

Perslis.AAC: User-Controlled Conversation for Augmentative and Alternative Communication

A technical white paper on the Tinkyspeak runtime and its model adapters

Perslis Research

Evidence reviewed: 6 October 2026 • Runtime version: 0.8.0

Abstract

Generative augmentative and alternative communication (AAC) systems can propose useful utterances, but a plausible suggestion is neither the user's intention nor evidence that communication occurred. Perslis.AAC addresses this distinction through the Tinkyspeak conversation runtime: a reusable state machine that separates partner input, model proposals, explicit selection or composition, delivery requests, and host-confirmed delivery. Only partner words and successfully displayed or spoken user words enter conversational history. Applications retain responsibility for access methods, artwork, microphones, and output devices. The runtime supports authored boards, a local English-only child model, replaceable text and translation providers, and optional image analysis. Communication profiles filter complete proposals rather than shortening user meaning. A separate hosted implementation uses Gemini 2.5 Flash through Vercel AI Gateway, session-owned PostgreSQL state, revisions, and bounded request replay protection. We describe these components and distinguish their evidence: historical offline integration checks, current automated tests, recorded model failures, and narrow hosted developer smoke checks. A fresh test run identifies unresolved vision-review contract failures; historical passing records therefore do not establish current suite health. No AAC-user study or clinical outcome evaluation is reported. The contribution is an inspectable implementation boundary for user-directed communication, together with a candid account of what structural checks establish and what still requires semantic, accessibility, and user evaluation.

Keywords: augmentative and alternative communication; user agency; conversation runtime; language models; multimodal interfaces; delivery acknowledgment.

1. Introduction

AAC communication involves more than producing grammatically plausible text. A person may need to answer a partner, decline an offer, ask for clarification, change a topic, or express uncertainty. A model that offers fluent replies can still exclude the intended answer, confuse a label with its sentence, or introduce a personal experience the person never supplied. These errors are especially consequential when the interface presents model-generated words as candidate speech in the person's own voice.

Conversational generation is an established AAC research direction. Dempster, Alm, and Reiter developed a prototype that generated conversational utterances from a domain knowledge base; later work examines personal narratives, conversational context, and AI-assisted expression. The present work does not claim to

originate conversational AAC. Its question is narrower: **what should an application record as the user's contribution when generation, selection, translation, and delivery are separate operations?** [Dempster et al., 2010](#).

Perslis.AAC is the application-facing accessibility system discussed here; TinkySpeak is its reusable conversation runtime. TinkyMind is a particular local model adapter. These names designate different layers, not interchangeable capability claims. The installed child checkpoint is English-only. Optional multilingual and image providers have their own coverage and failure modes. The public website is a networked deployment of the runtime using a remote model; it is not a demonstration of that child checkpoint running offline.

The central design is a **proposal-to-delivery boundary**. A provider may propose data, but cannot assign authoritative runtime identities, commit a user turn, or report completed speech. The host application invokes selection or composition, requests delivery, and returns an outcome. This boundary makes a consequential distinction explicit in code and observable state, although it cannot establish that a user understood every option or that a listener heard an output.

The paper makes four implementation-grounded contributions:

1. A reusable conversation state machine with separate identities for proposals, drafts, and delivery requests, and a restricted path for committing user history [S1].
2. A provider-independent integration contract that leaves access methods and output ownership with the application, while supporting explicit profiles, bilateral language flow, and visual conversation [S2–S5].
3. A concrete hosted adapter that persists owned sessions and checks revisions and repeated requests without treating generated text as completed communication [S6–S7].
4. An evidence account that separates protocol behavior, model quality, historical checks, current failures, and prospective user evaluation [E1–E5].

The intended audience is AAC application developers and researchers assessing an integration architecture. This is an engineering white paper, not a clinical efficacy report or a claim of suitability for every AAC user.

