IADIS International Journal on Computer Science and Information Systems
Vol. 21, No. 2, pp. 126-144
ISSN: 1646-3692

LEARNED PATTERNS AND OFFLINE OPERATION IN SPANISH: A HUMAN-CENTERED, MULTI-DISTRIBUTION NATURAL LANGUAGE INTERFACE FOR GNU/LINUX

Jose Ramon Aragon Toledo and Ricardo Ruiz Rodríguez
Universidad Tecnológica de la Mixteca (UTM), Mexico

ABSTRACT

GNU/Linux underpins roughly 90 % of cloud infrastructure, yet its command-line interface excludes most non-expert users, and the exclusion falls hardest on communities whose first language is not English. This paper presents an extended account of BOTAS, a conversational interface that translates spoken Spanish into validated, executable GNU/Linux commands under Human-Centered Artificial Intelligence principles. Extending our earlier conference report, we describe a learned-pattern tier that converts each request the language model resolves into a reusable parameterised template, retrieved by TF–IDF and cosine similarity and refilled through adaptive lexical anchors; a fully offline mode; and multi-distribution support verified by unattended installation across three package managers. Four evaluations are reported. On a 250-task benchmark scored by an automated harness under a strict action-agreement criterion, the system reaches 78.0 % coverage and 87.6 % precision when it acts, with no safety false negatives; a second model with near-identical precision declines to act three times as often, making model choice an interaction-design decision. A usability study (n = 14) found voice 2.1 to 2.5 times slower than pointer and keyboard on tasks the graphical environment already handles well. A controlled within-subject comparison (n = 7) against the baseline the system actually displaces—a non-expert working out what to type—found it 2.80 times faster, a 66.6 % reduction, with every participant faster (Wilcoxon exact $p = .0156$, $d_z = 3.63$). Such interfaces do not replace direct manipulation; they substitute for the terminal where direct manipulation leaves users unserved, and their benefit scales with distance from expertise ($r = 0.97$).

KEYWORDS

Human-Centered AI, Natural Language Interface, GNU/Linux, Voice Interaction, Usability Evaluation, Digital Equity

1. INTRODUCTION

GNU/Linux powers approximately 90 % of cloud infrastructure, runs on 96.3 % of the world's top one million servers, and holds the entire supercomputing market (W3Techs, 2023; Top500.org, 2023). Despite this ubiquity, the command-line interface that unlocks its potential remains inaccessible to most users. Someone who wants to list the files in a folder larger than 20 MB must first learn that the tool is called `find`, then that size is written `-size +20M`, and then that a bare invocation without `-type f` also returns directories. A ten-second intention becomes an expedition through documentation, and novice users have been documented spending eight to fifteen minutes assembling basic commands (Norman, 2013). Jenson (2024) argues that the resulting stagnation "is not a technical limit but a design capitulation": graphical interfaces sacrifice the command composition that makes GNU/Linux valuable, manual pages assume expertise, and canned scripts cannot anticipate every need. The distance between knowing what you want to do and knowing how to tell the computer to do it remains wide, and Shneiderman and Plaisant (2010) had already proposed natural language as the bridge.

In the conference version of this work (Aragon Toledo and Ruiz Rodríguez, 2026) we presented BOTAS (*Bot de Operaciones y Tareas Automatizadas del Sistema*—System Operations and Automated Tasks Bot), a conversational interface that translates spoken or typed Spanish into validated, executable GNU/Linux commands. It embodies Human-Centered Artificial Intelligence (HCAI) principles (Shneiderman, 2020) through five commitments: amplify rather than replace human capability; explain what each command does and why; keep the user in control through pre-execution confirmation; prioritise safety with deterministic detection of dangerous operations; and build competence rather than dependence. It targets Spanish speakers—over 500 million people underserved by Anglocentric interfaces—because in Latin America linguistic and economic barriers compound one another in educational institutions (Warschauer, 2004; CEPAL, 2023). That paper described an architecture and an exploratory evaluation. Since then, the system has acquired a learning tier, a fully offline mode and verified multi-distribution support, and has been put through two studies with users who had never seen it. This article reports all of that, contributing a *learned-pattern tier* that turns each request the model resolves into a reusable parameterised template, retrieved by TF–IDF and cosine similarity and refilled through adaptive lexical anchors, so a paraphrase of a solved task never reaches the model again (Section 3.2); a *fully offline mode*, with the constraints that make a 6.7-billion-parameter model usable on a six-gigabyte consumer GPU (Sections 3.5 and 4.3); *validated multi-distribution support*, exercised by unattended installation across three package managers (Section 4.4); a *usability study* ($n = 14$) contrasting voice against the graphical environment under two model back ends, and a *controlled within-subject comparison* ($n = 7$) against the baseline the system actually displaces, a non-expert working out what to type (Sections 4.5 and 4.6); and a *re-evaluation* of command generation on an enlarged task set, reported transparently against the published figures (Section 4.1).

2. RELATED WORK

2.1 Natural Language Interfaces for the Command Line

The command line persists because of composition and scriptability, but Nielsen (1994) showed it demands 10–20 hours of training against 2–4 for graphical interfaces, a gap that has only widened as distributions ship ever more commands. NL2Bash (Lin et al., 2018) pioneered the mapping from natural language to shell commands at 36 % exact match but addressed neither execution nor safety. Large language models have since improved command generation markedly (Brown et al., 2020), yet general assistants such as ChatGPT (OpenAI 2023) remain suggestion-only: the user copies, pastes and hopes, with no validation before execution. Terminal-oriented tools such as ShellGPT (Shchekotov, 2024) add execution, but typically without a safety layer, without educational feedback, and without any memory of what the user has asked before. Reusing earlier responses for semantically equivalent requests is an established cost-reduction strategy—GPTCache (Bang, 2023) keys cached responses by embedding similarity—but a semantic cache returns a previous *answer*, whereas the tier of Section 3.2 stores a *parameterised command template* with the semantic role of each slot, so a paraphrase naming a different folder is still served locally with the new folder substituted in. Its retrieval machinery is deliberately classical (Manning et al., 2008), because it must run in pure Perl in under five milliseconds with no model and no network.

