agent workflows, mobile access, and purpose-bounded plug-ins. The contribution is a modular perception–conversation–action architecture that keeps model output, observation evidence, tool approval, computer permissions, and physical motion authority distinct. The paper intentionally omits operational secrets and implementation-sensitive topology.

Keywords: agentic software; local language models; multimodal interaction; embodied agents; human-in-the-loop systems; mobile computing; privacy-aware design

1 Introduction

Large language models have become useful not only as question-answering systems, but also as collaborators in software work: reading a repository, forming a plan, generating a change, running a test, and explaining the result. This shift is often characterized as *agentic* work: a language model operates in a bounded loop with tools, intermediate observations, and human direction. Research on reasoning-and-action patterns has shown why this combination can be more capable than a single isolated response [1]. Yet real adoption remains difficult. A person may be away from the desk; work may involve private files; the right model may differ by task; and a useful system must make its authority and limitations legible.

Telethryve addresses this interaction problem. It is a software concept and implementation for carrying a governed conversational work surface between a desktop computer and a mobile device. A user can begin a coding or research task on a computer, follow its progress from a phone, supply clarification, review a proposed result, and return to the desktop when a task requires attention. The interaction is deliberately collaborative: the human chooses the provider and work context, observes progress, and authorizes consequential actions. The software does not make a general claim of autonomous deployment or of replacement for expert review.

The important distinction is between *model capability* and *system capability*. A local model, a hosted model, a coding agent, a chat client, and a deterministic tool can each have different strengths. A usable system should make them composable without pretending that they are interchangeable. *Telethryve* therefore presents a user-facing work mode rather than a hidden, universal AI brain.

The current implementation makes the execution path visible rather than implying that all conversation turns share one runtime. A chat can retain a selected project, backend, and per-run identifier; more than one run may be active at a time. When a response or clarification must be associated with a specific active run, the bridge requests that identifier instead of guessing. This is a small but important form of interaction safety: continuity is preserved without silently directing a user response to the wrong task.

2 Design Premise

The design premise is simple: one conversation can be a durable, user-governed place to organize work across devices and providers. This does not mean a single undifferentiated model session. It means that the person sees one coherent work relationship while the system preserves distinctions that matter: selected model or agent,

local versus hosted processing, project context, pending questions, and the boundaries of authorized action.

This premise responds to four practical constraints. First, software work is iterative. Developers need to ask for a change, inspect a diff or artifact, run a check, and correct course. Second, many users work intermittently and mobile devices are useful for supervision, clarification, and lightweight approval rather than for replacing a development workstation. Third, privacy and reliability are contextual. A small local task may be better served by a resident model and local files; a difficult reasoning or coding task may benefit from a hosted provider selected by the user. Fourth, perception and action have different consequences. A camera observation, a spoken request, a computer operation, and physical robot motion must not inherit authority from one another merely because they participate in one conversation.

Figure 1 expresses the public model. Perception and interfaces provide attributed input; the selected local or hosted cognition layer interprets that input within its configured context; and each outcome crosses its own boundary. The desktop is well suited to authoring, inspecting files, and completing extended technical work. The phone is well suited to maintaining situational awareness, answering a clarification, or reviewing a result. A response, plan, artifact, computer operation, or robot command is presented in a form that permits human understanding and control.

3 Local AI Operation

3.1 Why a local option matters

Local language-model operation is useful when a user values control over data location, wants to work with locally available material, or needs a bounded workflow that can continue with limited connectivity. *Telethryve* includes an opt-in local profile based on an Ollama-served Gemma model. Gemma is a family of lightweight open models designed for a range of deployment scales [2]; Ollama provides a practical local model-serving environment [3]. In this framing, a local model is not characterized as a drop-in replacement for every hosted frontier model. It is an intentionally selected component with performance, hardware, and task-fit constraints.