2. Related work and positioning

Dempster et al.'s prototype places utterance generation within a constrained domain and a user-centered development process. That work establishes a relevant precedent for offering complete conversational material rather than requiring every utterance to be constructed from scratch. TinkySpeak instead exposes a replaceable provider contract and a delivery state machine; this difference concerns integration, not demonstrated superiority in conversation quality. [Dempster et al., 2010](#).

Pal et al. combine fine-tuning and retrieval from user-authored content to support personal narratives. Their work makes personal source material part of generation. TinkySpeak's explicit context and delivered session history are more limited: they are inputs to an adapter, not a persistent personal knowledge base or evidence of individualized training. Reported results from that study are not results for Perslis.AAC. [Pal et al., 2024](#).

COMPA uses live conversation context to support common ground between AAC users and conversation partners. Its interface associates contributions with their intended context and exposes conversation activity

to partners. TinkySpeak supplies a host-neutral state contract rather than that particular group-conversation interface. It does not implement COMPA's complete interaction design or establish a benefit from substituting its own board. [Valencia et al., 2024](#).

SPICA is another important comparison: its framework organizes user-relevant information and uses persistent contextual memory for personalized conversation. TinkySpeak's bounded history and host-supplied context do not provide equivalent persistent personalization. A personal-memory provider could be connected, but its governance and accuracy would require separate evidence. No claim that AAC lacks conversation or personalization infrastructure is necessary to motivate this runtime. [Pal et al., 2026](#).

Explicit confirmation also has a cost. Weinberg et al. report that participants in a study of timely humorous AAC comments were willing to exchange some agency for efficiency. This finding challenges any assumption that an extra confirmation step is universally preferable. TinkySpeak exposes separate stages so a host can choose its interaction policy; whether that policy serves a person in a specific context remains an empirical question. [Weinberg et al., 2025](#).

3. System boundaries and architecture

The runtime receives final text transcripts; it does not acquire microphone input or infer that a visible person is addressing the user. A host can explicitly mark whether a response is needed. An optional turn-policy adapter can inform generation when the host permits it, but an explicit host decision takes precedence. The default flow offers choices for an addressed partner turn [S1].

Figure 1 separates model work from authoritative state transitions. The provider can return candidate sentences, labels, emoji, and optional artwork descriptors. The runtime validates and normalizes these candidates, assigns their identities, and exposes them to the host. The host renders the interface and reports the user's action. A selected candidate becomes a draft; a draft is not yet a user-history turn.

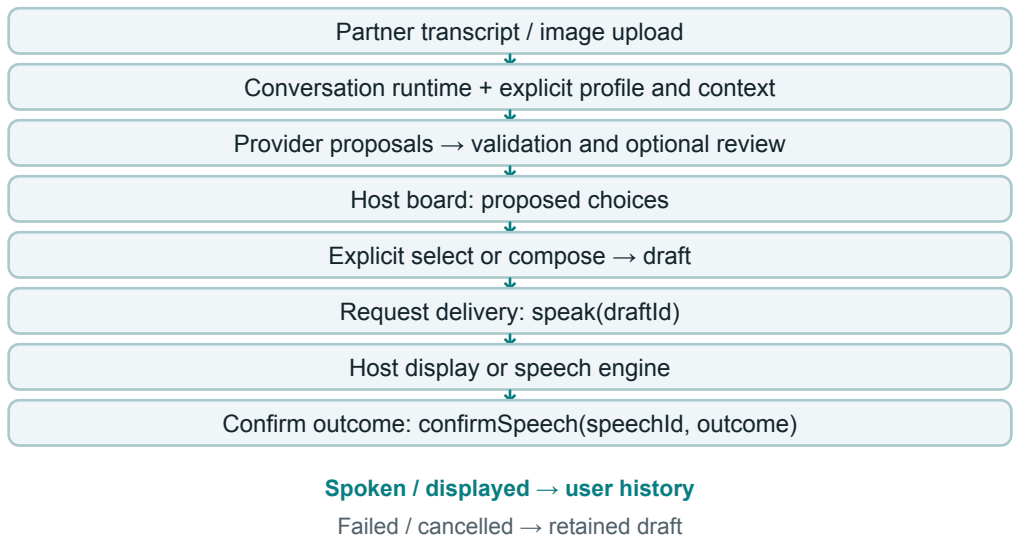


Figure 1. Provider outputs propose communication. Host-confirmed successful delivery is the entry point for user contributions to history. A host may display words instead of producing audio; these outcomes remain distinguishable.