2.2 Voice Assistants for GNU/Linux

Free-software voice assistants for the desktop have a troubled history. Jasper (Saha et al., 2014) established the always-listening, offline-first pattern but predates modern language models and relied on hand-written intent modules; Mycroft, the most visible attempt, ceased operations in 2023 after a patent dispute (Dobberstein 2023), and its community continuation OpenVoiceOS (OpenVoiceOS, 2024) preserves the skill-based architecture. All share one structural constraint: capability is bounded by the skills somebody wrote in advance. BOTAS inverts that relationship—the language model supplies open-ended interpretation, while a deterministic layer, not the model, supplies the guarantees.

2.3 Human-Centered AI and Digital Equity

Shneiderman (2020, 2022) established HCAI as a framework emphasising transparency, safety and user empowerment, arguing that high automation and high human control are complementary rather than opposed; Amershi et al. (2019) derived design guidelines from it that inform several decisions below. Luger and Sellen (2016) found that absent state feedback is among the most reliable sources of frustration in conversational agents, which motivated the four-state indicator of Section 3.6. On equity, Warschauer (2004) documents how technical barriers entrench inequalities, and Wobbrock et al. (2011) proposed ability-based design, in which interfaces adapt to users rather than the reverse—a philosophy BOTAS applies by accepting ordinary spoken Spanish as the primary modality.

3. SYSTEM ARCHITECTURE

BOTAS is implemented in Perl—chosen for its Unix integration, text-processing strength and mature GTK3 bindings—across twelve modules totaling some 16 500 lines, plus an embedded Python process for the acoustic components. Figure 1 shows the pipeline: input arrives through the graphical interface, the microphone button or the background wake-word detector; it is interpreted by a three-tier pipeline that escalates only when cheaper tiers fail; the resulting command passes a deterministic safety layer; and execution is reported back in plain language and, optionally, in speech.

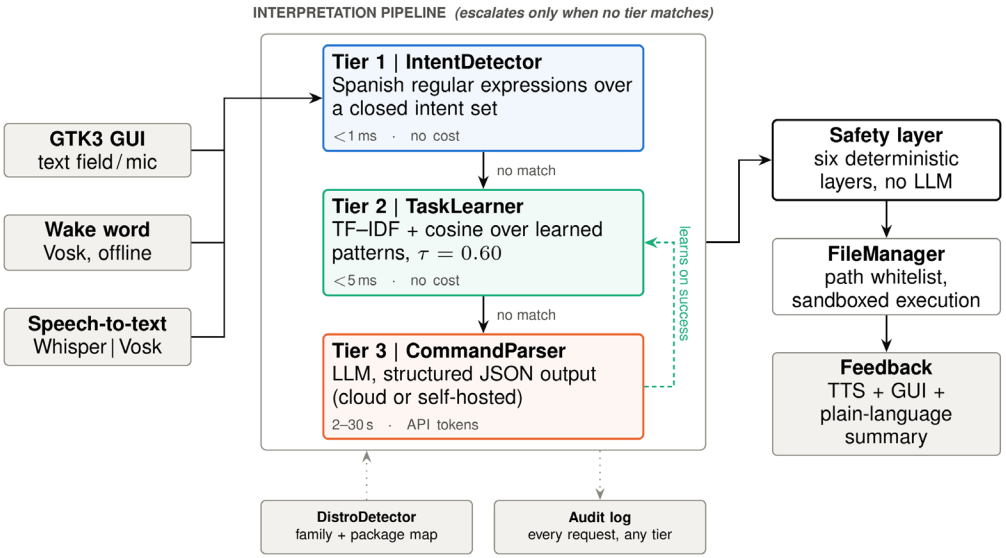


Figure 1. System architecture. The interpretation pipeline escalates from regular expressions to learned patterns to the language model, so cost is incurred only when the cheaper tiers cannot answer. The dashed return path is the learning loop: every request the model resolves successfully becomes a pattern the second tier can serve next time

3.1 The Three-Tier Interpretation Pipeline

The central design decision is that interpretation is not a single act. The orchestrator first deduplicates the transcription—Whisper occasionally repeats an entire utterance two or three times on longer recordings, inflating cost and corrupting pattern matching—then tries three mechanisms in ascending order of cost. Each tier returns the same JSON structure, so safety validation, execution and logging are indifferent to which tier produced the command. We call these *tiers* rather than layers because they are not traversed in sequence: a request stops at the first tier that resolves it, and only what none of them can answer reaches the model. *Layer* is reserved here for the safety mechanism of Section 3.3, through which every generated command does pass.

Tier 1 handles requests that need no linguistic understanding at all: the time, the date, free memory, launching an application. `IntentDetector.pm` matches curated Spanish regular

expressions and, on a hit, answers in under a millisecond at no cost. Its coverage is deliberately narrow—conversational and immediate-information intents, not command generation—and Section 4.2 quantifies how narrow.

Tier 3, described before Tier 2 because the second tier learns from it, sends the request to a large language model through a system prompt (Wei et al. 2022) returning JSON with four fields: an `action` from a closed set, a dictionary of `parameters`, a `tts_response` in plain language used for the educational summary and speech synthesis, and a `requires_confirmation` flag. One action, `clarify`, carries a follow-up question instead of a command and is how the system declines to guess. The prompt also carries the working directory, the detected distribution family and the last ten exchanges, letting the model resolve anaphora such as *"those ones"*. Three providers are selectable: the OpenAI API, the DeepSeek API—whose interface is OpenAI-compatible, so integration reduces to changing the base URL—and any self-hosted server exposing the same protocol, which is how the offline mode operates. Structured output removes the ambiguity of free text and makes everything downstream of it deterministic.

3.2 Tier 2: Learned Task Patterns

`TaskLearner.pm` (1 295 lines, 35 subroutines) is the component added since the conference version and the system's long-term memory. Its premise is that the requests a user repeats are precisely the expensive ones, and that a task the model has already solved once should not be sent to it again. Figure 2 shows its two paths.

