ChatGPT

Comparison of the Soldier Turn Pipeline vs Parlant's Framework

Introduction

The **Soldier** document describes an 11-phase pipeline for processing a single user message. Each phase performs a specific function such as context loading, intent sensing, retrieval of rules, scenario orchestration, tool execution, response planning, generation, and guardrail enforcement. In contrast, **Parlant** presents itself as a *conversational AI framework* that emphasises instruction-following through **guidelines** (condition–action pairs), **journeys** (state machines for multi-turn flows), and **tools** attached to guidelines. Parlant's architecture revolves around three main modules – **Guideline Proposer**, **Tool Caller** and **Message Generator** – and uses **Attentive Reasoning Queries (ARQs)** to guide the language model through structured reasoning. The following sections compare Soldier's pipeline to Parlant's framework, highlighting similarities, differences, and potential opportunities for improvement.

High-level architectural differences

Aspect	Soldier pipeline	Parlant framework	Observations
Design principles	Emphasises multi-tenant architecture, stateless pods, deterministic control, schema-driven data and observability. LLMs are treated as sensors and judges but not policy engines.	Designed for reliability and control; developers define **guidelines** (When → Then rules) and attach **tools** to guidelines. Parlant emphasises structured reasoning via ARQs and provides **journeys** for multi-turn flows [1].	Both frameworks seek predictable, auditable behaviour. Soldier focuses on fine-grained phase separation, whereas Parlant bundles logic into fewer modules and relies on guidelines/ARQs for control.

Aspect	Soldier pipeline	Parlant framework	Observations
Number of phases	11 phases: identification & context loading, situational sensor, customer data update, retrieval & selection, rule selection, scenario orchestration, tool execution, response planning, generation, enforcement & guardrails, persistence/audit.	3 core modules: **Guideline Proposer**, **Tool Caller** and **Message Generator** [2]. Tool caller and proposer may iterate; message generator performs the final synthesis [3].	Soldier's pipeline is more granular and includes separate retrieval, scenario orchestration and guardrail enforcement phases; Parlant's pipeline is more compact, with reasoning encapsulated in dedicated modules and enforced through guidelines. More phases can introduce additional latency and complexity.
Control mechanism	Soldier retrieves rules and scenarios using vector/ lexical retrieval and selects them via LLM filter and static logic. Policies and guardrails are enforced after generation by deterministic expressions and a judge LLM.	Parlant uses **guideline proposition** to score and activate guidelines based on context; guidelines include conditions and actions [4]. Tools are called only when their associated guideline is active, and the **message generator** synthesises the final response with ARQ-based instructions [5].	Parlant's control happens *before* response generation: only relevant guidelines and tools are loaded, reducing cognitive load [6]. Soldier's guardrails occur *after* generation, which may require regenerations and extra LLM calls.

Aspect	Soldier pipeline	Parlant framework	Observations
Tool handling	Tools are bound to rules and scenario steps; Soldier resolves variables and executes tools for missing data in Phase 7.	Tools are always associated with guidelines; the tool caller determines whether to run a tool based on active guidelines [7] [8]. Parlant prevents duplicate calls and enforces self-contained tool invocations [9].	Parlant's tool mechanism is tightly coupled with guidelines and emphasises error prevention and efficiency. Soldier could adopt guideline-based tool triggers to reduce unnecessary tool invocations and improve explainability.
Flow orchestration	Soldier uses **scenarios** as state machines. Phase 6 decides whether to start, continue or pause a scenario and determines contribution types (ASK, INFORM, CONFIRM).	Parlant offers **journeys**, which are state diagrams specifying states and transitions; journeys permit flexible skipping or revisiting of states so that conversations adapt to user behaviour [10].	Both frameworks implement orchestrated flows. Parlant emphasises adaptive traversal of states; Soldier's scenarios may benefit from similar flexibility and explicit state diagrams to improve clarity and reuse.
Reasoning guidance	Soldier uses LLMs as a situational sensor to extract variables and detect intent (Phase 2) and as a generator (Phase 9). There is no explicit structured reasoning step; optional LLM rule filters exist in Phase 5.	Parlant employs **Attentive Reasoning Queries (ARQs)** within each module: the guideline proposer uses ARQs to score guidelines, the tool caller uses ARQs to choose tools and parameters, and the message generator uses ARQs to synthesise responses [5]. ARQs reinstate critical instructions and facilitate step-wise reasoning [11].	ARQs improve instruction-following and reduce hallucinations. Soldier could incorporate structured reasoning prompts or ARQs to increase accuracy and reduce misaligned outputs.

