ORCHESTRA: AI-Driven Microservices Architecture to Create Personalized Experiences

Jaime Bellver-Soler[1], Samuel Ramos-Varela[1], Anmol Guragain[1], Ricardo Córdoba[1], Luis Fernando D'Haro[1],

[1]Speech Technology and Machine Learning Group - ETSI de Telecomunicación
Universidad Politécnica de Madrid, Spain

Correspondence: jaime.bellver@upm.es

Abstract

Industry stakeholders are willing to incorporate AI systems in their pipelines, therefore they want agentic flexibility without losing the guaranties and auditability of fixed pipelines. This paper describes ORCHESTRA, a portable and extensible microservice architecture for orchestrating customizable multimodal AI workflows across domains. It embeds Large Language Model (LLM) agents within a deterministic control flow, combining reliability with adaptive reasoning. A Dockerized Manager routes text, speech, and image requests through specialist workers for ASR, emotion analysis, retrieval, guardrails, and TTS, ensuring that multimodal processing, safety checks, logging, and memory updates are consistently executed, while scoped agent nodes adjust prompts and retrieval strategies dynamically. The system scales via container replication and exposes per-step observability through open-source dashboards. We ground the discussion in a concrete deployment: an interactive museum guide that handles speech and image queries, personalizes narratives with emotion cues, invokes tools, and enforces policy-compliant responses. From this application, we report actionable guidance: interface contracts for services, where to place pre/post safety passes, how to structure memory for RAG, and common failure modes with mitigations. We position the approach against fully agentic and pure pipeline baselines, outline trade-offs (determinism vs. flexibility, latency budget), and sketch near-term extensions such as sharded managers, adaptive sub-flows, and streaming inference. Our goal is to provide a reusable blueprint for safely deploying agent-enhanced, multimodal assistants in production, illustrated through the museums use case.

1 Introduction

AI applications often require coordinating multiple specialized models or services to solve complex, multimodal tasks. For example, an AI system may need to analyze a user's text input, detect the emotional tone, then use that information to craft a personalized response. Handling text, audio, video, and images in an integrated way demands an orchestration mechanism that can route data through the appropriate sequence of models (Liu et al., 2023).

Two main paradigms have emerged for such orchestration: agent-based systems and workflow-based systems (Durante et al., 2024). In the agent-based paradigm, a large language model (LLM) serves as a central planner that dynamically decides which tools or models to invoke and in what order. In contrast, workflow-based orchestration (often implemented as a microservice or pipeline architecture) follows an explicit, predefined sequence of modules designed by developers, possibly with limited conditional branches for flexibility. These two paradigms represent opposite ends of a spectrum: agent-based approaches maximize autonomy, while workflow-based systems emphasize determinism and auditability. Recent research and industrial practice have explored hybrid forms (Liu et al., 2023), and several orchestration frameworks (Chase, 2022; Liu, 2022; Wu et al., 2024) now combine both by letting LLMs adapt prompts or retrieval parameters inside a fixed graph. LLMs alone exhibit strong reasoning and linguistic capabilities but remain limited in real-time learning, long-term memory, and multi-step execution (Jovanovic and Voss, 2024; Wang et al., 2023), motivating structured workflow-based architectures like ORCHESTRA.

This paper introduces ORCHESTRA, a microservice-based orchestration architecture that guarantees end-to-end safety, memory management, and observability while embedding agentic LLM flexibility. We share the design considerations, trade-offs, and implementation challenges faced when moving from prototypes to production, together with practical solutions obtained in real deployments. The goal is to

Proceedings of the 16th International Workshop on Spoken Dialogue Systems Technology, pages 158–167
February 26–March 1, 2026.

offer readers a reproducible blueprint illustrating how to structure agent-enhanced workflows, ensure deterministic safety coverage, and maintain scalability and transparency.

Definitions and Scope Throughout this paper, we use agent-based to mean LLM-first systems that dynamically plan tools use and control flow, and workflow-based to mean developer-specified graphs/pipelines where the control flow is explicit. Microservice orchestration (our approach) is a concrete instance of the workflow-based paradigm: each capability runs in its own service, and a Manager routes requests through a predefined graph. ORCHESTRA embeds scoped agentic decisions at selected nodes inside this fixed graph. In short: microservices $\subset$ workflow-based orchestration; agentic LLMs are used inside the workflow, not to replace it.