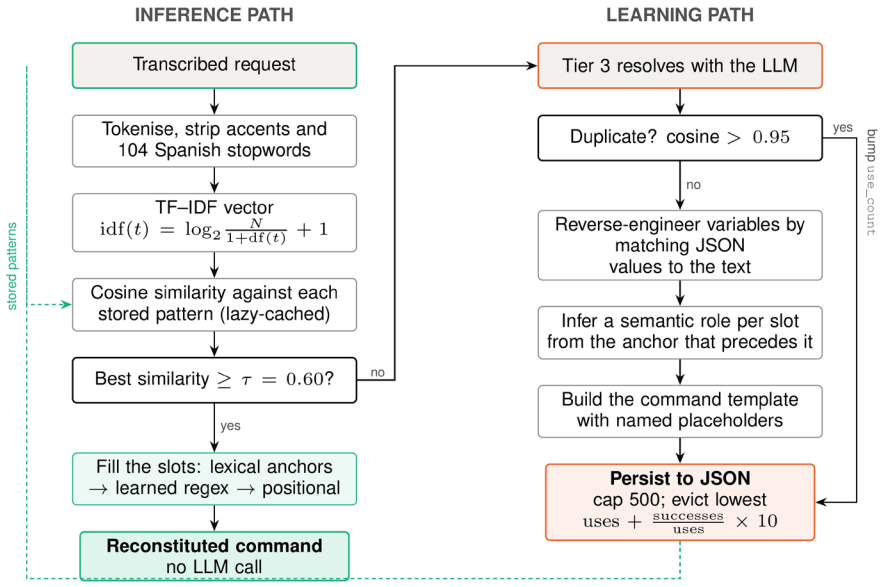


Figure 2. The two paths of the learned-pattern tier. On the left, an incoming request is matched against stored patterns and, above the confidence threshold, reconstituted into a command without any model call. On the right, a request the model has just resolved is turned into a parameterised template and persisted

3.2.1 Retrieval

The similarity engine is pure Perl with no external dependency. Input is lowercased, stripped of accents and punctuation and filtered against 104 Spanish stopwords. Terms are weighted with a smoothed inverse document frequency,

$$\mathrm{idf}(t) = \log_2\left(\frac{N}{1+\mathrm{df}(t)}\right) + 1,$$

where N is the number of stored patterns and $\mathrm{df}(t)$ the number containing term t, and the resulting sparse vector is compared against every stored pattern by cosine similarity. Pattern vectors are computed lazily and cached, the cache being invalidated whenever the pattern set changes. The best match is accepted at a configurable threshold, $\tau = 0.60$ by default, tuned empirically: lower, and near-misses produce false matches, risking the wrong command; higher, and only near-verbatim repetitions match, wasting the learning.

3.2.2 Acquisition and Template Refilling

When Tier 3 resolves a request, the orchestrator hands the pair (request, command) to `learn()`. A duplicate check discards anything above cosine 0.95, incrementing a use counter instead. Otherwise, the module performs what we call reverse engineering of variables: it searches the request for the concrete values the model placed in the JSON—a file name, a destination, a search string—and replaces each with a named placeholder, yielding a template rather than a fixed command.

The difficulty is that the next invocation will not be worded the same way. A pattern learned from *"Crea una carpeta llamada Matemáticas en Documentos"* must still serve *"Quiero que crees una carpeta llamada Backup dentro de la carpeta Documentos"*, where the wording is longer, the preposition differs and the order is not identical. A rigid regular expression fails here, and a positional heuristic captures spurious fragments such as *"la carpeta Documentos"*. The solution is to bind each slot to a *semantic role* and recognise that role by the Spanish expressions introducing it, regardless of position. The module defines six roles over 62 anchors: `location` (*dentro de la carpeta*, *en el directorio*, *en*), `name` (*que se llame*, *llamada*, *el archivo*), `new_name` (*renómbrala a*), `source` (*mueve*, *copia*), `query` (*busca sobre*, *acerca de*) and `pattern` (*que contengan*, *con extensión*). Extraction proceeds in six steps: normalisation preserving character offsets, so values keep their accents and capitalisation; exhaustive search for every anchor of every role; overlap resolution, sorting hits by ascending position and, on ties, by descending length, so *dentro de la carpeta* wins over the *dentro de* embedded within it; extraction of each value as the span between one anchor and the next; edge cleaning that strips stopwords and domain noise from the *ends* of a value but never its middle; and assignment of values to slots by role. On the paraphrase above, the anchors *llamada* and *dentro de la carpeta* are detected, the embedded *dentro de* is discarded, and the slots receive *Backup* and *Documentos*—yielding `~/Documentos/Backup` with no model call.

Anchor extraction is the first of three strategies, tried from most flexible to most rigid: failing it, the module falls back to the regular expression built when the pattern was learned, and then to a positional heuristic. If all three fail the request escalates to the model, which in turn teaches the module the new variant. Patterns are capped at 500; on overflow the lowest utility index, $\mathrm{uses} + (\mathrm{successes}/\mathrm{uses}) \times 10$, is evicted, favouring patterns that are both frequent and reliable.

3.3 Safety

Because the system executes file-system operations derived from model output, no safety guarantee may depend on the model. Six deterministic layers stand between an interpreted request and its effects. A `BEGIN` block strips loader variables that could force unauthorised libraries into the process. Every resolved path is checked against a whitelist of permitted roots, so a traversal such as `../../etc/passwd` raises an exception rather than reaching the file system. Each operation is classified as safe, warning or dangerous by rules keyed on action type and parameters—regular expressions and heuristics, never model inference, so behaviour is consistent and auditable. Warning and dangerous operations require explicit confirmation. A single instruction may affect at most one hundred files. And every interaction is logged with its timestamp, original text, detected intent, outcome and originating tier. This is defense in depth rather than a single check, and its scope should be stated plainly: it detects *known* categories of dangerous operation. It cannot anticipate a novel attack vector, and it does not reason about context—deleting a directory is classified identically whether it holds cache files or a dissertation.

3.4 Distribution Detection and Multi-Distribution Support

Fragmentation across distribution families is a well-known source of confusion: `apt install foo` fails on Fedora, which wants `dnf`, and on Arch, which wants `pacman`. `DistroDetector.pm` parses `/etc/os-release`, reading `ID`, `ID_LIKE` and `VERSION_ID`, and classifies the host into one of six families—`debian` (`apt-get`), `fedora` (`dnf`), `arch` (`pacman`), `suse` (`zypper`), `alpine` (`apk`) and `void` (`xbps-install`)—falling back to probing for package managers when that file is unavailable. A dictionary maps 12 generic dependency names, written in Debian convention, onto each family's equivalents, with `cpanm` as the common fallback when no native package exists. The detected family is cached and injected both into the model's system prompt and into the first tier's handlers, so every generated command suits the machine it will run on.

