FRI-1.414-1-MIP-03

ENABLING ENGINEERS USING ARTIFICIAL VOICE ASSISTANT BASED ON LARGE LANGUAGE MODEL[3]

Alexander Zlatanov
Faculty of Natural Sciences and Education
Department Informatics and Information Technologies
University of Ruse "Angel Kanchev"
Tel: +359 878 58 58 64
Email: aleks_zlat@abv.bg

Assist. MSc Martin Dzhurov, PhD student
Faculty of Natural Sciences and Education
Department Informatics and Information Technologies
University of Ruse "Angel Kanchev"
Tel: +359 893 97 51 07
Email: mdzhurov@uni-ruse.bg

Abstract: *This paper explores the feasibility of employing large language model (LLM), such as Llama, as virtual engineering assistant. The paper presents a practical application designed to enhance engineers' working environments through modular architecture with a core system that operates as an operating-system-level agent. The agent coordinates natural-language interaction, controlled tool invocation and automating routine engineering activities (e.g., specification drafting, troubleshooting guidance and documentation). The research advances the position that such assistant can increase engineer productivity across variety of workflows.*

Key words: *Artificial Intelligence, Large Language Models, Assistant, Engineering, Workflow Automation, Productivity, Operating-system-level Agent.*

INTRODUCTION

Engineering work is increasingly shaped by digital artifacts and complex toolchains: requirements in Application Lifecycle Management (ALM) systems, Computer-Aided Design (CAD) assemblies, simulation outputs, source code repositories, test logs, and compliance documentation. In this environment, productivity losses commonly come from context switching (moving between tools and documents), information fragmentation (knowledge distributed across PDFs, tickets, and emails), and documentation overhead (capturing rationale and evidence for decisions).

Recent progress in locally executable AI systems makes it realistic to embed an Artificial Voice Assistant (AVA) directly into the engineer's workstation environment. The technical premise is that a local AVA can combine: speech recognition for hands-free interaction, a small Large Language Model (LLM) for intent handling and drafting, retrieval from local/enterprise documents to ground answers, and speech synthesis to deliver low-friction feedback. The enabling software ecosystem for local LLM inference has matured; for example, *llama.cpp* explicitly targets "LLM inference with minimal setup" and aims for performance "on a wide range of hardware." [1]

However, engineering is a high-consequence domain. A plausible-sounding but incorrect statement can propagate into unsafe decisions, expensive rework, or noncompliance. Therefore, this paper positions the AVA as assistive rather than autonomous: the assistant drafts, summarizes, retrieves, and proposes; engineers validate, approve, and remain accountable for decisions. The objective is augmentation—reducing the time spent on routine cognitive and documentation tasks—while preserving professional judgement and safety constraints.

[3] Докладът е представен на конференция на Русенския университет на 24 октомври 2025 г. в секция „Математика, информатика и физика" и отразява резултати от работата по проект № 25-ФПНО-04, финансиран от фонд „Научни изследвания" на Русенския университет.

EXPOSITION

1) Local AI feasibility: why "run it on the engineer's machine" now works

Local execution is viable because the tooling around efficient inference has become robust and reproducible. In practice, a locally run LLM does not need to be the largest available model to be useful for engineering tasks such as drafting, summarizing, transforming formats, extracting structured fields, or guiding workflows. The critical requirement is that inference must be sufficiently fast and stable on common hardware, and that deployment must be repeatable within corporate IT constraints.

The *llama.cpp* ecosystem exemplifies this maturity. Its stated goal is to support local inference with "minimal setup," optimized across heterogeneous hardware targets. [1] That matters for engineering organizations because it reduces integration effort: internal teams can package a known-good runtime, model file(s), and configuration as a controlled tool rather than a research prototype.

From an engineering deployment perspective, "local" does not mean "isolated." A modern local AVA can be workstation-hosted while still accessing enterprise systems (PLM/ALM, document management) through authenticated connectors. The key is that sensitive content (design files, requirements baselines) can remain within the organization's network boundary, while the assistant performs inference locally or on an on-premise host.

2) Hardware readiness: practical compute for local inference in engineering environments

Local inference is no longer restricted to specialized labs. A typical engineering workstation includes multi-core CPUs, substantial RAM, and often a discrete GPU. Even without relying on datacenter hardware, these resources are sufficient for small-to-mid scale models and voice pipelines when engineered for efficiency.

