A Constraint-Driven Conversational Architecture for Staged Task Interaction Using Large Language Models

Joseph Benjamin Ilagan, Jose Ramon Ilagan
Ateneo Business Insights Laboratory for Development (BUILD)
Ateneo de Manila University, Quezon City, Philippines 1108

***Abstract*—Large language models (LLMs) enable flexible conversational interfaces, but remain difficult to deploy in structured, staged tasks that require controlled progression, bounded information disclosure, and task validity. Prompt-based control is inherently probabilistic and has been shown to degrade under multi-turn interaction, leading to premature solution disclosure, stage skipping, loss of state coherence, and constraint violations. This study presents a constraint-driven conversational architecture that separates probabilistic language generation from deterministic task governance. An external control layer manages dialogue state, stage transitions, and constraint enforcement, while a simulation layer represents task logic independently of the LLM. We instantiate the architecture in the context of customer discovery tasks to illustrate how staged processes and bounded disclosure can be operationalized without embedding task logic directly into prompts. This work focuses on architectural design and control mechanisms rather than outcome evaluation, offering a reusable architectural pattern for LLM-driven conversational systems that must preserve staged progression, enforce constraints, and prevent premature disclosure during multi-turn interaction.**

***Keywords*—Large language models; conversational systems; task-oriented dialogue; deterministic control; constraint enforcement; staged interaction**

I. Introduction

Large language models (LLMs) have enabled conversational interfaces for tasks that previously relied on scripted workflows or rule-based dialogue systems. However, when tasks require staged progression and procedural integrity, prompt-based control remains unreliable. Common failures include premature solution disclosure, stage skipping, uncontrolled transitions, loss of state coherence, and domain constraint violations. These failures limit use in settings where the process itself must be preserved.

Prior work has improved multi-turn interaction through prompting techniques and learned policies, but task governance is still often mediated through model behavior. This makes guarantees on state, constraints, and stage boundaries difficult to obtain.

This study proposes a constraint-driven conversational architecture that separates probabilistic language generation from deterministic task governance. An external control layer maintains dialogue state, regulates stage transitions, and enforces constraints, while a simulation layer represents task logic independently of the LLM to support runtime checks and bounded information disclosure. We instantiate the architecture in customer discovery tasks to illustrate staged interaction without embedding task logic directly into prompts. This work focuses on architectural design and control mechanisms rather than outcome evaluation.

It is important to clarify the nature of the contribution made in this work. The proposed architecture does not claim novelty in staged task representations, finite-state control, or deterministic dialogue management as isolated techniques. Related ideas appear in classical dialogue system architectures and in hierarchical control formulations for sequential decision-making [1], [2].

The contribution instead lies in repositioning deterministic task governance as a first-class architectural layer in LLM-driven conversational systems, where language generation is probabilistic and state coherence cannot be assumed. In this setting, staged progression, constraint enforcement, and bounded disclosure are treated as external guarantees instead of emergent properties of prompt design or model reasoning. This distinction is central to the architectural argument advanced in this study.

A. Research Questions

This study addresses the following research questions:

- RQ1. What system architecture can support structured, staged task interaction with LLM-based conversational interfaces, while preserving natural language flexibility?
- RQ2. How can dialogue state, stage transitions, and permissible system actions be represented and governed deterministically and not through prompt interpretation?
- RQ3. What runtime mechanisms enable enforceable constraints and bounded information disclosure, including integration of an external task logic or simulation layer independent of the LLM?

B. Contributions

This study makes the following contributions:

- We present a constraint-driven architecture for LLM-based conversational systems that separates language generation from deterministic task control for staged interaction.

- We define a dialogue state and control model that governs stage transitions, permissible system actions, and bounded information disclosure outside the LLM.
- We show how the architecture can be operationalized with an external simulation/control layer, using customer discovery as an illustrative domain.

II. Review of Related Literature

This section reviews prior work related to: 1) controlling large language model (LLM) behavior in multi-turn settings, 2) maintaining explicit state and constraints in structured tasks, and 3) architectural patterns that separate probabilistic language generation from deterministic task governance. The review is organized around control mechanisms (prompting and agents), externalization of state (memory, simulators, controllers), and system-level designs for constrained interaction.