The same table drives installation. A universal installer classifies the host, translates the dependency list, installs through the native manager with `cpanm` only as a fallback, writes an initial configuration and registers a desktop entry; a companion script handles the offline wake-word model. A portable directory builds an architecture-independent `.deb`—everything is interpreted Perl and Python, so one package serves `amd64` and `arm64`—and the same machinery produces an `.rpm`.

3.5 Online and Offline Operation

BOTAS runs in two modes that swap three components between cloud and local implementations and can be switched at run time. Transcription moves from OpenAI Whisper to a local Vosk model (Alpha Cephei 2023; Radford et al. 2023); interpretation from `gpt-4o` or `deepseek-chat` to `deepseek-coder-6.7b` served locally by LM Studio; speech synthesis from the OpenAI service to eSpeak. The trade is latency and cost against privacy: 2–5 seconds and API tokens online, 5–45 seconds and nothing at all offline, where no audio or text leaves the machine. A single callback propagates the change to all three modules without

restarting. One component never moves: the wake-word detector runs on Vosk in both modes, so passive listening never generates network traffic—the system is not "always listening" to anyone else's servers.

Small local models are not slower versions of large remote ones, and the offline path adapts to them: history is truncated to the last two exchanges rather than ten; the token limit is reduced, because a small model left unconstrained will ramble to the limit; the HTTP timeout rises to 300 seconds; a shorter system prompt replaces the full one; and JSON-mode enforcement, offered by cloud providers, is replaced by heuristic cleaning that extracts the span between the first brace and the last.

3.6 Voice Pipeline and State Feedback

Hands-free operation runs as follows: the wake-word detector listens offline with latency under 200 ms; on activation a 30-second attention window opens; the utterance is transcribed and processed exactly as typed input; and the reply is spoken. Addressing Luger and Sellen (2016)'s (2016) finding that missing state feedback frustrates users, the interface carries a four-state indicator—idle, listening, recording, processing—distinguished by colour and position, and every executed command is accompanied by a plain-language statement of intent and, for flagged operations, of risk. Figure 3 shows both surfaces: the main window, where the user sees what the system is doing, and the confirmation dialogue of Section 3.3, where nothing proceeds without an explicit decision.



Figure 3. The two surfaces through which the user retains control, in their native Spanish, with intervening whitespace elided. Left, the main window: (A) activity log, (B) push-to-talk, (C) hands-free attention mode, (D) help, (E) tutorial. Right, the dialogue raised before any warning or dangerous operation: (A) what will be done and where, (B) the warning marker, (C) cancel or execute

4. EXPERIMENTAL EVALUATION

The system was evaluated along four lines: a benchmark of command generation and risk detection; a viability test of the offline mode; a usability study; and a controlled time comparison against the baseline the system displaces. The studies involving people used informed consent, coded identifiers and the right to withdraw.

4.1 A Note on the Figures Published in the Conference Version

The conference version reported 89 % command-generation accuracy over 200 tasks, a 9.4× speed-up against manual construction, and roughly 30 % of requests resolved locally—figures that came from manual verification by the lead developer, from informal timing, and from an estimate of how often routine interactions fall within the first tier's scope. All three have since been replaced by measurements taken with better instruments, and the numbers moved; we report the change explicitly rather than quietly substituting one set for the other.

Three things changed. The task set grew from 200 to 250, and evaluation was automated through a harness that drives the real pipeline and records every response. The correctness criterion was tightened to *action agreement* over an *in-scope* subset: a response counts as correct only when the action it selects is functionally equivalent to the expected one, and tasks whose expected command lies outside the system's repertoire leave the denominator rather than being scored generously. And the informal timing was replaced by the comparison of Section 4.6. Under the stricter criterion the headline accuracy is lower, the local-coverage figure much lower, and the speed-up smaller but, for the first time, statistically supported. The figures below supersede those in Aragon Toledo and Ruiz Rodríguez (2026).

4.2 Command Generation on the 250-Task Set

The evaluation set contains 250 Spanish requests across eight categories, each paired with an expected command and an expected risk level (219 safe, 23 warning, 8 dangerous); runs were performed on Ubuntu 22.04 LTS. With `deepseek-chat` the harness parsed all 250 responses without error and agreed on the action in 78.0 % of the 127 in-scope tasks. Agreement was complete on process control, web search and application launching, high on system information (94.1 %) and networking (92.3 %), and lowest on file management (63.2 %) and project creation (45.0 %). Residual errors are mostly benign—reading *copia* as a move, or *"muestra el contenido"* as an edit—and the weakest category reflects the model alternating between two adjacent actions rather than misunderstanding the request.

Running the same 250 tasks through `gpt-4o` produced a more interesting result than a ranking (Table 1). Where the two models differed the difference was systematic: faced with a request that omits a path—*"renombra archivo.txt a documento.txt"*—`gpt-4o` returns `clarify` where `deepseek-chat` assumes a sensible default, pushing its agreement down to 56.7 %. But once each model's clarification fallbacks leave the denominator, the two are indistinguishable and the ordering inverts: both exceed 87 % precision when they choose to act. The gap is not in competence but in temperament. The widest divergences were in file management (18.4 % against 63.2 %) and permissions (0 % against 75.0 %), both categories where a destination must be inferred; on web search and application launching both reached 100 %.

Table 1. Coverage against precision for the two cloud models over the in-scope subset (127 of 250 tasks). Precision when acting is correct answers divided by in-scope tasks minus clarification fallbacks. The two models agree on the action in 64.8 % of all 250 tasks

Metric (in-scope)	`deepseek-chat`	`gpt-4o`
Action agreement (coverage)	**78.0 %**	56.7 %
Fallback to `clarify`	11.0 %	36.2 %
Precision when acting	87.6 %	**88.9 %**

Of the 250 requests, 28 produced an action that modifies state. Five materialised as destructive operations, all deletions, and all five triggered confirmation: no safety false negatives, for either cloud model. Requests dangerous in a shell but outside the repertoire (`reboot`, `pkill`, `systemctl stop`) were routed to an informational action or to `clarify` rather than executed.