The local profile supports everyday interaction patterns such as explanation, summarization, repository-oriented context work, and bounded assistance. It can be paired with local memory and local knowledge retrieval so that relevant user-approved context can remain on the computer. Offline-oriented operation can limit a workflow to local files, installed capabilities, and previously available material. These choices help users distinguish between a task that is genuinely local and a task that depends on a hosted service or current web information.

3.2 Deterministic support around probabilistic models

A language model is probabilistic; many desired workstation operations are not. *Telethryve* treats deterministic helpers and verification steps as complements to model inference. Examples include formatting an artifact, checking a result, retrieving selected local context, or presenting a well-defined user interface control. The purpose is not to conceal action behind a conversational veneer. The purpose is to make a workflow more reproducible and reviewable when a human asks for it.

This division also improves clarity. A response generated by a local model should be understood as a model output. A successful verification or a completed file operation should be reported as an observed outcome. Keeping those categories distinct reduces the temptation to treat fluent language as evidence. The distinction aligns with established guidance that human–AI systems should communicate their capabilities, limitations, and relevant changes in state [7].

3.3 Native local coding terminal and tool boundary

The local coding terminal is a distinct, inspectable execution surface. It launches the packaged Codex terminal with a loopback-only transport to the selected local Ollama model and a Telethryve MCP server. The transport performs protocol adaptation; it does not read or select prompts, choose tools, modify tool arguments, decide approvals, or replace the coding agent's native user interface. The terminal keeps its own Telethryve-scoped session state and injects only the bounded Telethryve MCP surface, avoiding accidental inheritance of unrelated operator extensions.

Each exposed tool remains subject to its own sandbox and approval rules. In particular, non-read-only Telethryve MCP operations require the terminal's write approval mode. Tool-control events retain bounded status metadata while full tool results return to the coding agent. Consequently, a tool invitation, a model statement, a receipt, and a verified external outcome remain different claims. This design does not make a local model equivalent to Codex or confer unrestricted computer authority; it makes the boundaries of a local coding session more explicit.

A public conceptual view of the *Telethryve* local stack

Perception and interfaces		**Governed cognition**		**Bounded outcomes**
Named camera observation	⟶	Consciousness sidecar and Gemma	⟶	Responses and artifacts
Microphone and speech		Selected hosted or coding agent		Permissioned computer action
Desktop and mobile interfaces		Task-appropriate tools		Separately armed robot motion

Observations, model outputs, tool receipts, and motion authority remain distinct throughout the interaction.

Figure 1: A conceptual multimodal interaction model. This figure describes user-visible roles and authority boundaries rather than operational topology.

4 Multimodal Local Agency and Governed Embodiment

4.1 Consciousness sidecar and local continuity

The local stack pairs the selected Gemma profile with a component named the **Consciousness sidecar**. The sidecar supplies governed identity material, scoped context, local memory, and continuity information before an applicable model turn. Its name identifies a software component; it is not a scientific claim of sentience or human-equivalent consciousness. The model still produces probabilistic outputs, while deterministic policy layers retain authority over tools, permissions, and consequential actions.

This separation makes continuity useful without making it sovereign. Context can help the model remember a project, interpret a current request, or maintain a consistent working posture. It cannot grant microphone access, open a camera, approve a desktop action, connect a robot, arm motors, or clear an emergency stop. Those transitions belong to their respective operating-system and hardware boundaries.

4.2 Seeing, hearing, and speaking

Visual perception is exposed through the model-facing capability **aura_look**. When current physical context would materially improve a response, Aura may request one fresh still image from an explicitly named camera. The local vision layer bounds the image, runs object detection, and returns structured detections together with the associated image evidence. The result declares that motion is not authorized. It is an observation for reasoning and human review, not a control command.

The voice path completes a second local interaction loop. A configured microphone and speech-recognition service can convert a spoken request into text for the active conversation. Completed text can be synthesized through a local speech service and streamed to the selected audio output. Hearing and speaking therefore describe an explicit data path—capture, transcription, model turn, synthesis, and playback—rather than a metaphor for text input and output. Each stage depends on visible device permissions, available local services, and a selected input or output route.