A. Prompt Engineering for Task Control

Prompt engineering is a widely used mechanism for steering large language models (LLMs) in conversational systems without modifying model parameters. Surveys of prompting methods document a range of strategies, including instruction prompts, role prompting, and structured prompt templates intended to improve task adherence across diverse settings [3].

Several prompt engineering techniques have emerged to increase reliability in multi-turn interaction. Common approaches include explicitly stated behavioral constraints, stepwise instructions, and structured response formats that narrow the space of acceptable outputs. In practice, these methods remain inherently probabilistic because task logic and constraints are interpreted as opposed to enforced, and can degrade across longer conversations or under distribution shift [3].

Few-shot learning extends prompt-based control by providing a small number of exemplars that demonstrate the expected behavior [4]. In staged tasks, exemplars can illustrate valid responses and interaction norms, but they do not provide explicit guarantees on state persistence, bounded information disclosure, or correct stage transitions, particularly when user behavior deviates from the examples.

A related technique known as *chain-of-thought* (CoT) prompting encourages models to generate intermediate reasoning steps prior to producing a final answer [5]. CoT prompting can improve multi-step reasoning performance, but the reasoning trace remains part of the model output and not an external representation of task state. As a result, CoT does not by itself enforce task constraints or regulate progression in structured, staged interactions.

Overall, prompt-centric approaches provide flexible and low-overhead control of LLM behavior, but task control, constraint enforcement, and state management are encoded implicitly in natural language prompts. This implicit encoding makes it difficult to ensure consistent adherence to staged task structures or to prevent failure modes such as premature information disclosure and uncontrolled stage transitions in conversational systems.

B. Agentic and Tool-Augmented LLM Systems

Tool-augmented and agentic systems extend LLMs with actions such as retrieval, API calls, and tool usage. ReAct couples reasoning traces with action selection in an interleaved loop [6]. Such approaches improve task completion on interactive problems, but the policy is typically still mediated through model generations, not an external controller.

Toolformer proposes self-supervised learning of tool usage, allowing models to invoke tools through simple APIs [7]. These frameworks motivate modularity, yet many implementations still treat constraints, stage boundaries, and permissible transitions as soft rules embedded in prompts or tool schemas, not explicit governance layers.

C. State, Memory, and Environment-Mediated Interaction

Long-horizon task interaction requires stable representations of state, goals, and constraints. Existing work spans symbolic dialogue management, hybrid neural-symbolic systems, and environment-mediated learning, where an external environment provides dynamics and feedback [2]. Related lines of work use simulators for training, evaluation, or controlled interaction loops [8]. A key distinction for this study is treating simulation/control as a persistent runtime layer that governs progression, not only as a training or evaluation artifact.

D. Architectural Patterns for Conversational Systems

Architectural perspectives on dialogue systems emphasize modular separation of components such as NLU, dialogue state tracking, dialogue policy, and NLG [1]. These patterns motivate separation of concerns, but they do not fully anticipate LLM-first dialogue generators where the generator can blur boundaries among state, policy, and surface realization.

E. Synthesis and Gap

Across the strands reviewed in this section, task governance is predominantly implicit and probabilistic, even when external tools or environments are introduced. Constraints, stage boundaries, and permissible transitions are typically specified as prompt rules, learned behaviors, or conventions mediated through language generation, rather than enforced through explicit deterministic control structures.

Although deterministic state machines and hand-crafted control policies are well-known in task-oriented dialogue more broadly, these designs largely predate LLM-first conversational architectures and are rarely positioned, in the LLM-centric literature reviewed here, as the primary mechanism for guaranteeing staged progression, bounded disclosure, and constraint satisfaction during runtime interaction.

This work responds to this gap by treating deterministic task governance as a first-class architectural abstraction. The proposed constraint-driven conversational architecture explicitly separates probabilistic language generation from deterministic task control, elevating stage representation, enforceable transition logic, and constraint checks to core components that regulate what the LLM may say and when (see Table I).