2 Related Work

We organize prior work by orchestration style and highlight reliability, safety coverage, and observability, three axes central to our design goals. Agent-first systems maximize flexibility but make it harder to guarantee that mandatory steps (e.g., safety filters, memory updates) are always executed. Workflow-first systems trade some flexibility for determinism and easier auditing. Our work positions a microservice workflow as the default path while confining agent autonomy to well-scoped micro-decisions.

2.1 Agent-Based Systems

Agentic LLM systems use an LLM as a central "brain" that autonomously plans and invokes tools or other models to achieve a goal (Durante et al., 2024). Such an agent perceives a user request, breaks it into sub-tasks, and decides which operations to perform in sequence, using techniques like chain-of-thought prompting for planning (Wei et al., 2022). The agent holds a memory of the dialogue or past steps and can use external tools (via plugins, APIs, etc.) to fetch information or take actions. This paradigm was popularized by frameworks like LangChain (Chase, 2022), which provides an abstraction that lets an LLM choose among available tools to answer queries. HuggingGPT (Shen et al., 2023) demonstrated that LLM agents can indeed use tools to improve accuracy and handle tasks beyond pure text generation. Notably, HuggingGPT showcased an LLM (ChatGPT) acting as a controller that, given a user query, plans a task list, selects appropriate expert models for each subtask, executes them, and composes the final answer. This allowed tackling a wide range of multimodal problems by delegating to specialized models, with the LLM orchestrating the entire process. Similarly, Microsoft's TaskMatrix.AI (Liang et al.) concept envisioned "foundation models" like ChatGPT as a brain that can call up millions of external APIs or models as needed, rather than trying to solve everything with a single model. These works illustrate the promise of agent-based orchestration: extensibility (the agent can incorporate new tools as they become available) and flexibility (the sequence of actions is decided dynamically per the task at hand).

However, purely autonomous LLM agents also come with significant challenges. By letting the model decide its own tools and plan, we entrust a lot of control to a probabilistic system that is not able to clearly cover all possibilities and therefore potentially become unpredictable. Researchers and practitioners have observed that LLM agents are prone to hallucinations and mistakes in multi-step reasoning (Wang et al., 2024). Errors can compound when an agent reasons incorrectly yet continues down a wrong path autonomously. For instance, AutoGPT (Yang et al., 2023) early experimental "autonomous agent" often got stuck in loops or failed to complete objectives reliably. These agents keep going on the wrong path, amplifying small mistakes into large failures. Another issue is related to tool selection: an LLM faced with dozens of possible tools may not reliably choose the correct one at each step. The LangChain team noted that an agent is "more likely to succeed on a focused task than if it has to select from dozens of tools" (LangChain, 2024). In other words, giving an agent complete freedom can lead to it skipping or misusing tools that a human designer would deem necessary. Moreover, debugging or observing the reasoning of an autonomous agent is difficult. The agent's decision process is essentially hidden in its internal chain-of-thought, making it hard to trace why it took a certain action when things go wrong. This lack of transparency and determinism is problematic for high-stakes or enterprise applications.

Due to these issues, recent efforts have tried to improve agents' reliability. One approach is using multiple agents to oversee each other in a multi-agent setup. Multi-agent frameworks (Chase, 2022;

Wu et al., 2024; CrewAI, 2024) allow designing specialized agents that collaborate. Each agent can be assigned a particular role or expertise, and they communicate to solve the overall task. This division of labor makes each agent's job easier and more constrained. Indeed, multi-agent designs have shown benefits such as improved factuality and reasoning and the ability to handle longer contexts by splitting input among agents (Guo et al., 2024). Multi-agent frameworks enable developers to script conversation patterns between agents (and humans) to enforce a structure in their interaction. Nonetheless, even in multi-agent mode, the core planning logic is often driven by LLMs generating next-step decisions. Fully autonomous agent frameworks thus trade off adaptability for reliability. They shine in open-ended scenarios where the sequence of actions cannot be known in advance, but they risk failure in scenarios where certain steps are critical.