Aspect	Soldier pipeline	Parlant framework	Observations
Enforcement & compliance	Soldier has a dedicated **enforcement & guardrails** phase (Phase 10) that collects constraints, evaluates deterministic expressions, and optionally asks an LLM to judge subjective rules. Violations can trigger regeneration or fallback.	In Parlant, compliance is enforced primarily by the **guideline-based** architecture. Only active guidelines and authorised tools are presented to the message generator, and the message generator is instructed to avoid offering services not included in its context [12].	Soldier's explicit guardrail phase provides strong policy enforcement, but it also adds another LLM call and potential latency. Parlant bakes compliance into the earlier stages, which may be more efficient; however, it assumes guidelines capture all hard constraints.
Persistence & observability	Final phase persists session state and customer data, records a `TurnRecord` for auditing and emits metrics/traces.	Parlant automatically stores tool results in the session and emphasises explainability through guideline tracking and ARQs [13] [14].	Both frameworks support observability; Soldier explicitly stores audit trails, whereas Parlant focuses on explaining why guidelines matched. Combining these strengths could yield comprehensive audit and explainability.

Detailed step-by-step comparison

The following matrix evaluates each Soldier pipeline phase against Parlant's nearest equivalent and notes potential efficiency or compliance advantages.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 1: Identification & Context Loading	Resolves tenant/agent/customer/session, loads `SessionState` and customer data, builds `TurnContext`.	During guideline proposition, Parlant receives interaction history, agent profile and domain lexicon [15].	Both frameworks initialise context before reasoning. Soldier's multi-tenant and stateless design is robust, but Parlant demonstrates that storing only the relevant domain lexicon and profile may reduce context size. Soldier could optimise by loading only relevant slices of customer data for each turn.
Phase 2: Situational Sensor	Uses an LLM to perform schema-aware extraction, intent detection and tone analysis to produce a `SituationalSnapshot`.	Parlant relies on the **Guideline Proposer** to evaluate which guidelines apply; the proposer uses ARQs to score guidelines based on conversation history and agent profile [16].	Soldier's sensor extracts variables generically; Parlant uses targeted ARQs, which improves reasoning efficiency and reduces hallucinations. Soldier could adopt ARQ-like prompts to extract only variables relevant to potential guidelines rather than broad free-form extraction.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 3: Customer Data Update	Validates extracted variables against schemas, updates in-memory `CustomerDataStore` and marks persistent updates for final phase.	Parlant automatically persists tool results in session for later use [13].	Both update data. Soldier's explicit validation ensures type-safety; Parlant's automatic persistence emphasises convenience. A missed opportunity in Soldier is to propagate tool results and variable updates directly into the retrieval and scenario logic to avoid redundant extraction.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 4: Retrieval & Selection	Computes embeddings and lexical features; performs hybrid retrieval to find candidate rules and scenarios; applies selection strategies such as adaptive-k or entropy.	Parlant does **not** rely on vector search; guideline conditions are matched through the guideline proposer using ARQ scoring [16].	Soldier's embedding retrieval can be powerful for large rule bases but adds computational overhead and may retrieve irrelevant rules. Parlant's explicit conditions avoid embedding look-ups and make matching transparent. Soldier could incorporate structured conditions or keyed retrieval to reduce reliance on vector operations.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 5: Rule Selection	Pre-filters rules by scope and lifecycle; optionally uses an LLM to classify rules as APPLIES/ NOT_RELATED/UNSURE; expands via relationships.	Parlant's guideline proposer computes a 1-10 applicability score for each guideline, accounting for temporal continuity and re-activation [17].	Parlant's scoring approach unifies selection and prioritisation, whereas Soldier splits retrieval and selection into separate LLM calls. The optional rule filter in Soldier may suffer from misclassifications; using ARQs to generate a structured applicability score could improve precision and reduce chain-of-thought overhead.