4.3 Computer and application action

Permissioned computer action extends the interaction beyond generation. Structured tools can inspect files and system state, operate supported applications, use browser or accessibility surfaces, and verify the resulting state. The model proposes or requests an operation; deterministic adapters and the operating system decide whether the operation is available and permitted. High-trust actions remain opt-in, and consequential external actions may still require explicit human confirmation.

This architecture closes a practical local loop: perceive relevant state, interpret it with governed context, communicate with the user, act through a bounded tool, and inspect the outcome. The systems-level breakthrough is the integration of those stages around one local work relationship while retaining separate evidence and authority for each stage.

4.4 Robotics layer and physical movement

The robotics layer extends the same design into a physical environment through a hardware-neutral adapter. The public command surface is deliberately small: connect, arm, move within configured limits, grasp or release where supported, disarm, reconcile, and emergency stop. Commands are validated, bounded by workspace and timing limits, associated with stable command identifiers, and recorded as receipts. Motion fails closed when the body is disconnected, disarmed, stopped, or in an uncertain state.

The default adapter is simulated and cannot move hardware. Physical movement requires an explicitly injected vendor driver, measured calibration, a defined coordinate frame and workspace, independent firmware limits and watchdogs, collision handling, a motor-power interlock, and a physical emergency stop. Vision remains

a separate input. Even a valid object detection cannot connect or arm a body, and it cannot bypass the robot controller's lifecycle gates.

4.5 Relation to embodied multimodal research

Embodied multimodal research has explored models that connect language to continuous sensory inputs and robot tasks. PaLM-E directly incorporates real-world sensor modalities into an embodied language model [9], while RT-2 represents robot actions as tokens in an end-to-end vision-language-action model [10]. *Telethryve* is not presented as either kind of trained foundation model. It is a modular systems architecture that composes a local language model, a governed context sidecar, bounded perception, deterministic computer tools, and a separately authorized robot adapter. That difference is material: the contribution is not a claim of learned robotic generalization, but an inspectable way to join multimodal interaction and physical capability without erasing control boundaries.

5 Hosted Provider and Coding-Agent Operation

5.1 The regular hosted stack

For work that benefits from hosted inference, *Telethryve* can present user-selected routes to OpenAI/Codex, Anthropic Claude, and xAI Grok services. These are provider-specific services with their own accounts, policies, availability, model behavior, and tool semantics. The design goal is interoperability at the interaction layer, not an assertion that the providers are technically identical.

OpenAI describes Codex as an agentic coding system for software tasks [4]. Anthropic documents Claude Code as a terminal-oriented coding tool that can work from natural-language requests and project context [5]. xAI documents Grok as a hosted assistant and developer platform [6]. In *Telethryve*, those services are useful when the user has deliberately selected and authenticated the applicable provider. This paper does not assume a particular commercial plan, model version, or account is available.

Provider selection is treated as a meaningful control. The chosen route should remain understandable to the user, and an unavailable route should be reported clearly rather than quietly substituted with another provider. Similarly, the presence of a provider name in the interface does not imply that a request has been executed, that an account has been authenticated, or that a result has been verified. Those are separate observable states.

5.2 Coding agents with a local LLM stack

The phrase coding agent on a local stack can otherwise be ambiguous. In this paper it means that a coding-agent experience—such as Codex, Claude Code, or a Grok-oriented coding workflow—may be selected as a user-facing coding collaborator while local models, local context, and deterministic helpers serve bounded roles around the work. The coding agent and local model need not be the same model, and the local option does not imply that it replicates the coding quality or tool behavior of a hosted agent.