Reliability and observability. Prior agentic systems report failure modes such as tool misselection, looping, or skipped safety checks (Yang et al., 2023; Wang et al., 2024; LangChain, 2024). Because plan construction is probabilistic and internal to the model, auditing and reproducing failures is difficult. This motivates our choice to bound agent autonomy within a deterministic graph that guarantees safety and memory passes while still allowing local adaptation.

2.2 Workflow-based Systems

In contrast to agents, workflow-based orchestration uses a predetermined sequence of modules and tools, often orchestrated by a central manager or controller service treating the AI system like a pipeline or flowchart (LlamaIndex). The emphasis is on a scripted flow designed by humans rather than spontaneous tool use. Such architectures are common in industrial AI deployments and are analogous to microservice architectures in software engineering: each AI capability is a microservice, and a top-level controller service routes data and results between them. For instance, an AI assistant might always follow these steps for a text query: transcription → emotion analysis → response generation → response speech synthesis. By hard-coding this workflow (possibly with configurable parameters), we ensure the system always performs all key steps in the loop. The trade-off is a reduced flexibility; if a query doesn't actually need a certain step, the pipeline might do extra work. But the benefit is predictability and completeness: crucial operations won't be accidentally skipped due to an LLM's whim or error.

This workflow philosophy has been reinforced by recent frameworks that highlight structured orchestration. IBM's Watsonx Orchestrate (IBM, 2023), for example, focuses on AI-powered workflow automation for business processes, where the sequence of tasks is largely predefined and the AI components fill in specific steps (with humans in the loop as needed). In the LLM tooling space, libraries like Haystack (deepset, 2025) take a pipeline approach to tasks like question answering; a developer explicitly chains a retriever component with an LLM component, rather than letting the LLM figure that out itself. Another example is the Semantic Kernel (Microsoft, 2024), which encourages developers to define skills and plans that an LLM can execute, combining code with AI calls in a controlled manner. These platforms underscore software engineering principles in AI: deterministically orchestrating modules, monitoring each step, and handling errors or timeouts in a predictable way.

One advantage of workflow orchestration is ease of monitoring and debugging. Because the flow is explicit, developers can insert logging, perform unit tests on individual modules, and know exactly which stage produced a faulty output. This is much harder in an agent that decides its own sequence. Workflow orchestrators also enable guardrails at each step that evaluate and correct LLM outputs, ensuring the general safety of the system (Ayyamperumal and Ge, 2024).

It is worth noting that "workflow" does not mean completely inflexible or linear. Many orchestration systems support conditional branches and even loops. LangGraph is an illustrative example: it lets developers construct a graph of LLM calls and tool calls with defined transitions. This approach allows limited agent improvisation inside a controlled workflow. Guiding the agent with a workflow can yield a more robust system than a completely self-directed agent (LangChain, 2024). This insight aligns with our proposal: rather than trust an agent to always know which tools to use, we prescribe a default workflow (sequence of tool usages) and only give the agent autonomy in well-scoped micro-decisions.

Another recent example of structured and flexible orchestration is the introduction of function calling in LLM APIs (Kim et al., 2023). Instead

of relying on the model to output a well-structured tool invocation via plain text, function calling lets the developer predefine the available tools and their JSON schema, and the LLM will return a structured invocation if it decides one is needed. This reduces hallucinations, because the LLM's decision is immediately validated against a schema.

An alternative to APIs is the Model Context Protocol (MCP) (Hou et al., 2025): instead of shipping the JSON schema with every request, developers host it on an MCP server that advertises its capabilities via a standard JSON-RPC interface; LLMs can then discover and invoke those tools at runtime. This separation of discovery from invocation allows teams to publish a capability once and reuse it across many agents.

The state of the art in AI orchestration is converging on the idea of safe autonomy: harnessing the creativity of LLM agents while orchestrating them within guardrails and structured flows. In this context, ORCHESTRA contributes practical insights. Since it is not publicly released, our focus is on sharing the architectural rationale, design lessons, and engineering trade-offs that practitioners can apply to their own systems. We detail how workflow determinism was reconciled with agentic adaptability, what technical constraints emerged when managing multimodal data, and which debugging and observability strategies proved essential for reliability in production. We complement the conceptual discussions in existing open-source frameworks like LangChain or LlamaIndex, providing a grounded perspective on what it takes to create hybrid orchestration architectures.