Measured on the same tasks, the local intent detector answered 11, or 4.4 %—far below the 30 % estimated in the conference version. The discrepancy is instructive rather than embarrassing: the first tier was never designed to generate commands, but to resolve conversational and immediate-information intents, which are frequent in daily use and almost absent from a benchmark composed deliberately of command-generation tasks. Its hits concentrate where one would predict: 40 % of application-launching tasks, 11.1 % of system-information tasks, 5.4 % of networking tasks, nothing elsewhere. The mechanism that genuinely reduces model calls for real work is the second tier, whose contribution we return to in Section 5.4.

4.3 Offline Viability

To bound what the system can do with no network, a 100-task subset—the 50 most recently added tasks plus a stratified sample of 50 from the rest, weighted towards warning and dangerous operations—was run against a local model. Three constraints had to be reconciled. A 1.3-billion-parameter model was abandoned for producing malformed JSON and drifting into unrelated prose; roughly seven billion proved the floor for consistently valid structured output. Reasoning-oriented models were ruled out because their deliberation traces multiply latency beyond what a voice assistant tolerates. And fitting six gigabytes of video memory meant Q4_K_M quantization rather than Q5 and a context pinned to 4 096 tokens, the model lacking grouped-query attention and its key–value cache costing roughly half a megabyte per token. Offloading all 32 layers to the GPU raised generation from about 1.7 to about 21 tokens per second.

The resulting configuration—`deepseek-coder-6.7b-instruct`, Q4_K_M, 4 096-token context, on an NVIDIA GTX 1060 with 6 GB—produced valid JSON on all 100 tasks and agreed on the action in 67.5 % of the 77 in-scope ones, against 77.2 % for `deepseek-chat` and 64.6 % for `gpt-4o` on the same tasks. Two caveats matter more than that headline. Latency rose from roughly 2–5 to roughly 5–45 seconds; and the local model never asked for clarification, so when confronted with a dangerous request outside its repertoire it reinterpreted it as some action it did know, producing two safety false negatives where both cloud models produced none. The deterministic layer absorbed them—precisely the case it exists for—but the finding qualifies the offline mode rather than endorsing it. These numbers establish that the approach already works on modest hardware, not what its ceiling is.

4.4 Multi-Distribution Validation

Supporting six families is a property of the code; working on six families is a claim that needs testing. The unattended installer was run on clean virtual machines, in each case through a single command—clone the repository, run the installer—with no manual intervention afterwards. Five criteria were checked: correct family detection, correct manager and package names, resolution of Perl dependencies including fallback to `cpanm`, a working application, and—deliberately harsher—installation of the offline wake word, Vosk plus its Spanish acoustic model.

Table 2. Unattended installation on clean virtual machines. Kali's partial result reflects the test network blocking the acoustic-model download, not a failure of the installer

Family	Distribution	Installer	Application	Offline wake word
`debian`	Ubuntu 22.04 LTS	yes	yes	yes
`debian`	Kali (rolling)	yes	yes	partial
`fedora`	Fedora 44 Workstation	yes	yes	yes
`arch`	EndeavourOS / Manjaro	yes	yes	yes

The installer adapted autonomously in all four cases (Table 2). Fedora is the most telling result: a machine from a different family, with a different manager and different package names, went from bare install to working application including the offline wake word without a single manual step. The exercise also surfaced four robustness problems invisible inside the development family, all now fixed—a cold `dnf` metadata cache aborting the batch, `HOME` pointing at `/root` under `sudo`, one failing package aborting the whole batch, and Perl's HTTPS support being absent from several base package sets. Four of six families, covering the three most widespread managers, are verified; the other two rest on the same dictionary but have not been exercised.

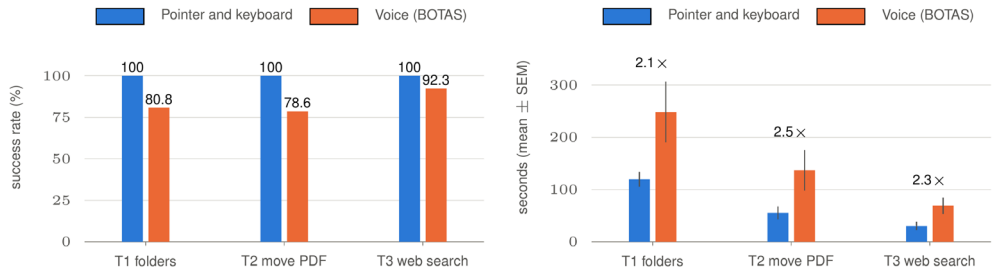
4.5 Usability Study

Fifteen sessions were run with members of the university community; the first was a pilot and its data are excluded, leaving fourteen participants (eight men, six women; mean age 25.9, range 22–32; eleven habitual Windows users, two macOS, one GNU/Linux). The design was within-subject for modality—every task first with pointer and keyboard, then by voice—and between-subject for the language model, participants being assigned alternately by order of arrival to `deepseek-chat` (seven) or `gpt-4o` (seven). Neither participants nor observers were told which model was in use.

Three tasks were presented as contextualised scenarios on cards, without prescribing the words to say: creating a course folder with six subject subfolders (T1); locating a specific PDF in the downloads folder and moving it into one of them (T2); and running a web search by voice (T3). Sessions followed a think-aloud protocol, the facilitator intervening only on explicit request or after three minutes without progress. Data came from observer sheets, post-task and final questionnaires, and system logs, analysed by inductive thematic coding (Braun and Clarke 2006) with effectiveness, efficiency and satisfaction framed as in International Organization for Standardization (2018).

4.5.1 Effectiveness and Efficiency

Success was scored with partial credit (1.0 complete, 0.5 partial, 0 failed). In the graphical environment every participant completed every task; by voice the weighted rate averaged 83.9 %, highest on web search (92.3 %) and lowest on moving a file (78.6 %), and voice was slower on all three tasks by factors between 2.1 and 2.5 (Figure 4).