Phase 6: Scenario Orchestration	Manages lifecycle decisions (start, continue, pause, complete, cancel), controls step transitions and determines contributions (ASK/ INFORM/CONFIRM).	Parlant's **Journeys** provide state diagrams with chat and tool states; agents can adaptively skip, revisit or jump ahead based on user behaviour [10].	Both frameworks use state machines, but Parlant emphasises flexibility to improve user experience. Soldier's scenario engine may be more rigid; adopting journey-style diagrams and allowing adaptive transitions could reduce friction and improve efficiency.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 7: Tool Execution	Collects tool bindings from rules and scenarios, resolves variables from session and executes tools for missing variables.	Parlant's **Tool Caller** runs only tools whose guidelines are active, supports proactive calls, prevents duplicate calls and ensures self-contained parameters [18].	Parlant's guideline-based gating avoids unnecessary tool calls and duplicates. Soldier's tool execution could erroneously call tools even when conditions are partially met; aligning tool invocations with explicit guideline conditions would improve reliability and reduce compute.
Phase 8: Response Planning	Determines global response type (ask/ answer/mixed/escalate), merges scenario contributions and injects constraints such as must-include or must-avoid.	Parlant does not have a separate planning module; the **Message Generator** uses active guidelines, tool results and built-in instructions to synthesize a response [3].	Soldier's explicit planning step offers fine control over response type but adds an extra stage. Parlant's message generator handles prioritisation within a single LLM call, benefiting latency. Combining planning and generation into one structured ARQ call could reduce runtime costs in Soldier.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 9: Generation	Builds a prompt from the `ResponsePlan`, rules, variables and glossary; generates response and appends semantic categories; applies post-formatting for channels.	Parlant's **Message Generator** uses ARQs and instructs the LLM to adhere to active guidelines and tool results while avoiding hallucinations [12].	Both rely on LLMs. Parlant's generator is highly specialised; it inherits applicability scores and avoids unauthorized services. Soldier's generation step could benefit from ARQs to enforce guideline priorities and reduce hallucinations.
Phase 10: Enforcement & Guardrails	Collects matched hard constraints and global constraints; uses deterministic expressions for objective rules and an LLM judge for subjective rules; may perform relevance/grounding checks and regenerate if needed.	Parlant bakes compliance into guideline selection and message generation; unauthorized actions are prevented because tools cannot be called unless their guideline is active and the message generator avoids offering services not in context [12].	Soldier's enforcement provides robust safety and supports explicit policy expressions; however, it adds another LLM call. Parlant's integrated compliance reduces latency but relies on guidelines capturing all constraints. Soldier could reserve the enforcement step for truly critical checks and gradually shift simpler constraints into earlier phases.

Soldier phase / feature	Soldier implementation and evidence	Parlant equivalent (evidence)	Evaluation of efficiency and misses
Phase 11: Persistence, Audit & Output	Persists session state and customer data, records a `TurnRecord` with full audit trail and emits metrics and traces.	Parlant automatically keeps tool results in session and emphasises explainability through guideline tracking and ARQs [13] [14].	Both frameworks preserve state and support auditing. Soldier's explicit metrics/ tracing is advantageous. Parlant's focus on explainability (through ARQs and guideline tracking) is a good complement. Combining both could enhance debugging and compliance audits.