This arrangement has several practical patterns:

- A local model can help organize private notes, summarize repository context, or support offline-oriented preparation before the user chooses a hosted coding agent for a harder implementation task.
- A hosted coding agent can perform a selected software task while the desktop and mobile interfaces keep the human informed and available for questions or approval.
- A local model can remain useful for lower-risk, bounded assistance after a coding task, including explanation and review support, without making claims about full coding-agent equivalence.

The innovation is not a claim that one provider has been converted into another. It is a practical separation of concerns: user interaction, model selection, task context, deterministic assistance, and human authority can be coordinated without erasing their different properties.

6 Desktop, iOS, and Bridge-Chat Interfaces

6.1 Desktop as the primary technical workspace

The desktop experience is the primary place for development work that requires a local project, longer attention, file review, testing, or artifact inspection. It can preserve a project-oriented conversation so that a user can return to a named task after interruption. The design is especially useful for software work that alternates between natural-language direction and concrete evidence: a plan, a proposed change, a test result, a document, or a question requiring human judgment. The bridge may track multiple concurrent runs within a chat and renders run-specific status or requests for input; ambiguous replies are not routed automatically.

6.2 Custom iOS application

The custom iOS app extends the same work relationship to a paired mobile device. It is not framed as an at-

tempt to reproduce every desktop developer tool on a small screen. Instead, it supports the activities for which mobility is strongest: receiving progress, reviewing compact status, responding to a question, opening a shared artifact, and initiating an explicitly bounded follow-up. Where a richer interaction is useful, the app can present conversation-native controls rather than forcing the user to memorize commands. This is a human-factors choice: context and state should be visible at the moment a decision is requested.

Media handling follows the same principle. A temporary local artifact can be shared to a paired phone for immediate use while persistent storage remains a customer decision, for example in a customer-controlled synced location. The goal is to avoid turning routine mobile delivery into an implicit hosted archive.

6.3 Telegram-compatible bridge chat

A bridge-chat interface provides an alternative familiar chat surface. It can accept ordinary text, documents, photos, and concise control commands, then relay readable status and artifacts back to the conversation. The interface is valuable where users already work in messaging-oriented habits or need a low-friction supervisory channel. It remains a transport and interaction surface, not a reason to expose internal control protocols to a chat client.

The iOS app and bridge chat are therefore complementary. The custom client can offer structured mobile controls and a tailored paired-device experience; the chat interface can offer reach and familiarity. Both are designed to preserve the user's ability to see what is being requested and to continue the same work thread across contexts.

7 Plug-in Ecosystem

Telethryve is designed to be extended by optional, purpose-bounded plug-ins. The following descriptions are functional categories, not a statement that every installation has every component active.

Phonebot deserves special care because voice and phone tasks can affect other people. Its conceptual role is to connect voice-oriented work to the same governed interaction model: clear operator intent, appropriate consent, observable status, and the ability to stop. In this way, the plug-in is an example of why provider choice alone is insufficient. A responsible system also needs authority boundaries appropriate to the consequence of the action.

Vision, voice, media, desktop-control, local-knowledge, robotics, and search components make the ecosystem useful beyond code generation. They can support a field worker reviewing an artifact, a developer listening to a summary while away from the desk, a customer retaining local project context, a workstation agent observing and operating a permitted application, or a configured robot executing a bounded physical command. Modularity also helps organizations deploy only the functions they need.

8 Innovation and Use Cases

8.1 Innovation

The central innovation is a *continuity layer for governed multimodal AI work*. Rather than centering a single model, the system centers a person and a task that may traverse desktop, mobile, local, and hosted contexts. Three features were central to the original design: provider plurality, mobile supervision, and local-first discretion. A fourth feature now extends that design.

First, **provider plurality** is explicit. The system can make room for a local Gemma profile, a hosted coding agent, or another selected service without hiding the difference. This offers a more honest response to rapidly changing model capabilities and organizational constraints.

Second, **mobile supervision** is treated as a first-class use case. Mobile devices do not merely receive notifications; they can participate in the task through status, clarification, review, and bounded controls. This is particularly relevant when agentic work is long-running or when the human must remain accountable for an outcome.