(a) weighted success (b) time on task

Figure 4. Usability study, n = 14. Voice trails the graphical environment on both measures; the wide error bars reflect how differently participants behave on first contact with the modality

The dispersion matters more than the means. On T1 the voice standard deviation is 202.4 seconds against a mean of 248.1, with individual times from 61 to 773 seconds; the corresponding pointer figures are 39.5 and 119.4. Observation explains why: the slow sessions were slow not because the task was hard but because participants were establishing trust in an unfamiliar modality—pausing before speaking, rereading the result, rephrasing once to see what would happen. These are first-contact costs, not steady-state performance, which makes the reported means a conservative upper bound.

4.5.2 Satisfaction

Despite being slower, voice was rated equal to or slightly better than the graphical environment for perceived ease on all three tasks (4.23 against 4.14 on T1; 4.38 against 3.67 on T2; 4.31 on both for T3, on a five-point scale). Table 3 gives the final questionnaire, with an overall mark of 8.21 out of 10. The pattern across dimensions is worth reading closely: learnability scores highest at 4.54, comparable to longitudinal findings for voice assistants among non-expert users (Jakob et al. 2025), while naturalness of dialogue scores lowest at 3.46. Participants found the system easy to learn and awkward to talk to—the same asymmetry Berdasco et al. (2019) report for commercial assistants, suggesting a property of conversational interaction today rather than of this implementation.

Table 3. Final satisfaction questionnaire. The overall mark is on a ten-point scale, the remaining dimensions on five-point scales. Condition A is `deepseek-chat`, condition B is `gpt-4o`

Dimension	Overall	SD	Condition A	Condition B	n
Overall mark (1–10)	8.21	0.97	7.86	8.57	14
Learnability	4.54	0.66	4.33	4.71	13
Perceived accessibility	4.23	1.01	4.33	4.14	13
General satisfaction	3.62	0.77	3.67	3.57	13
Perceived speed	3.62	1.19	3.50	3.71	13
Naturalness of dialogue	3.46	1.13	3.50	3.43	13

Frustration never reached the moderate or high levels of the observation scale; every episode recorded was absent or mild, concentrated on T2. Most participants preferred voice for creating folders (8 of 12 expressing a preference) and for web search (7 of 13), while opinion split evenly on moving a file—the most spatial, most pointer-friendly task. With seven participants per condition no inferential claim is available, but the direction was consistent: `gpt-4o` led on eight of twelve compared dimensions, including voice effectiveness on all three tasks.

4.6 Voice Against Looking the Command Up

The study above compares voice with the graphical environment, and on that comparison voice loses on time. But the graphical environment is not what BOTAS replaces. A user who wants the files in a folder above a given size will not find that in a file manager without effort; they will search the web, find `find`, and adapt it. The relevant question is therefore not whether speaking beats clicking, but whether speaking beats *finding out what to type*—a comparison that had never been instrumented.

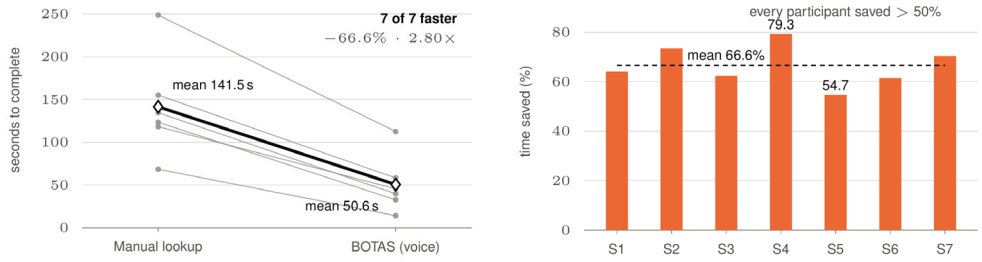
4.6.1 Design

Seven participants (S1–S7), none of whom had taken part in the earlier study or used the system, performed one task under two conditions in counterbalanced order, with a five-minute cap per condition. The task was read verbatim: *"In the Downloads folder there are many files. Show me all the files larger than 20 MB."* The reference solution is `find ~/Descargas -type f -size +20M`, and the folder was seeded so the correct answer was a proper subset of its contents. In the manual condition participants had a browser and a terminal and could consult anything they wished—search engines, forums, tutorials, and general-purpose conversational assistants—but not BOTAS and not the observer: a demanding baseline, and in 2026 the only honest one. Timing ran from the end of the scenario reading to the appearance of the correct listing.

The sample was screened for inexperience, and it was: five had never used GNU/Linux, five had never opened a terminal, six had never heard of `find`; ages ranged from 21 to 52 (mean 28.7), occupations from software development to accounting and graphic design. Seven pairs is small, and deliberate: the exact Wilcoxon signed-rank test (Wilcoxon 1945) has an arithmetic floor of $2/2^n$ on a two-tailed p, so five pairs cannot go below 0.0625 and six below 0.0312, whereas seven reaches 0.0156. A sensitivity analysis puts the minimum detectable effect at $d_z = 1.27$ for 80 % power at $\alpha = 0.05$ (Lakens 2013).

4.6.2 Results

Every participant was faster with BOTAS, with no exception and no tie (Figure 5); nobody hit the five-minute cap. Manual lookup averaged 141.5 seconds against 50.6 (Table 4): 90.9 seconds per task, a 66.6 % reduction, a factor of 2.80 on means and 2.95 on medians. The least favourable case still saved 54.7 % of the time.



(a) paired times (b) time saved

Figure 5. Manual command lookup against BOTAS, n = 7, within-subject and counterbalanced. Grey lines are individual participants, the heavy line the mean. The effect is present in every participant, including the one who already knew the command

Table 4. Descriptive statistics for the seven paired observations. Seconds unless stated otherwise

Condition	Mean	SD	Median	Min	Max
Manual command lookup	141.5	54.7	134.6	68.5	248.7
BOTAS by voice	50.6	30.8	45.6	14.2	112.6
Difference	90.9	25.1	91.0	54.3	136.1
Reduction (%)	66.6	8.3	64.1	54.7	79.3

The exact Wilcoxon signed-rank test gives $W^+ = 28$, $W^- = 0$, $p = .0156$; the sign test, 7 of 7, agrees; the paired t gives $t(6) = 9.60$, $p < .001$; the effect size is $d_z = 3.63$ and the 95 % confidence interval of the mean difference $[67.8,\ 114.1]$ seconds. $W^+ = 28$ is the complete sum of ranks 1 to 7, so $p = .0156$ is the smallest value seven pairs can produce: the design yielded the maximum evidence available to it. The observed effect is close to three times the minimum detectable effect, the interval is nowhere near zero, and the direction is unanimous. What is established is the existence and direction of the effect; the point estimate of 2.80× carries real uncertainty at this sample size.