2.3 Positioning and Comparison

The following summary situates ORCHESTRA relative to representative frameworks. Our contribution is not a new agent algorithm, but a systems design that *guarantees* end-to-end safety and memory coverage with per-step observability while preserving scoped agentic flexibility. A qualitative comparison of representative orchestration frameworks is as follows:

- **LangGraph** (LangChain, 2024): partially agentic planning within developer-defined graphs; limited safety enforcement; moderate observability; typically text-only; distributed as a Python library.
- **LlamaIndex Workflows** (LlamaIndex): combines retrieval and LLM nodes in explicit pipelines; partial determinism and monitoring; primarily unimodal.
- **AutoGen** (Wu et al., 2024): fully agentic multi-agent dialogues; minimal safety determinism; low observability; limited multimodal support.
- **ORCHESTRA (ours):** scoped agent autonomy inside a deterministic microservice workflow; full safety coverage and trace-level observability; multimodal (text, audio, image); deployable as containerized microservices.

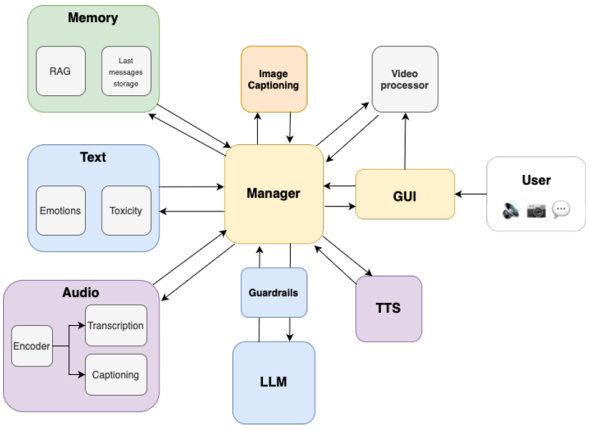

Figure 1: ORCHESTRA architecture diagram.

Our contribution lies in the engineering design and empirical insights gained from deploying such a hybrid orchestration system in production, not in proposing a new agent algorithm. ORCHESTRA demonstrates how to combine containerized workflows with embedded LLM agents to ensure deterministic execution of safety, memory, and logging steps, while still allowing adaptive behavior where it adds value.

3 Architecture

As illustrated in Fig. 1, ORCHESTRA is implemented as a constellation of Docker containers that communicate via lightweight HTTP/REST endpoints using FastAPI. At the center sits a `manager/` container that receives multimodal user requests, assigns a trace ID for observability, and dispatches data to the corresponding specialist workers. Each box is an independently deployable container exposing a uniform `/infer` endpoint. Metrics and traces are emitted at every hop, enabling full per-step observability through Grafana dashboards.

This architecture provides three strategic benefits: (i) scalability—containers can be replicated

or assigned to separate GPU nodes, (ii) fault isolation—failures in one module do not propagate to others, and (iii) flexible development—although the current implementation is in Python, any container can be replaced by binaries in other languages without altering the control plane.

3.1 Component Stack

The layered component stack is summarized below. All containers share a common request/response schema based on JSON objects, which may embed raw bytes (image or audio) or base64-encoded embeddings. The layered component stack can be summarized as follows:

- **Orchestration layer:** `manager/` container. Receives requests, detects modality, assembles the per-task workflow, aggregates outputs, and returns the response.
- **AI service layer:** Specialized modules such as `asr/`, `captioning/`, `emotion/`, `toxicity/`, `lm/`, and `tts/`. Each exposes a lightweight REST `/infer` endpoint.
- **Memory layer:** `retriever/` plus a vector database (FAISS, Chroma, or Neo4j). Handles both short-term dialogue context in RAM and long-term retrieval-augmented memory.
- **Interface layer:** Gradio GUI and REST API endpoints that provide human and programmatic access for downstream applications.
- **Observability layer:** Prometheus and Grafana dashboards offer centralized logging, per-module latency and throughput metrics, toxicity counters, and distributed tracing.

3.2 Data Flow (Baseline Pipeline)

The `manager/` executes a predefined baseline pipeline (Steps 1–8).