Third, **local-first discretion** gives users a practical choice about where preliminary work and contextual memory reside. It does not promise absolute privacy or universal offline capability. Instead, it makes locality, network use, and provider selection intelligible choices.

Fourth, **governed embodiment** connects perception, speech, computer action, and optional physical motion without treating them as one undifferentiated autonomy grant. The user can understand which sensor produced evidence, which tool acted, what permission applied, and whether a robot body was independently connected and armed.

8.2 Representative use cases

1. **Software development supervision.** A developer starts a task on the desktop, asks a coding agent to examine a scoped codebase, and receives progress or a clarification on a phone. The developer returns to review a change and its evidence before accepting it.
2. **Private project preparation.** A user uses the local model and local context to outline, summarize, or organize material, then decides whether a later stage should use a hosted provider.
3. **Field and travel continuity.** A consultant or small business operator remains connected to a workstation-bound project through the iOS app or bridge chat,

Table 1: Public-facing plug-in roles in the *Telethryve* ecosystem.

Component	Public role
Consciousness sidecar and Aura	Governed context, continuity, local memory, and bounded model-initiated visual observation. The sidecar informs a turn but does not grant device, computer, or motion authority.
Phonebot	Phone- and voice-oriented workflow support, with explicit operator controls and appropriate consent, authorization, and communication boundaries. It is not a license for unreviewed calling or message automation.
Vocodex and voice/media relay	Voice replies, paired-device media delivery, and conversational controls that help the desktop and phone participate in the same user-directed workflow.
Desktop stream and control	A reviewable, user-authorized way to observe or interact with permitted desktop tasks. It is intended for trusted, configured environments and does not eliminate desktop permission requirements.
Local knowledge and memory	User-governed local context, recall, and project-oriented continuity. It helps with relevance and resumption, rather than asserting infallible memory or external synchronization.
Search helpers	Fresh public-information or product-lookup assistance when the user permits networked work; local and offline modes can constrain these lookups.
Local Gemma profile	An opt-in Ollama-served local model profile for bounded local assistance and workflows that benefit from local context.
Native local coding terminal	A packaged coding-agent terminal with an isolated local-model transport, a bounded MCP surface, and native approval semantics. Its tool availability does not bypass sandbox or operating-system permission checks.
Robot body adapter	Optional bounded motion and gripper commands for a compatible, explicitly configured and armed robot body with independent physical safety controls. The default adapter is simulation-only.

receiving concise artifacts and responding to time-sensitive questions without treating the phone as a full replacement for the desktop.

4. **Accessibility and voice-oriented work.** Voice and phone components can support hands-busy or screen-limited interaction when deployed with the required permissions, consent, and review.
5. **Human-in-the-loop operations.** A user can route a bounded task to a chosen provider while retaining the opportunity to inspect progress, provide missing context, and authorize consequential steps.
6. **Local multimodal assistance.** A user can ask the configured local stack to inspect one current camera view, accept a spoken follow-up, explain what it observed, and speak the response without treating a detection as proof or permission to act.
7. **Bounded embodied work.** A compatible robot can receive an explicitly authorized movement or manipulation command through calibrated limits and an independent safety lifecycle, while observation and actuation remain separate and auditable.

9 Evaluation Agenda and Research Questions

The paper presents a software concept and implementation posture, not a claim of universal performance. Its strongest evaluation questions are therefore interactional and operational as well as model-centric. A useful study would compare a conventional desktop-only coding session with a *Telethryve* session in which a participant may follow an ongoing task from a mobile device. The primary outcome is not whether the agent produces more text. It is whether the participant can regain context, detect a misunderstanding, supply a needed constraint, and verify a result with less avoidable delay.