Counterbalancing behaved as it should: participants who used BOTAS first saved 71.4 % and those who used it second 62.9 %. Had the advantage been an artefact of repetition, the group meeting BOTAS first should have shown the *smaller* advantage; the opposite occurred. The effort measures are as telling as the times. All seven completed the task unaided with BOTAS, against six of seven manually—the exception being the oldest participant, who did not know how to open a terminal. Participants averaged 2.43 command attempts manually against 1.00 by voice ($t(6) = 3.87$, $p = .008$), and made eleven consultations of search engines, tutorials or conversational assistants against none. All seven arrived at the same command in both conditions, and BOTAS generated it correctly every time without once asking for clarification.

The obstacle in the manual condition was not typing the command but discovering it. S1 obtained the correct command from a conversational assistant and still took 141.9 seconds against 50.9, because the cost lies in the whole journey—open the browser, phrase the question, read the answer, switch windows, transcribe, run—not the final keystroke. The voice times are conservative too: two participants paused the capture to think about how to phrase the request, and that hesitation is inside the measurement.

The benefit is also unevenly distributed. The correlation between a participant's manual time and the seconds they saved is $r = 0.97$: the further someone is from knowing the command, the more the system is worth to them. The accountant with no terminal experience took longest manually (248.7 seconds, five attempts, four web searches, and the only request for help) and saved the most in absolute terms, 136.1 seconds. Yet even S4—a junior developer who already knew `find` and produced it manually in 68.5 seconds—finished in 14.2 seconds by voice, the largest relative saving at 79.3 %. Removing that participant changes little: the remaining six average a 64.4 % reduction.

5. DISCUSSION

5.1 Two Speed Results that Appear to Contradict Each Other

The two studies produced findings that look opposed: voice was 2.1 to 2.5 times *slower* than pointer and keyboard, and 2.80 times *faster* than working the command out. Read together they say something more useful than either alone.

The three usability tasks—create folders, move a file, search the web—are ones the graphical environment already handles well: a file manager was built for exactly that, users have years of practice, and dictating six subject names is genuinely slower than typing them. The controlled task was different in kind. Filtering a directory by file size is something the graphical environment does not offer in any convenient form, so the realistic alternative is the terminal, and reaching it means discovering `find`, then its size syntax, then that `-type f` excludes directories.

A conversational interface should therefore not be positioned as a replacement for direct manipulation. It does not compete with the mouse on the tasks the mouse was designed for; it substitutes for the terminal on the tasks the graphical environment leaves unserved—precisely the tasks that exclude non-expert users today. Framed that way, the 83.9 % voice success rate is not a shortfall against the graphical environment's 100 % but a measure of how a modality performs on its rival's home ground at first contact. Every measurement in both studies was taken at first contact with no practice period, so the regime the learned-pattern tier is designed for—the second and third time a user asks for something—is one these studies do not reach.

5.2 Accessibility, Cost and Privacy

The correlation of 0.97 between manual time and time saved gives the accessibility argument an empirical footing it lacked: those who benefit most are furthest from the command line, the population the platform currently excludes. The absolute saving was largest for the participant with no terminal experience, and the relative saving largest for the one who already knew the answer—so the mechanism is not only knowledge substitution but the elimination of context switching. Cost and privacy interact with this. Fifty cloud requests a day cost a few dollars a month, trivial in some settings and an obstacle in the institutions this work targets; the offline mode removes that cost and, more importantly, removes data egress. The price is accuracy and latency: the offline mode is adequate for everyday safe tasks and demands the deterministic safety layer rather than merely benefiting from it.

5.3 Model Choice as an Interaction-Design Decision

The most transferable result of the benchmark is that two models with essentially equal precision when acting—87.6 % and 88.9 %—differ by more than twenty points in how often they act at all, `gpt-4o` asking for clarification three times as often as `deepseek-chat`. That is not a quality gap but a disposition, and it propagates into the user's experience: the same system feels autonomous with one model and cautious with the other. For a tool built on the premise that users stay in control this is a design parameter that a single accuracy number conceals entirely, and it argues for reporting coverage and precision separately whenever a system may decline to act.

5.4 Limitations

1. **The learned-pattern tier is characterised but not quantified.** Section 3.2 reports its mechanism, thresholds and costs, but no measured share of requests resolved: the benchmark harness runs with learning disabled—also the shipped default—to isolate parser accuracy, so the 4.4 % of Section 4.2 describes the first and third tiers only. Quantifying the second tier requires longitudinal use by real users, which no study here provides. This is the most significant gap in the evaluation.
2. **Sample sizes.** Fourteen participants support qualitative findings and descriptive statistics, not inference, and the comparison between model conditions is exploratory. The controlled comparison reaches significance, but its seven pairs bound the precision of the effect estimate, not its existence.
3. **One language, and researcher involvement.** Everything reported here is in Spanish, with a single Vosk acoustic model; performance on other Spanish varieties, let alone other languages, is untested, and the lexical-anchor dictionary is Spanish-specific by construction. The task set was also compiled by the lead developer, who facilitated the usability sessions: automating the harness removed the developer from scoring, but not from the design of what is scored.
4. **Coverage and composite requests.** Four of six families were verified, and even there only installation, not the full task battery. The architecture also processes one action per input: single-turn clarification handles missing parameters, but a request chaining several operations—find the large files, move them to a backup folder, compress it—is not decomposed into verifiable atomic steps.
5. **Scope of the safety layer.** It detects known categories of dangerous operation and does not evaluate context-dependent risk. The offline results show why this matters: a small local model reinterpreted out-of-repertoire dangerous requests as actions it did know, and the deterministic layer, not the model, stood in the way.