A realistic strategy is to deploy the AVA in tiers – **Figure 1**:

- **Tier 1 (single workstation):** the engineer runs the AVA locally for latency-sensitive drafting and personal knowledge retrieval (local standards, project notes, recent tickets).
- **Tier 2 (team on-prem server):** the same AVA architecture runs on a local server for multi-user concurrency and larger retrieval indexes, still inside corporate infrastructure.
- **Tier 3 (hybrid allowed by policy):** optional cloud augmentation for non-sensitive tasks, while sensitive work remains local.

This tearing supports the paper's core claim: the "time has come" because software runtimes have stabilized for local inference [1], and the voice pipeline components have reliable open implementations for local deployment [2], [3].

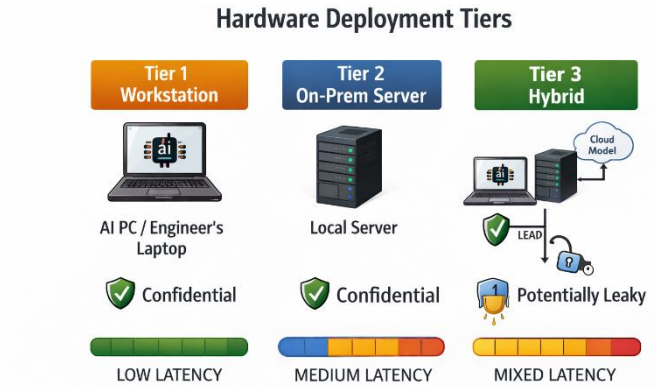

Figure 1: Hardware deployment tiers (workstation vs on-prem server vs hybrid), with latency and confidentiality trade-offs.

3) AVA reference architecture: speech <=> reasoning <=> retrieval <=> tools

A credible engineering-grade AVA is a system-of-systems. A reference architecture consists of:

1. Automatic Speech Recognition (ASR)

The AVA must transcribe domain vocabulary: part numbers, standards, and test terminology. *whisper.cpp* provides a practical path: it is a "Port of OpenAI's Whisper model in C/C++" [2] and offers a straightforward local workflow. Even its quick start is engineered for repeatability (clone, download a model, build, run). [2]

2. Local LLM runtime (core dialogue + drafting)

The local LLM is responsible for intent parsing (what the engineer wants), dialogue state (what has already been clarified), and producing structured outputs (tables, checklists, draft procedures). A key point is that the AVA should privilege structured, verifiable outputs (requirements, test steps, change request drafts) over free-form prose when risk is high.

3. Retrieval grounding (document-backed answers)

Engineering answers must be tied to authoritative sources: internal standards, drawing notes, approved test methods, and revision-controlled documents. The AVA should retrieve relevant snippets and cite them in its output. This reduces hallucination risk and improves auditability.

4. Tool execution layer (connectors and guarded actions)

To "excel the work environment," the AVA needs controlled access to tools: search tickets, open a requirement object, draft a pull request description, generate a test matrix, or produce a CAD note template. Any action that changes state (closing a ticket, modifying a baseline) must require explicit engineer confirmation.

5. Text-to-Speech (TTS)

For hands-busy environments (lab benches, manufacturing lines), speech output reduces friction. *piper* is described as "a fast, local neural text to speech system," making it suitable for on-device spoken feedback. [3]

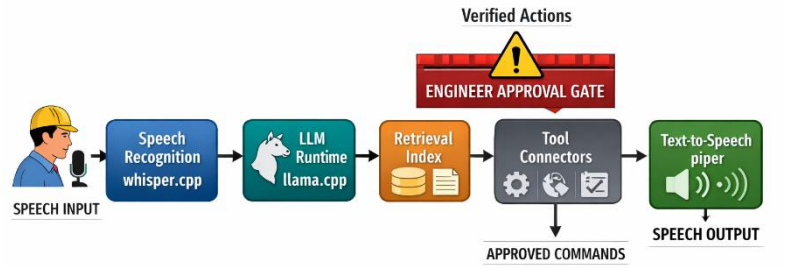

Figure 2: AVA system diagram: ASR (whisper.cpp) => LLM runtime (llama.cpp) => retrieval index => tool connectors => TTS (piper), with "engineer approval gate"

4) Six capability areas where the AVA can excel engineering work (supervised augmentation)

Below are six subpoints showing where the idea is most credible and valuable, with the AVA explicitly designed to keep engineers in control as depicted in **Figure 2**.