1. **Ingress and parsing**: The Manager inspects the request type and user metadata, attaches a trace ID, and logs initial parameters.
2. **Modality dispatch**:

   Image → `captioning/` → image captions

   Audio → `asr/` → transcript + acoustic features

   Text sent directly to Step 3
3. **Emotion and Safety Pass**: The text (original or transcribed) is sent to `toxicity/` and `emotion/`. If toxicity $> \tau$, the request is refused and rerouted to the `guardrails/` module, which generates a safe fallback.
4. **Prompt composer**: The Manager merges raw content, detected emotion, and short-term memory into a task-specific prompt.
5. **Agent LLM (`lm/`)**: The embedded LLM agent (Llama 3-7B) generates the final response and may use external tools through MCP for retrieval, reasoning, or web access.
6. **Safety Pass (post-generation)**: The output is rechecked by `toxicity/`. If blocked, predefined rules produce a safe fallback.
7. **Optional TTS**: When speech output is requested, `tts/` synthesizes the final response.
8. **Response and Logging**: The Manager records latency and module-level metrics under the trace ID, updates memory stores, and returns the result to the GUI/API. Anonymized logs are retained for 30 days.

This deterministic end-to-end path guarantees that every request passes through mandatory safety and memory stages. The only adaptive segment is inside Step 5, where a confined agent decides which tools or tonal style to use. The overall flow remains auditable.

3.3 Module Specifications

Each component in ORCHESTRA is an independent microservice that can be updated or replaced without changing the control plane. Lightweight models are preferred for privacy and efficiency, but the platform can mix local and API-based models.

Manager Acts as the orchestration brain: parses incoming requests, tags with trace IDs, dispatches to modality services, composes prompts, gathers outputs, and handles observability hooks (traces, metrics, logs).

Image Captioning We use the SmolVLM-Instruct (Marafioti et al.) model for frame-level captioning and vision tasks, although any other vision LLM could be used.

Video Processor Leverages SmolVLM's multimodal capabilities for video understanding, frame sampling, temporal reasoning, and scene summarization (Team, 2025).

Text (Emotion & Toxicity) The text module performs multimodal emotion classification using a multimodal Speech Emotion Recognition (SER) model (Bellver et al., 2024) and enforces content safety via Llama Guard-3's toxicity detector (Inan et al., 2024).

Audio The audio module begins with a Transformer encoder that turns raw waveforms into high-dimensional embeddings. Automatic speech recognition is handled by OpenAI Whisper Large v3 (Radford et al., 2023), providing accurate transcriptions complete with timestamps. Finally, a unified audio-captioning and emotion-recognition pipeline annotates acoustic scenes and emotional tone, employing a emotion recognition model (Bellver-Soler et al., 2025) alongside QwenAudio2 for comprehensive audio understanding (Chu et al., 2024).

LLM Agentic Embeds Llama 3-7B (Grattafiori et al., 2024) for generative response via tool-using agents (MCP). Llama 3 offers a balance of reasoning and cost, available in open-source form.

Guardrails Enforces YAML-based content policies backed by *Llama Guard-3* safety classification; unsafe or toxic inputs/outputs are refused or sanitized before response (and again post-generation when applicable).

Memory (Retrieval-Augmented Generation & Session Messages) The memory module comprises a Qdrant vector database (Qdrant Tech, 2023) that stores document embeddings produced by a dedicated embedder model (Warner et al., 2024), enabling efficient cosine-similarity retrieval of context relevant to each user prompt. Concurrently, recent conversation turns are retained in an in-RAM session buffer, without persisting to an external database, to maintain short-term dialogue state.

TTS Runs Coqui XTTS (v2) (Casanova et al., 2024) for neural, multi-speaker voice synthesis via a diffusion-inspired backbone.

GUI Gradio-based (Abid et al., 2019) web interface and REST endpoints for human chat, file upload, and downstream integrations.

All modules expose a `/infer` endpoint and declare their own CPU/GPU requirements. Models listed above (Llama3-7B, Whisper-large-v3, SmolVLM, SmolVLM2, Llama Guard-3, Coqui XTTS, etc.) are entirely pluggable: you can swap in alternative architectures (e.g. Rust/C++ binaries, other open-source or proprietary checkpoints) without any changes to the Manager's control flow or inter-service schema.