The reusable Node package has no npm dependencies declared in its current package manifest and requires Node 22 or later. This statement excludes optional inference dependencies: the local model adapter needs Python, NumPy, ONNX Runtime, and separately installed weights. The hosted service additionally uses PostgreSQL connectivity. The package is currently private and marked `UNLICENSED`; the existence of integration code does not imply an available public npm release [S2].

Applications can embed `createTinkySpeak()` directly or run the local JSON HTTP server and use its client contract. The REPL is a developer console using the same state machine, not the only product surface. The host owns board layout, eye-gaze or scanning controls, symbol assets, camera permission, microphone processing, and speech hardware. Optional speaker adapters include a macOS adapter; the default conversation has no speaker, and the HTTP server does not emit audio itself [S2].

4. User control and conversation semantics

4.1 States, identities, and history

The runtime has seven phases: listening, generating, choosing, drafted, translating, speaking, and closed. Generation and translation carry cancellation signals and an internal epoch. A newer operation invalidates older work, preventing a late response from replacing current proposals. Choices receive runtime-assigned IDs and generation IDs; drafts and speech requests have separate IDs. Selection requires a choice from the current set, and delivery requires the current draft [S1].

The history rule is intentionally restricted. Partner input is appended by `hear()`. A user contribution is appended by `confirmSpeech()` only for `spoken` or `displayed`. A generated suggestion, rejected set, selected but undelivered draft, failed output, or cancelled output does not enter user history. Translation can annotate a recorded turn, but does not create a new user contribution. Successful delivery clears the active draft and choices. Failure or cancellation retains a draft for an explicit retry [S1].

The default history holds 40 turns, with configurable limits between 2 and 200. Omitted-turn accounting records eviction; it does not preserve an unlimited transcript. `reset()` clears conversation and visual memory while preserving the configured provider, language pair, and profile. These are session operations. They do not retrain weights, establish permanent personal memory, or authorize a model to invent biography.

This is a state-transition invariant, not a proof of intention. The runtime trusts the host to invoke actions because of a user's interaction and to report delivery honestly. A defective host could select a tile automatically or report speech that did not occur. The architecture makes that responsibility identifiable; it does not remove it.

4.2 Communication profiles and artwork

Profiles specify age, reply level, display preference, choice count, and vocabulary preference. Choice counts range from 1 to 24. Reply levels impose maximum word counts of 1, 4, 15, or 40 for word, phrase, sentence, or full modes. Language-aware segmentation counts word-like units. Over-limit generated sentences are discarded as whole proposals rather than truncated. The choice-count cap can reduce the number of proposals shown; it does not shorten a sentence [S3].

Explicit composition and edited selections remain available. Their words are user-authored and are not subjected to the generated-proposal length filter. If no proposal satisfies a profile, generation fails explicitly

and the application can offer composition or another model. A profile describes communication preferences, not intelligence, diagnosis, or a younger mental age. Age-sensitive wording in a prompt is a request to a provider, not clinical adaptation.

Each choice retains its complete sentence alongside its label and artwork metadata. A symbol-library identifier designates an asset; the runtime choice ID designates the proposed utterance. Keeping those identities separate prevents a picture from silently becoming an authoritative sentence. Custom descriptors can reference family photos, images, or host-defined symbol libraries. The runtime does not fetch or render them. Emoji fallback and valid metadata are insufficient to establish that a symbol is understandable or distinguishable to its intended user [S2–S3].