4.1: Requirements drafting, review support, and traceability (systems/product engineering)

Engineers spend time translating ambiguous stakeholder intent into testable requirements and maintaining links to verification. The AVA can draft requirement statements in consistent style, propose acceptance criteria, detect ambiguity, and generate review checklists. Retrieval grounding allows the AVA to cite existing standards or internal precedents rather than inventing constraints. This is especially useful during reviews: voice queries like "summarize requirement changes since last baseline" can produce structured diffs and draft meeting minutes.

4.2: CAD-related documentation and design rationale capture (mechanical design)

Even when the AVA is not directly generating geometry, it can accelerate the work around CAD: documenting design intent, producing design review agendas, summarizing change rationale, or turning

voice notes into controlled templates. Research in LLMs for CAD emphasizes that datasets and experimental setups increasingly connect natural language to CAD representations; for instance, work in this area describes building "a dataset of CAD models, sketches, and image data" for mechanical components such as "gears, shafts, and springs." [6] This direction supports the idea that language interfaces can become a practical layer around CAD workflows—initially for documentation and automation, and later for deeper integration.

4.3: Code scaffolding, test automation, and quality support (software/test/control engineering)

An AVA can draft scripts, generate test cases, propose refactors, and summarize diffs—then the engineer reviews and runs tests. This maps to observed impacts of AI-pair programming: the ACM study reports results showing AI-pair programming has "a positive impact on code quality and developer satisfaction." [4] In an engineering organization, the AVA becomes the "pair" that accelerates first drafts while engineers retain responsibility for correctness, safety constraints, and integration.

4.4: Electronic Design Automation (EDA) assistance (electronics / hardware engineering)

LLMs are increasingly studied as tools within Electronic Design Automation (EDA) workflows. A survey in ACM Computing Surveys presents "a comprehensive exploration of LLM applications in EDA." [5] Practical AVA functions here include: summarizing constraint files, explaining tool warnings, drafting verification plans, and organizing design reviews. The AVA's retrieval component is critical because EDA decisions must remain consistent with internal libraries, process rules, and project-specific constraints.

4.5: Troubleshooting, manufacturing support, and field service reporting (industrial engineering)

Voice interfaces are particularly valuable when engineers are hands-busy. The AVA can guide troubleshooting trees, retrieve relevant procedures, and draft service reports from dictation. Local deployment matters when field logs or maintenance documentation is confidential or when connectivity is unreliable. Spoken output via local TTS (e.g., "fast, local neural text to speech" [3]) supports continuous workflow without stopping to type.

4.6: Compliance documentation and audit readiness (regulated environments)

In regulated industries, compliance evidence is a deliverable, not an afterthought. The AVA can draft verification protocols, summarize test logs into standard templates, generate traceability matrices, and assemble evidence packs by retrieving approved documents. Because these artifacts are subject to audit, the AVA should produce outputs with explicit citations to retrieved sources, and it must keep logs of what documents were used to generate each draft.

5) "AI taking over engineering jobs" vs "AI reshaping engineering tasks"

The realistic question is not whether AI will replace "engineers" as a category, but which tasks inside engineering roles will be automated or accelerated. The evidence across domains suggests that AI is strongest at producing first drafts and assisting with routine transformation tasks, and weakest at accountable decision-making under safety and liability constraints.

A grounded interpretation of the software side is supported by AI-pair programming results: if AI assistance can improve productivity and quality under supervision [4], then engineering organizations can expect task redistribution. Engineers may spend less time producing boilerplate artifacts (templates, drafts, summaries) and more time on integration, judgement, safety analysis, and sign-off.

For mechanical and electronics engineering, the growing research base indicates increasing feasibility of language-driven support around CAD and EDA workflows [5], [6]. This trajectory points toward augmentation: accelerating document-heavy and tooling-heavy processes while keeping expert engineers responsible for validation, safety, and compliance.

6) Supervision, validation, and governance: making the AVA safe enough to deploy

A locally run AVA is not automatically safe; local deployment reduces data exfiltration risk but does not eliminate technical failure modes. The system must be engineered for predictable behavior and controlled outputs as shown in **Figure 3**.