3.4 Deployment

We target servers with at least 48 GB VRAM, 24 CPU cores, and 32 GB RAM (64 GB recommended for concurrent multimodal pipelines). Storage should be NVMe SSDs (500 GB) for model checkpoints and fast I/O. Docker Swarm pins GPU-intensive modules (ASR, captioning, LLM inference) to GPU nodes.

Artifact and reproducibility. A minimal Docker Compose configuration with services for `asr/`, `retriever/`, and `lm/` reproduces the full control flow and observability features without proprietary models. The package includes seed exhibit data and example Grafana dashboards.

3.5 Security and Fault Tolerance

Toxicity assessment runs twice (pre- and post-generation) using *Llama Guard-3* (Inan et al., 2024) as the primary safety classifier. The `guardrails/` module enforces YAML-based content policies and injects refusals or sanitized rewrites when required. Memory writes are filtered to exclude banned content, and all operations are logged with the trace ID.

Threat model and hardening. We assume authenticated clients and an adversary who may attempt prompt-injection or adversarial inputs. The Manager sanitizes user prompts before forwarding, containers run with least privilege and no host mounts, and inter-service traffic is restricted to internal networks.

Fault tolerance and degraded modes. Each worker periodically reports its health status to the Manager, which supervises all modules through lightweight heartbeats and latency metrics. When a call fails, the Manager automatically retries it with exponential backoff ($max_{retries} = 2$) to handle transient network or GPU issues. If a module remains unavailable after retries, the system activates a degraded mode designed to maintain continuity and safety rather than failing silently.

For example, if `emotion/` becomes unreachable, the Manager injects a neutral emotional tone so downstream modules can continue operating consistently. If `retriever/` fails, the LLM still produces a response but without external context, explicitly adding a confidence disclaimer such as "I may not have access to all information right now." Similar fallbacks are defined for other modules to ensure graceful degradation instead of service interruption. All such events are recorded with their trace IDs in centralized logs, allowing developers to audit failure patterns, analyze uptime, and fine-tune retry or fallback strategies.

3.6 Extensibility

Adding a new modality requires only creating a new container folder with `app/main.py` and a `Dockerfile`, then registering its endpoint in the Manager routing table. Replacing the LLM backend involves swapping the `lm/` Docker image (e.g., with a LoRA-fine-tuned variant) without changing API contracts.

4 Use Case

To showcase ORCHESTRA in a specific scenario, we consider its deployment as an interactive museum guide. In this setting, visitors interact with the system via speech or text, asking questions about artworks, exhibits, or museum logistics. ORCHESTRA's microservice workflow ensures each user interaction passes through (i) speech recognition, (ii) emotion and safety filtering, (iii) context-aware retrieval, (iv) adaptive LLM response generation, and (v) dynamic tools integration, all while maintaining full observability and audit trails.

Upon user utterance, the `asr/` service transcribes the query, while the emotion-classification module gauges the visitor's tone (e.g., curiosity, confusion). A short-term memory buffer tracks previous questions, enabling follow-ups such as "Tell me more about that artist." The retrieval module then fetches relevant exhibit metadata from a Qdrant vector store. At the `lm/` agent node, the LLM generates a personalized narrative, choosing an appropriate level of knowledge based on detected expertise and user profile parameters.

Beyond text, visitors may request a map view to locate themselves. ORCHESTRA's MCP-enabled agent can invoke a 'museum_map' tool (a simple microservice returning an image of the floor plans), embedding visual information directly into the response. Apart from that, if a visitor poses sensitive questions (e.g., about politically charged artworks), the pre- and post-generation toxicity checks ensure policy compliance, routing any borderline content through an explicit guardrails service that supplies safe and neutral summaries. All service calls emit trace IDs and metrics to Grafana, helping curators to monitor peak usage times or common information gaps.

In production, the museum guide can scale horizontally, handling visitor streams. Fault isolation ensures that an upgrade to a service does not disrupt core dialogue flows. Moreover, detailed logs of retrieval queries and agent decisions can be reviewed to refine prompt compositions or to retrain the proprietary models for a more engaging experience.