4.3 Language flow

The user and partner can have different language settings. Incoming partner text can receive a translation into the user's language; a selected outgoing draft can be translated into the partner's language. The original user words remain represented in the delivery request and history. When both languages match, native conversation need not pass through English [S1, S4].

Language metadata distinguishes known restrictions, documented lists, and unknown coverage. Discovery of a model name is not a language evaluation. The local child model cannot acquire Spanish, Mandarin, or vision support by selecting a different interface language. A provider's claimed coverage does not demonstrate correct negation, natural wording, cultural appropriateness, or compatible speech voices. These concerns require evaluation of the actual provider, language pair, and output device.

5. Model adapters and semantic review

5.1 Local TinkyMind child model

The local child adapter launches a Python worker that loads a model pack containing an ONNX graph and tokenizer. The engine uses explicit CPU inference and six branch tokens to generate alternative slots through greedy decoding. Its verified configuration uses a maximum sequence length of 256, a 20-token generation bound, and two prior exchanges. The input budget reserves space for structural and output tokens; older exchanges can be dropped, while an oversized latest partner turn fails rather than being silently clipped [S4].

Incomplete slots, reserved-token outputs, and duplicate replies are filtered. Six branches are therefore a ceiling on proposed slots, not a promise of six useful choices. The adapter checks English language support and rejects adult age profiles for the child checkpoint. A shorter response level cannot change a model's training audience.

A recorded 1 October installed-CLI check completed under a macOS network-denial policy. That record supports the narrow claim that the exercised child-text path ran without network access; it does not establish offline operation of every adapter. The same accepted transcript contains irrelevant pizza proposals after a drink question. Offline execution and useful communication are separate properties [E2].

5.2 Optional text providers and review

The runtime supports authored boards, the local child adapter, native Gemini, Ollama, compatible completion endpoints, and custom generation adapters. Authored tiles provide deterministic vocabulary without contextual model inference. Remote providers send supplied context and conversation material to their configured endpoint. A loopback address is not proof that a gateway performs inference locally [S2, S4].

For reviewed completion and vision configurations, a model review stage checks each proposal's source and label fidelity. Personal choices and open questions need not be established as external facts: a possible answer such as “No” remains available without evidence that it is already the user's preference. Added claims about an event or scene require supplied supporting material. A repair pass can revise rejected, unselected proposals, followed by another review. It cannot revise committed history or explicitly authored words [S5].

The reviewer is another model operation, often using the same selected model. Its verdict is not independent factual proof. Requiring an exact quote from supplied text is a checkable constraint; deciding whether that quote actually supports an implication is still a semantic task. Correlated generator and reviewer errors, prompt injection, and incomplete evidence remain possible. The runtime can reject an invalid review contract, yet an accepted review can still be wrong.

The adult/senior checkpoint is withheld from release as a finished model. Its 30 September assessment compared three checkpoints using 24 fresh probe conversations and 12 saved examples per checkpoint, with six slots each: 648 generated slots. The report records failures in refusal continuity, stated alternatives, names, times, and unsupported personal history. It also reports that 10,429 of 10,478 evaluation rows share a source conversation with training, including separately counted exact-input and input-plus-target duplicates. This is a dataset-split warning, not an accuracy rate. The assessment calls for repairing provenance and splitting before further training [E3].

6. Visual conversation

Image analysis is configured independently of a text-only model. An object, menu, or scene scan produces proposed observations and communication choices. The reusable scan contract supports bounded uploads, a catalog of categories and items, object descriptions, optional normalized rectangles, relations, and source-image indices. Menu prices are display metadata; selecting an ordering sentence does not place an order or authorize payment [S5].

The built-in vision path analyzes source images and merges validated results. Catalog pagination creates fresh selectable IDs. Profile filtering can exclude catalog sentences while retaining the full catalog for later profile changes. Pagination is a view of a stored result, not a fresh recognition operation.