One measurement family concerns **continuity**. A study can record time from notification to informed response, time to resume a project after an interruption, and the fraction of questions resolved without restarting a task. Another concerns **calibrated trust**: whether users correctly identify the selected provider, whether they understand when a task is local or hosted, and whether they distinguish a model statement from a verified outcome. A third concerns **control**: the rate at which users successfully inspect, correct, approve, stop, or defer a proposed action. Multimodal studies should additionally measure camera-observation freshness and accuracy, speech transcription and playback latency, computer-action verification, robot command rejection under unsafe state, emergency-stop behavior, and recovery from uncertain physical outcomes. Terminal and bridge studies should also measure whether participants can correctly identify the active backend and run, distinguish a tool receipt from an externally verified re-

sult, and recover without misrouting a clarification when several runs are active.

These measurements matter because many apparent agent failures are coordination failures. A request may be incomplete, a model may need a decision that the user has not yet seen, or an otherwise correct change may lack verification. Mobile access can be valuable precisely because it shortens the path to human context without implying that every decision should be made on a phone.

9.1 Local and hosted comparison

Local and hosted modes should be evaluated according to task fit rather than a single leaderboard. Relevant dimensions include response latency, energy and hardware requirements, availability without network access, ease of retaining context locally, coding accuracy, and the need for fresh outside information. For a local model, a study should report the model configuration, hardware class, task set, and success criteria. For a hosted provider, it should report the relevant product surface and account conditions without conflating them with the behavior of another provider.

A fair comparison also separates model quality from workflow quality. For example, a local profile may be valuable for private preparation and local retrieval even when a hosted coding agent is preferable for a difficult repair. Conversely, a hosted model may provide strong reasoning while a local, deterministic helper gives clearer evidence for a workstation-bound result. The design question is how to make those choices understandable and reversible for the user.

9.2 Failure reporting

An academically useful evaluation should record failure states rather than collapse them into a generic unsuccessful result. A provider may be unavailable; a user may not have supplied permission; a task may require clarification; a model response may be incomplete; a camera or microphone may be unavailable; a desktop action may be denied; a robot may be disarmed or stopped; or verification may fail. These states have different remediation paths and different implications for trust. Reporting them distinctly supports the broader principle that a conversational interface should communicate uncertainty and state, not merely produce confident prose.

10 Discussion: A Human-Centered Systems Perspective

The systems perspective behind *Telethryve* differs from an application that simply embeds a chat box beside a code editor. The core unit is a human-directed work episode: an objective, selected context, available capabilities, intermediate evidence, and a conclusion that the human can understand. A conversation is valuable when it preserves those elements across time and devices.

This perspective has implications for product design. First, a provider picker is not merely a preference control. It communicates a material choice about where inference occurs, what tool semantics may be available, and what account relationship governs the work. Second, a mobile notification is not enough; the recipient needs a compact but intelligible representation of the question, status, or artifact. Third, a plug-in should announce its consequence and required authority. A phone-oriented plug-in, for example, carries different social and consent implications from a local summarization tool.

The same reasoning applies inside the multimodal local stack. Seeing is not knowing, hearing is not authentication, fluent speech is not evidence, computer access is not blanket authorization, and a motion-capable adapter is not a general autonomy license. The architecture is strongest when those distinctions remain visible in both interfaces and receipts.

The perspective also resists two unhelpful extremes. One is the premise that all intelligence must remain entirely local regardless of task requirements. The other is the premise that every workflow should silently delegate to a remote service. *Telethryve* instead treats locality and hosted capability as choices that should be legible to the person responsible for the work. That choice-oriented posture can support individual developers, small teams, and organizations with different privacy, cost, and reliability needs.

Finally, human-centered design requires that stopping and reconsidering are first-class outcomes. A well-designed agentic workflow can end in a proposed plan, a request for clarification, a verification failure, or a decision to defer. Treating only completed automation as success would obscure the valuable role of the human operator: setting goals, exercising judgment, and remaining accountable for consequential outcomes.