Summary of misses and recommendations

- **Structured reasoning (ARQs) is absent in Soldier.** The Soldier pipeline uses generic LLM calls for intent sensing, rule filtering and generation. Parlant's Attentive Reasoning Queries have been shown to improve instruction-following performance and reduce hallucinations, achieving higher success rates than chain-of-thought approaches [16] [19]. Incorporating ARQ-style prompts or other structured reasoning techniques into phases such as the situational sensor, rule selection and generation could enhance reliability.

- **Guideline-based control vs. vector retrieval.** Soldier relies on embedding retrieval and LLM filters to choose applicable rules. Parlant instead uses natural-language guideline conditions and a scoring mechanism to activate them [4] [17]. Embedding retrieval adds computational cost and may surface irrelevant rules. Shifting toward explicit conditions or hybridising retrieval with structured conditions would reduce noise and improve explainability.

- **Tool invocation gating.** In Soldier, tools are executed after rule selection and scenario orchestration, but the pipeline does not explicitly gate tool calls based on guidelines. Parlant's tool caller ensures that tools run only when their associated guideline is active and prevents duplicates [18]. Adopting similar gating would prevent unnecessary tool calls and help align business logic with conversation context.

- **Flexible journey orchestration.** Soldier uses scenarios with lifecycle states, but it is not clear whether users can flexibly skip or revisit steps. Parlant's journeys emphasise adaptive traversal,

allowing agents to skip or jump ahead based on the conversation [10]. Implementing a more flexible state machine could improve user experience and reduce rigid conversational flows.

- **Integration of compliance into earlier phases.** Soldier's dedicated enforcement phase ensures policy compliance and is valuable for high-risk domains; however, it adds latency and may trigger regenerations. Parlant's compliance is built into the guideline selection and message generation steps [12]. Where possible, incorporate simple constraints into rule/scenario definitions or guidelines so that the generator does not propose invalid actions, reserving the final enforcement phase for non-negotiable policies.

- **Observability and explainability.** Soldier produces a `TurnRecord` and metrics; Parlant provides explainability via guideline tracking and ARQs [14]. Combining explicit audit logs with guideline-level explanations would give developers and auditors both macro-level and micro-level views of agent behaviour.

Conclusion

The Soldier pipeline is comprehensive and modular, with explicit phases for context loading, data updates, retrieval, rule selection, orchestration, tool execution, response planning, generation, enforcement, and persistence. Its design enables strong policy compliance and detailed auditing. **Parlant**, however, achieves reliability with fewer modules by treating guidelines as first-class citizens, using ARQs to guide reasoning, and tightly coupling tool usage to guideline conditions. The ARQ-based guideline proposer, tool caller and message generator collectively control what information enters the LLM, ensuring high instruction-following accuracy and reducing hallucinations [5] [3].

When comparing the two, Soldier's main misses relative to Parlant include the lack of structured reasoning prompts, heavy reliance on embedding retrieval, absence of guideline-gated tool invocation, and potentially rigid scenarios. Addressing these gaps—by adopting ARQ-like reasoning, shifting toward condition-based guidelines, gating tool calls, and building more flexible state machines—could improve efficiency, reliability and user experience while preserving Soldier's strong enforcement and auditing capabilities.

---

[1] [10] raw.githubusercontent.com
https://raw.githubusercontent.com/emcie-co/parlant/develop/docs/concepts/customization/journeys.md

[2] [3] [5] [8] [9] [11] [12] [15] [16] [17] [18] [19] AttentiveReasoningQueries.pdf
https://www.parlant.io/files/AttentiveReasoningQueries.pdf

[4] [6] [14] raw.githubusercontent.com
https://raw.githubusercontent.com/emcie-co/parlant/develop/docs/concepts/customization/guidelines.md

[7] [13] raw.githubusercontent.com
https://raw.githubusercontent.com/emcie-co/parlant/develop/docs/concepts/customization/tools.md