After a scan, follow-up generation uses saved observations and delivered conversation history through the image provider. It sends no new image. Consequently, “Do you need help?” after selecting words about shoes can use the previous observations, but the system cannot know that the shoes moved or were removed. A new scan is required when that distinction matters. Clearing vision returns conversation to the text path; resetting clears both forms of memory [S1, S5].

Session snapshots retain image descriptors, hashes, and observations rather than image bytes. Hashes identify a submitted representation; they do not prove that observations are true. Remote model processing

still receives uploaded image content during a request, and host storage and provider retention policies remain separate responsibilities. The website resizes supported user uploads before sending them and imposes a smaller request bound than the reusable runtime [S5–S7].

Historical tests expose the limits of structural checks. Recorded failures include unsupported event details, labels that do not faithfully summarize a sentence, ambiguous cup symbols, and a color description attributed to the wrong object. Some local-model fixture runs pass selected checks while failing others; there is no single comprehensive passing vision benchmark established by those records. They motivate tests of object identity, refusal, label fidelity, and scene uncertainty rather than a claim of solved visual grounding [E1].

7. Hosted implementation

The public accessibility demo uses a vendored Tinkyspeak 0.8.0 runtime. Its server constructs actual `Conversation`, `CompletionProvider`, and `VisionProvider` instances with Gemini 2.5 Flash through Vercel AI Gateway and review enabled. The website's dynamic choices are model proposals, not recorded response playback. Its current creation interface offers English, Spanish, Hindi, Chinese, and Russian selections; the smoke evidence discussed below verifies only selected English and Spanish cases [S6–S7, E5].

Anonymous sessions are associated with a random browser cookie whose hash is stored with server-owned PostgreSQL state. Cookie controls include `HttpOnly` and `SameSite Strict`, plus `Secure` in production. Actions load an unexpired owned row under a transaction lock. A revision mismatch returns current owned state and a conflict, while another owner cannot load the session. This is browser-session isolation, not identification of a person or a complete multi-tenant account system [S6].

The host also stores the latest request identifier and outcome with each session. An exact repeat of that latest request returns the recorded result rather than applying another mutation. This bounded replay behavior helps with a lost response; it is not a durable log of every request or exactly-once delivery across arbitrary failures. Records are neither signed receipts nor externally anchored proofs. Restored snapshots come from trusted server storage, not browser-supplied state [S6].

Sessions become ineligible after 30 minutes of inactivity. Expired rows are cleaned up during admission; this expiry is not a guarantee of physical deletion at precisely 30 minutes or deletion from database backups. Admission limits and bounded upstream retries reduce some resource pressure, but do not establish a load-tested availability guarantee. Provider outages can still leave the user without generated choices.

The browser queues actions per session. Hero input, chat examples, tablet controls, and the larger demonstration page call the same service. Explicitly selected words can be committed as displayed communication, and subsequent partner input uses that history. Audio requests use native browser speech synthesis; successful completion events cause `spoken` confirmation, while errors or cancellation preserve an undelivered draft. Replaying an already displayed reply does not append another user turn [S7].

A browser speech-completion event establishes the host's observed output lifecycle, not audibility to a listener, voice appropriateness, or successful communication. Volume, routing, missing voices, assistive-device behavior, and user comprehension remain outside that acknowledgment. The hosted adapter includes gateway-specific schema normalization and a personal-choice quote sentinel normalized before validation. These changes preserve the intended rule but mean the vendored implementation and local package must be evaluated separately [S6].