11 Privacy, Safety, and Limitations

The public design is guided by the proposition that authority should be commensurate with consequence. Low-risk assistance, such as drafting or summarization, differs materially from actions that could affect files, external accounts, communications, or another person. *Telethryve* therefore emphasizes user selection, scoped permissions, approval where appropriate, and observable outcomes. These ideas are consonant with the NIST AI Risk Management Framework's emphasis on governance and contextual risk management [8].

Several limitations are intentional:

- No provider route should be understood as a guarantee of correctness, availability, security, or fitness for a particular professional use.
- A local language model has hardware, latency, and capability limits; it is not presumed to equal a hosted frontier model or coding agent.
- A fluent model response is not evidence that a change, test, delivery, or external action succeeded. Important outcomes require direct inspection or verification.
- Camera detections and speech recognition can be incomplete or wrong. Sensor evidence should be time-bounded, attributed to its configured source, and interpreted conservatively.
- Computer and application control depends on operating-system permissions, tool availability, and the current state of the target application. Broad access should be enabled only for trusted, explicit workflows.
- A local coding terminal's loopback transport and tool catalog do not change the coding agent's approval responsibilities. A visible tool or receipt does not independently establish that an external side effect occurred.
- A software robot adapter is not a physical safety system. Real movement requires qualified hardware integration, calibration, independent firmware limits, collision protection, motor interlocks, watchdogs, and a physical emergency stop.
- Optional plug-ins may require configuration, device pairing, consent, permissions, or service availability. Their presence in a conceptual paper is not evidence that they are enabled in a particular installation.
- The system is not a substitute for security review, legal advice, clinical judgment, or human responsibility for consequential decisions.

The privacy posture is practical rather than absolute. Users can select local-first modes, retain context locally, constrain networked behavior, and use customer-owned locations for desired persistence. Hosted providers, when chosen, operate under their own terms and data practices. Users and organizations should evaluate those arrangements before placing sensitive information into a hosted workflow.

12 Conclusion

Telethryve proposes a way to make modern coding agents and language models more useful in real working life: retain the desktop as a serious technical workspace, extend supervision and conversation to the phone, permit local and hosted work modes, and give the configured local stack bounded ways to see, hear, speak, use the computer, and connect to a physical environment. The value lies not in claiming a universal model or concealed autonomy layer. It lies in joining perception, governed context, communication, and action while keeping the authority for each visible and reviewable.

Future evaluation should measure more than response quality. Useful metrics include time to regain context after interruption, rate of successful human intervention, clarity of provider and permission state, local-versus-hosted data choices, accessibility outcomes, and the quality of verification before consequential actions. These questions position *Telethryve* as a research and product direction for human-centered, provider-flexible software work.

References

[1] S. Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations*, 2023. https://arxiv.org/abs/2210.03629

[2] Gemma Team. Gemma 3 Technical Report. Google DeepMind, 2025. arXiv:2503.19786

[3] Ollama. Ollama product documentation, accessed August 2026. https://ollama.com/

[4] OpenAI. Codex developer documentation, accessed August 2026. https://developers.openai.com/codex/

[5] Anthropic. Claude Code documentation, accessed August 2026. https://docs.anthropic.com/en/docs/claude-code

[6] xAI. Grok documentation, accessed August 2026. https://docs.x.ai/grok/overview

[7] S. Amershi et al. Guidelines for Human-AI Interaction. *CHI Conference on Human Factors in Computing Systems*, 2019. https://doi.org/10.1145/3290605.3300233

[8] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, 2023. https://doi.org/10.6028/NIST.AI.100-1

[9] D. Driess et al. PaLM-E: An Embodied Multimodal Language Model. *Proceedings of the 40th International Conference on Machine Learning*, 2023. https://arxiv.org/abs/2303.03378

[10] A. Brohan et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. *Conference on Robot Learning*, 2023. https://arxiv.org/abs/2307.15818