Human approval gates. Any state-changing action (closing a ticket, modifying a baseline, generating an official release note) must require explicit confirmation. The AVA should default to "draft-only" outputs in high-risk workflows.

Grounding by retrieval and explicit citations. The AVA should treat internal documents as the source of truth and cite them for any numeric constraints, requirements, or procedure steps. This transforms the assistant from a speculative generator into a document-backed assistant suitable for engineering.

Reproducible build and deployment. Tooling should be packaged and versioned. Both *llama.cpp* [1] and *whisper.cpp* [2] offer engineering-friendly properties (repeatable builds, local binaries, stable model packaging approaches) that fit conventional IT governance.

Evaluation metrics. Deployment should track: time-to-first-draft, citation coverage rate, override rate (how often engineers correct the assistant), defect density in AI-generated code or documents, and latency (voice-to-response).

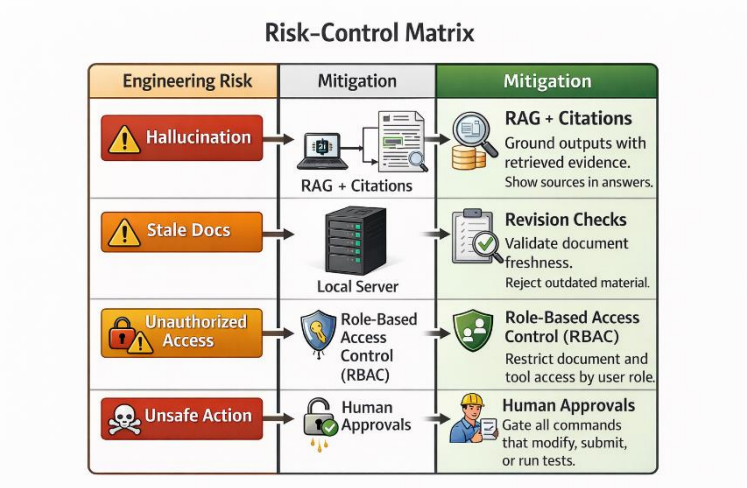

Figure 3: Risk-control matrix (hallucination, stale docs, unauthorized access, unsafe action) mapped to mitigations (RAG citations, revision checks, RBAC, approvals).

CONCLUSION

Engineering organizations now have the practical ingredients to deploy a locally executed Artificial Voice Assistant: a local LLM runtime for drafting and intent handling [1], a local ASR component for robust voice input [2], and a local TTS component for spoken feedback [3]. Meanwhile, empirical and survey literature indicates that AI assistance can improve productivity and quality in software workflows [4], and that LLM integration into EDA and CAD-related processes is becoming a structured research area with expanding datasets and application taxonomies [5], [6].

The highest-value and lowest-risk pathway is supervised augmentation. A local AVA should accelerate routine cognitive work—drafting, summarizing, retrieval, and structured documentation—while enforcing human oversight for decisions, safety-critical outputs, and state-changing tool actions. Under this design, the assistant does not replace engineers; it increases throughput, reduces context switching, improves knowledge reuse, and supports faster engineering cycles while preserving accountability.

REFERENCES

[1] Gerganov, G. (2023) llama.cpp [Computer software]. Available at: https://github.com/ggerganov/llama.cpp (Accessed: 30 September 2025).

[2] Bevenius, D. (2022) whisper.cpp [Computer software]. Available at: https://github.com/ggml-org/whisper.cpp (Accessed: 30 September 2025).

[3] Hansen, M. (2023) piper [Computer software]. Available at: https://github.com/rhasspy/piper (Accessed: 30 September 2025).

[4] Chen, T. et al. (2024) 'The impact of AI-pair programmers on code quality and developer satisfaction', Proceedings of the 2024 ACM Conference on Code. Available at: https://dl.acm.org/doi/10.1145/3665348.3665383 (Accessed: 30 September 2025).

[5] Pan, J. et al. (2025) 'A survey of research in large language models for electronic design automation', ACM Computing Surveys. Available at: https://doi.org/10.1145/3715324 (Accessed: 30 September 2025).

[6] Sun, Y. et al. (2025) 'Large Language Models for Computer-Aided Design (LLM4CAD) Fine-Tuned: Dataset and Experiments', ASME Journal of Mechanical Design, 147(4), 041710. Available at: https://doi.org/10.1115/1.4067713 (Accessed: 30 September 2025).