8. Integration example

The following embedded example uses the actual API. It chooses the installed local child provider; replace it with an evaluated provider appropriate to the user and deployment. `renderBoard`, `onUserChoice`, and `deliverWithHost` are application functions, not runtime exports. This is illustrative integration code, not a participant result.

```
import { createTinkySpeak } from 'tinkyspeak-runtime';

const session = createTinkySpeak({
  provider: { kind: 'tinkymind' },
  language: 'en', partnerLanguage: 'en',
  profile: { age: 7, level: 'sentence', choices: 6 },
});

const offered = await session.hear({ text: 'Water or juice?' });
renderBoard(offered.choices);

onUserChoice(async choiceId => {
  const selected = session.select({ choiceId });
  const pending = await session.speak({ draftId: selected.draft.id });
  let outcome;
  try {
    // Return spoken/displayed only after that delivery completes.
    outcome = await deliverWithHost(pending.speech);
  } catch {
    outcome = 'failed';
  }
  session.confirmSpeech({ speechId: pending.speech.id, outcome });
});
```

An application should serialize these handlers, expose cancellation, validate the host's outcome vocabulary, and retain manual composition when generation fails. Separate language settings can introduce an outgoing translation before delivery; the interface should provide a suitable preview. The local HTTP API offers analogous operations for web, Swift, and Python clients. Its loopback default and optional bearer token are useful deployment primitives; account authorization remains the integrating host's responsibility [S1–S2].

9. Engineering evidence and reproducibility

Table 1 reports evidence by implementation, date, and denominator. All system-specific evaluations here are developer checks or stored developer assessments. No entry represents a study with AAC users. Historical model transcripts are not relabeled as current executions, and narrow assertions are not converted into a semantic accuracy percentage.

Evidence	Date and denominator	Observed status	What it supports
Historical runtime audit [E1]	1 Oct 2026; 99 automated tests, five named local harness stages	Record reports passing core harness; model-quality findings remain open	Historical integration and packaging behavior

Evidence	Date and denominator	Observed status	What it supports
Installed child-text path [E2]	1 Oct 2026; one scripted CLI sequence under network denial	Recorded success; irrelevant replies also visible	Offline execution of the exercised text path, not general response quality
Adult checkpoint assessment [E3]	30 Sep 2026; three checkpoints × 36 conversations/examples × six slots = 648 slots	Withheld; qualitative failures and conversation-level split overlap	Diagnostic evidence against release readiness
Current local package tests [E4]	6 Oct 2026; 105 tests: 100 pass, five fail, zero skipped	Failing; five vision contract tests fail in a rerun outside the sandbox	Current defects requiring follow-up, not a passing release gate
Hosted adapter checks [E5]	6 Oct 2026 invariant check; preceding deployment's selected live English/Spanish/photo sequences	Invariant check passes; selected live paths passed, with rate-limit failure and separate photo recheck	Exercised persistence, revisions, ownership, profiles, and model transport

Table 1. Test cases, generated slots, and scripted sequences are different denominators. None estimates a population outcome. The hosted photo recheck is a separate execution, not evidence of a single uninterrupted all-case production pass.

The hosted smoke script checks an affirmative and negative game-reply option, explicit selection and displayed acknowledgment, retained follow-up history, stale identities and revisions, wrong ownership, cross-origin rejection, a one-word profile, a Spanish alternative, an apple-image scan, absent image bytes in returned state, visual follow-up, and reset. An apple string in serialized observations is a narrow fixture assertion, not object-recognition accuracy. Selected follow-up history demonstrates persistence of delivered words, not faithful reasoning about every earlier preference [E5].

The definitive current package run exited with failure after 2,281.920542 ms. Its five failed tests concern visual follow-up review, image observations, menu reading, and Qwen object/automatic scans. An initial sandbox run also had localhost-permission failures; those disappeared in the authorized rerun. The remaining failures cannot be dismissed as sandbox restrictions [E4].

Reproduction should first identify which layer is being tested. Run the package's automated suite for local runtime contracts; inspect the full output instead of accepting an old summary. Model-dependent checks require the original model pack or configured endpoint and credentials. For the hosted adapter, run its invariant check separately from the network smoke script. Live model responses can vary, and requests may be rejected by quota or review. Keep raw failures alongside successes.