Configuration. For the museum guide, we enable the baseline path `asr/` $\rightarrow$ `toxicity/` $\rightarrow$ `emotion/` $\rightarrow$ `retriever/` $\rightarrow$ `lm/` $\rightarrow$ `toxicity/` $\rightarrow$ `tts/` (optional). The `retriever/` is seeded with exhibit metadata (titles, artists, dates, room IDs) and floor-plan vectors. The `lm/` agent exposes tools: `museum_map(room_id)`, `open_hours()`, and `route(from,to)` via MCP.

Call sequence (speech query). (1) `asr/` returns transcript + timestamps. (2) `toxicity/` prefilter; if blocked, guardrails/ returns a safe refusal. (3) `emotion/` annotates tone (`curious/confused/hurried`). (4) `retriever/` fetches exhibit nodes (top-k) by cosine similarity. (5) `lm/` composes a narrative; when asked "Where is the Starry Night?", the agent calls `museum_map(room_id=101)` and embeds the returned image reference in the response. (6) `toxicity/` postfilter; if blocked, `guardrails/` rewrites. (7) `tts/` synthesizes audio when voice output is requested.

Operator observability. For each turn, Grafana dashboards show per-service latency, tool-call counts, and safety triggers keyed by the trace ID, enabling curators to pinpoint failure causes and popular exhibits.

5 Discussion and Future Work

5.1 Discussion

Our guided-workflow microservice architecture achieves three goals that current purely agentic systems struggle with: (i) predictable coverage of

critical checks such as safety filtering and memory updates, (ii) observability at every hop (trace-ID + metrics), and (iii) a clear operational contract for each module, enabling fault isolation. In effect, we graft a thin layer of agentic flexibility onto a rigid flow graph, mirroring the direction taken by graph-orchestration frameworks such as LangGraph, where developers "balance agent control with agency" by laying out an explicit state machine before injecting LLM calls.

Compared with multi-agent controllers such as AutoGen, which allow agents to negotiate an arbitrary conversation plan, our design gives up some breadth of problem exploration but gains auditability. In enterprise settings where a missed toxicity check or memory write is a hard failure, that trade-off is favorable. A side effect is lower mean latency because the Manager never loops indefinitely, whereas unsupervised agents may enter into a loop or stall.

A single Manager remains a potential bottleneck, both computationally (routing overhead, JSON validation) and organizationally (any new modality requires a Manager update). Moreover, a fixed flow can under-serve long-tail user tasks that do not fit the canonical path. Recent LLM releases with real-time multimodal streaming (e.g. GPT-4o) raise expectations of sub-second responses; our current pipeline may exceed that budget for long audio.

5.2 Limitations

Despite its strengths, the current prototype has four main constraints. First, we rely on a single Manager container to route and validate every request, which makes it a potential bottleneck. Second, the pipeline's graph is hard-coded: new or unforeseen composite tasks may require code changes rather than simply "emerging" from agent planning. Third, all services run sequentially; a 30-second audio clip, for example, still incurs cumulative ASR $\rightarrow$ emotion $\rightarrow$ prompt $\rightarrow$ LLM latency that can exceed the sub-second round-trip time end-users now expect. Fourth, our current observability stack focuses on infrastructure-level metrics (latency, module uptime, error rates) and traceability, but does not yet include automatic evaluation of reasoning quality or factual faithfulness. At this stage, our efforts have prioritized establishing robust guardrails, safety passes, and deterministic orchestration, laying the foundation for future optimization and quantitative monitoring of model behavior.

5.3 Future Work

1. Shard the Manager behind a stateless router to remove the single-bottleneck risk. 2. Add adaptive sub-flows via function-calling or policy rules so that agents can skip or merge steps when confidence is high, shaving latency without sacrificing safety. 3. Introduce streaming paths (chunked ASR, incremental LLM decoding, lightweight guardrails) to meet real-time interaction budgets while preserving the existing safety envelope.