5.5 Future Work

Four directions follow. The first is a longitudinal deployment measuring what share of requests the learned-pattern tier serves over weeks of real use—the missing number in this paper. The second is decomposition of composite requests into a plan previewed in full, authorised as a batch with per-step exceptions for dangerous operations, and executed with checkpoints; the constraint is that each step must remain an action from the existing repertoire, so risk

classification and path validation apply unchanged. The third is extension of clarification from one turn to a dialogue, and the fourth a quantitative comparison against baselines such as ShellGPT. We also intend to release the system as free software.

6. CONCLUSION

This paper has presented an extended account of BOTAS, a conversational interface translating spoken Spanish into validated GNU/Linux commands under Human-Centered AI principles and has reported the evaluation the conference version could only gesture at. Three results stand out. First, on a 250-task benchmark scored by an automated harness under a strict action-agreement criterion, the system reaches 78.0 % coverage and 87.6 % precision when it acts, with no safety false negatives; the separation of those two numbers matters more than either, since a second model with near-identical precision declines to act three times as often. Second, against the baseline the system actually displaces—a non-expert working out what to type—it reduced completion time by 66.6 %, a factor of 2.80, with all seven participants faster and the advantage preserved under counterbalancing. Third, the same modality is 2.1 to 2.5 times slower than pointer and keyboard on tasks the graphical environment already handles well, which locates the contribution precisely: a conversational interface is not a replacement for direct manipulation but a substitute for the terminal on the tasks direct manipulation leaves unserved.

The extensions reported here—a learning tier that turns each solved request into a reusable parameterised template, a fully local operating mode validated on a six-gigabyte consumer GPU, and multi-distribution support verified by unattended installation across three package managers—all point the same way: towards a system that costs less, discloses less and depends less on the network the longer it is used. Their central limitation is stated plainly above: the learning tier's real-world contribution remains unmeasured.

What these results suggest for the wider discussion of AI in society is narrower than a claim about democratising computing, and firmer. The benefit of this class of tool is not uniform; it scales with distance from expertise, at $r = 0.97$ in our data. That is an argument for building such interfaces where that distance is greatest, and for measuring them against the barrier they are meant to remove rather than against the interface that never posed it.

ACKNOWLEDGEMENT

The authors thank the participants of both studies for their time, and the Universidad Tecnológica de la Mixteca for institutional support.

REFERENCES

Alpha Cephei. (2023). *Vosk Speech Recognition Toolkit.* Alpha Cephei.

Amershi, S. et al. (2019). Guidelines for Human–AI Interaction. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13.

Aragon Toledo, Jose Ramon, and Ricardo Ruiz Rodríguez (2026). BOTAS: A Human-Centered Natural Language Interface for GNU/Linux System Management with Multi-Distribution Support. In P. Isaias, P. Miranda, and G. C. Peng (Eds), *Proceedings of the International Conferences on Artificial Intelligence in Society 2026, Digital Transformation and Innovation Management 2026 and Connected Smart Cities 2026*, 100–110. Valencia, Spain: IADIS Press.

Bang, F. (2023). GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings. *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software*, 212–218.

Berdasco, A. et al. (2019). User Experience Comparison of Intelligent Personal Assistants: Alexa, Google Assistant, Siri and Cortana. *Proceedings of UCAmI 2019,* 31(1), 51.

Braun, V., and Victoria, C.. (2006). Using Thematic Analysis in Psychology. *Qualitative Research in Psychology, 3*(2), 77–101.

Brown, T. et al. (2020). Language Models Are Few-Shot Learners. In *Advances in Neural Information Processing Systems*. Vol. 33.

CEPAL. (2023). *Panorama Digital de América Latina y El Caribe 2023: Avances y Desafíos de La Transformación Digital.* Comisión Económica para América Latina y el Caribe, Naciones Unidas.

Dobberstein, L. (2023). *Creator of Linux Virtual Assistant Blames 'Patent Troll' for Project's Death*. The Register.

International Organization for Standardization. (2018). *ISO 9241-11:2018. Ergonomics of Human–System Interaction — Part 11: Usability: Definitions and Concepts.* Geneva, Switzerland: International Organization for Standardization.

Jakob, D. et al. (2025). Adapting Voice Assistant Technology for Older Adults. *Digital, 5*(1), 4.

Jenson, S. (2024). *The Future of Desktop UX: Breaking the Cycle of Stagnation.* Scott Jenson, personal site.

Lakens, D. (2013). Calculating and Reporting Effect Sizes to Facilitate Cumulative Science: A Practical Primer for t-Tests and ANOVAs. *Frontiers in Psychology, 4*, 863.

Lin, X. V. et al. (2018). NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System. *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*.

Luger, E. and Sellen, A. (2016). Like Having a Really Bad PA': The Gulf Between User Expectation and Experience of Conversational Agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 5286–5297.

Manning, C. D. et al. (2008). *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press.

Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.

Norman, D. A. (2013). *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.

OpenAI. (2023). *GPT-4 Technical Report*. OpenAI.

OpenVoiceOS. (2024). *OVOS-core: The FOSS Artificial Intelligence Platform*. OpenVoiceOS project repository.

Radford, A. et al. (2023). Robust Speech Recognition via Large-Scale Weak Supervision. *Proceedings of the 40th International Conference on Machine Learning*.

Saha, S. et al. (2014). *Jasper: An Open Source Platform for Developing Always-on, Voice-Controlled Applications*. Jasper project repository.

Shchekotov, F. (2024). *ShellGPT: A Command-Line Productivity Tool Powered by AI Large Language Models*. ShellGPT project repository.

Shneiderman, B. (2020). Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy. *International Journal of Human–Computer Interaction*, 36(6), 495–504.

Shneiderman, B. (2022). *Human-Centered AI*. Oxford University Press.

Shneiderman, B. and Plaisant, C. (2010). *Designing the User Interface: Strategies for Effective Human–Computer Interaction* (5th ed.). Addison-Wesley.

Top500.org. (2023). *Operating System Family / Linux.* TOP500 Project.

W3Techs. (2023). *Usage Statistics of Linux for Websites*. W3Techs Web Technology Surveys.

Warschauer, M. (2004). *Technology and Social Inclusion: Rethinking the Digital Divide*. MIT Press.

Wei, J. et al. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems, 35*.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin, 1*(6), 80–83.

Wobbrock, J. O. et al. (2011). Ability-Based Design: Concept, Principles and Examples. *ACM Transactions on Accessible Computing, 3*(3), 1–27.