The accompanying claims ledger maps assertions to source files and evaluation records. Hashes identify inspected files, not an authenticated release. Prior live smoke outcomes are assistant-observed executions, without raw logs preserved in this paper's directory. No latency, throughput, comprehensive language-coverage, clinical-benefit, or user-preference result is inferred.

10. Limitations and threats to validity

The strongest evidence is for explicit state transitions. Semantic relevance, negation preservation, trustworthy symbols, appropriate choices, and truthful visual observations remain weaker and model-dependent. A valid JSON response can be harmful or misleading. A model reviewer can fail in agreement with a generator. Strict rejection can preserve an invariant while making the interface too slow or leaving no useful suggestions.

The child checkpoint's constrained tokenizer, bounded context, and short generation budget limit names, numbers, unfamiliar topics, and long conversations. Six distinct strings need not express six distinct intents. The adult assessment's split overlap prevents treating its saved evaluation loss as unseen-conversation evidence. Audience labels and age prompts do not establish therapeutic suitability.

Hosted observations do not transfer to offline privacy, and historical local vision results do not transfer to the deployed Gemini provider. Five exposed language selections do not constitute five validated language modes. Current local vision-review test failures also prevent claiming that the package is ready solely because the hosted integration works. These implementations share a design but have different transports, persistence, schema adaptations, and tests.

No controlled usability study, independent clinical review, measured communication-rate benefit, or longitudinal adoption result is available for this system. Development and documentation use automated coding and writing assistance; this paper's source audit is not independent human validation. The UI is an integration surface, not a demonstrated fit for every motor, visual, linguistic, or cognitive access requirement. The history boundary also cannot prevent an incorrectly implemented host from misreporting actions.

11. User evaluation and future work

A first user evaluation should be participatory and task-specific. Recruit AAC users with varied access methods and communication preferences, define supported tasks with them, and include conversation partners where appropriate. Obtain accessible consent and permit withdrawal, pauses, and use of existing communication systems. Determine sample size through feasibility and a prespecified study question rather than selecting a number to imply representativeness.

Compare an authored baseline, model proposals with explicit selection, and participant-controlled alternatives to delivery confirmation. Use counterbalanced tasks covering refusal, uncertainty, clarification, topic changes, personal narrative, and visual ordering. Measure time to a successfully understood contribution alongside effort, correction, abandonment, perceived authorship, and preference. Separate provider delay from navigation and composition time. Do not make speed the sole outcome.

Semantic review should include fluent reviewers for each studied language and accessible review by participants of what labels and symbols mean to them. Score unsupported external details, omitted alternatives, reversed negation, invented biography, ambiguous symbols, and inappropriate audience framing. Preserve complete candidate sets and failure cases with consent; a chosen correct sentence can conceal poor alternatives. Evaluate actual output-device completion separately from listener comprehension.

Engineering work should repair the current vision-review test contract, add regression cases that retain failures as evidence, improve conversation-level dataset splits, and evaluate unfamiliar names and numbers. Any persistent personalization should make source provenance, correction, deletion, and consent explicit.

Stronger delivery assurance could include host audit records or listener-side checks where appropriate; signed receipts would require a new design and would still not prove understanding. None of these planned capabilities is claimed as implemented here.

12. Conclusion

Perslis.AAC supplies a concrete separation between proposing words and recording communication. The TinkySpeak runtime commits a user contribution after host-confirmed spoken or displayed delivery, preserves explicit drafts after failure, and keeps model outputs from directly acquiring action authority. Replaceable providers and host-owned interfaces allow the same contract to support authored boards, constrained offline text, multilingual generation, and visual conversation. Current evidence establishes exercised engineering behavior and identifies unresolved defects; it does not establish clinical effectiveness or universally correct language. The next evaluation must test whether these boundaries help AAC users communicate on their own terms, including situations where timeliness and confirmation impose competing demands.