6 Acknowledgments

This work is supported by project BRAINS (PID2024-155948OB-C52) funded by MCIN/AEI/10.13039/501100011033 and, as appropriate, by the "European Union" and by the European Commission through Project ASTOUND (101071191 — HORIZON-EIC-2021-PATHFINDERCHALLENGES-01). In addition, it is supported by project BEWORD (PID2021-126061OB-C43) funded by MCIN/AEI/10.1303-9/501100011033 and, as appropriate, by "ERDF A way of making Europe", by the "European Union".

References

Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. 2019. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569*.

Suriya Ganesh Ayyamperumal and Limin Ge. 2024. Current state of llm risks and ai guardrails. *arXiv preprint arXiv:2406.12934*.

Jaime Bellver, Ivan Martín-Fernández, Jose M Bravo-Pacheco, Sergio Esteban, Fernando Fernández-Martínez, and Luis Fernando D'Haro. 2024. Multimodal audio-language model for speech emotion recognition. In *Proc. odyssey 2024*, pages 288–295.

Jaime Bellver-Soler, Mario Rodríguez-Cantelar, Ricardo Córdoba, and Luis Fernando D'Haro. 2025. Cutting through overload: Efficient token dropping for speech emotion recognition in multimodal large language models. In *Proceedings of the 15th International Workshop on Spoken Dialogue Systems Technology*, pages 284–289.

Edresson Casanova, Kelly Davis, Eren Gölge, Görkem Göknar, Iulian Gulea, Logan Hart, Aya Aljafari, Joshua Meyer, Reuben Morais, Samuel Olayemi, et al. 2024. Xtts: a massively multilingual zero-shot text-to-speech model. *arXiv preprint arXiv:2406.04904*.

Harrison Chase. 2022. Langchain: Language model application development framework. https://

github.com/langchain-ai/langchain. Version 0.1.16.

Yunfei Chu, Jin Xu, Qian Yang, Haojie Wei, Xipin Wei, Zhifang Guo, Yichong Leng, Yuanjun Lv, Jinzheng He, Junyang Lin, et al. 2024. Qwen2-audio technical report. *arXiv preprint arXiv:2407.10759*.

CrewAI. 2024. CrewAI: The leading multi-agent platform. https://www.crewai.com/. Accessed: 2025-07-02.

deepset. 2025. Haystack: Open source AI framework for building LLM-powered applications. https://haystack.deepset.ai/. Accessed: 2025-07-02.

Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. 2024. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

IBM. 2023. watsonx: Ai and data platform. https://www.ibm.com/products/watsonx. Accessed: 2025-07-02.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. 2024. Llama guard: Llm-based input-output safeguard for human-ai conversations, 2023. *URL https://arxiv. org/abs/2312.06674*.

Mladjan Jovanovic and Peter Voss. 2024. Towards incremental learning in large language models: A critical review. *Preprint*, arXiv:2404.18311.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An llm compiler for parallel function calling. arxiv. *arXiv preprint arXiv:2312.04511*.

LangChain. 2024. LangGraph: Multi-Agent Workflows. https://blog.langchain.com/langgraph-multi-agent-workflows/. Accessed: 2025-07-02.

Y Liang, C Wu, T Song, W Wu, Y Xia, Y Liu, Y Ou, S Lu, L Ji, S Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. arxiv 2023. *arXiv preprint arXiv:2303.16434*.

Jerry Liu. 2022. Llamaindex: Data framework for llm applications. https://github.com/run-llama/llama_index. Licencia MIT.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. 2023. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*.

LlamaIndex. Introduction to Workflows. https://docs.llamaindex.ai/en/stable/understanding/workflows/. Accessed: 2025-07-02.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, et al. Smolvlm: redefining small and efficient multimodal models (2025). *arXiv preprint arXiv:2504.05299*.

Microsoft. 2024. Semantic Kernel: Lightweight, open-source SDK for AI agents. https://learn.microsoft.com/en-us/semantic-kernel/overview/. Accessed: 2025-07-02.

Qdrant Tech. 2023. Qdrant: Open-source vector database and similarity search engine. https://qdrant.tech/. Accessed: 2025-07-15.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Hugging Face TB Research Team. 2025. Smolvlm2: Bringing video understanding to every device. Hugging Face Blog.

Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. 2024. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, et al. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *arXiv preprint arXiv:2412.13663*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions, 2023. *URL https://arxiv.org/abs/2306.02224*, 3.