References

1. Martin Dempster, Norman Alm, and Ehud Reiter. 2010. *Automatic generation of conversational utterances and narrative for Augmentative and Alternative Communication: a prototype system*. SLPAT, pp. 10–18. Association for Computational Linguistics. [Canonical record and paper](#).
2. Sayantan Pal, Souvik Das, Rohini K. Srihari, Jeffery Higginbotham, and Jenna Bizovi. 2024. *Empowering AAC Users: A Systematic Integration of Personal Narratives with Conversational AI*. CustomNLP4U, pp. 12–25. DOI: 10.18653/v1/2024.customnlp4u-1.2. Author names follow the paper PDF; the Anthology metadata spells Higginbotham differently. [Paper](#).
3. Stephanie Valencia, Jessica Huynh, Emma Y. Jiang, Yufei Wu, Teresa Wan, Zixuan Zheng, Henny Admoni, Jeffrey P. Bigham, and Amy Pavel. 2024. *COMPACT: Using Conversation Context to Achieve Common Ground in AAC*. CHI. DOI: 10.1145/3613904.3642762. [Author-hosted paper](#).
4. Tobias Weinberg, Kowe Kadoma, Ricardo E. Gonzalez Penuela, Stephanie Valencia, and Thijs Roumen. 2025. *Why So Serious? Exploring Timely Humorous Comments in AAC Through AI-Powered Interfaces*. CHI. DOI: 10.1145/3706598.3714102. [Author preprint, version 4](#).
5. Sayantan Pal, Nikhil Murali, Atharva Vikas Jadhav, Jenna Bizovi, Antara Satchidanand, Manohar Golleru, Shalini Agarwal, Todd Hutchinson, Jeff Higginbotham, and Rohini K. Srihari. 2026. *SPICA: Scalable and Personalized Conversational Agent Framework for AAC Users*. IUI, pp. 218–235. DOI: 10.1145/3742413.3789116. [Publisher record](#).

Appendix A. Evidence identifiers

Paths below are relative to the inspected project roots. The accompanying `claims-and-sources.json` records absolute paths and claim mappings. They are local engineering artifacts; publication availability is not asserted.

ID	Artifact
S1	tinkyspeak-runtime/src/runtime.js; test/runtime.test.js
S2	tinkyspeak-runtime/src/api.js, src/server.js, src/client.js, src/adapters.js; README.md; package.json
S3	tinkyspeak-runtime/src/profile.js, src/symbols.js
S4	tinkyspeak-runtime/src/tinkymind.js, src/capabilities.js, src/languages.js; python/tinkymind_engine.py, python/tinkymind_worker.py
S5	tinkyspeak-runtime/src/quality.js, src/vision-provider.js, src/vision-data.js
S6	kist-site/api/aac.mjs; api/_lib/aac_runtime/runtime.js, quality.js, SOURCE.md
S7	kist-site/assets/accessibility/live.js, landing.js, demo.js
E1	tinkyspeak-runtime/evaluation/audit/report.json, automated-tests.txt, cited subordinate transcripts
E2	tinkyspeak-runtime/evaluation/child-runtime/check.json, installed-cli.txt
E3	tinkyspeak-runtime/evaluation/adultcube-v1/assessment.json, results.json
E4	Accessibility-White-Paper/runtime-tests-20261006.txt; any rerun is separately identified in the claims ledger
E5	kist-site/scripts/check_accessibility_overview.js, scripts/aac_live_smoke.mjs; Accessibility-White-Paper/verification-record.json records the current invariant execution; earlier live outcomes have separate provenance

Evidence discipline. The paper was checked against source and stored records using the saved academic white-paper writer methodology. That self-audit found no basis for clinical participant counts, therapy outcomes, comprehensive language validation, signed delivery receipts, or a blanket current-test pass. Those claims are deliberately absent.