TABLE I. COMPARISON OF TASK GOVERNANCE APPROACHES IN LLM-BASED CONVERSATIONAL SYSTEMS

Property	Prompt-centric	Agentic	Proposed (Constraint-driven)
Stage enforcement	Probabilistic	Emergent	Deterministic
State representation	Implicit	Partial	Explicit
Disclosure control	Prompt-level	Tool-schema	Controller-enforced
Transition authority	Model	Model	Controller
Recovery on violation	None/retry	Model-driven	Deterministic fallback
Task logic location	In prompt	Mixed	External layer
Cross-domain reuse	Low	Moderate	High (config-driven)

III. ARCHITECTURE

A. Conceptual Model for Deterministic Task Governance

The architecture proposed in this study is guided by a simple conceptual model that distinguishes between *probabilistic language generation* and *deterministic task governance*. While large language models (LLMs) are well suited for producing fluent, context-sensitive responses, they do not provide intrinsic guarantees on task progression, state persistence, or constraint satisfaction across multi-turn interactions. These properties must, therefore, be enforced outside the language model.

Procedural integrity, as used in this study, refers to the property that stage position, transition validity, and disclosure boundaries are maintained by deterministic system components throughout the interaction lifecycle, independent of model behavior.

At the core of the model is the treatment of task state as a first-class system artifact. Dialogue state, stage membership, and permissible actions are represented explicitly and persist independently of conversational turns. Instead of inferring state from natural language alone, the system maintains an external representation of where the interaction currently resides within a staged task structure. This enables consistent reasoning about what information may be disclosed, which actions are valid, and which transitions are allowed at any point in the interaction.

A second principle is the separation of task logic from surface realization. Task constraints, progression rules, and evaluation conditions are encoded in a simulation or control layer that operates deterministically. The role of the LLM is restricted to generating natural language realizations that are conditioned on, and bounded by, decisions made in this control layer. This separation prevents task logic from being embedded implicitly in prompts, where it would otherwise be interpreted probabilistically and remain difficult to enforce over extended conversations.

Finally, grounding is achieved through controlled exposure to task-relevant context in place of generic conversational guidance. System responses are generated with reference to the current task state and its associated constraints, allowing feedback and guidance to be tailored to the user's position within the staged process. This design supports specificity and consistency in guidance while preserving the flexibility of natural language interaction.

Together, these principles define a constraint-driven conversational model in which LLMs function as flexible linguistic interfaces governed by explicit, deterministic task structures. This conceptual framing underpins the architectural design presented in the following section.

B. Architectural Contribution Clarification

The architecture proposed in this study should be understood as a governance pattern and not a new dialogue algorithm. Its primary contribution is the explicit separation of conversational fluency from procedural authority in LLM-based systems. Unlike prompt-centric or agentic designs, where task control is mediated through model interpretation or reasoning traces, the proposed architecture assigns exclusive authority over task state, stage transitions, and disclosure boundaries to deterministic system components. This framing addresses failure modes that arise specifically in LLM-first conversational settings and are not adequately handled by classical dialogue pipelines or workflow engines when language generation is open-ended and user-driven.

C. Overall System Architecture

This subsection provides a high-level view of the proposed conversational architecture, outlining the primary components and their interactions (see Fig. 1). The design emphasizes a clear separation between probabilistic language generation and deterministic task governance to support structured, staged interaction.

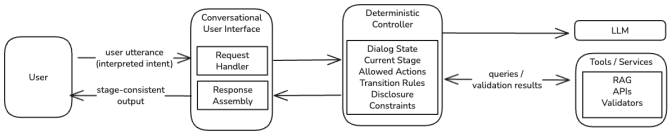

Fig. 1. Constraint-driven conversational architecture with deterministic task governance. The controller governs dialogue state, stage transitions, and disclosure constraints, invoking the LLM and external tools as bounded services.

D. Dialogue State Representation

Dialogue state is represented as an explicit, structured data object maintained by the control layer across conversational turns. The state encodes the current task stage, completed and pending artifacts, validated inputs, and flags relevant to progression and constraint enforcement. This representation persists independently of the conversation transcript and is not inferred solely from natural language interaction.

State updates occur through deterministic events, not through linguistic interpretation alone. User inputs may trigger validation checks or proposed updates, but acceptance of new state values depends on evaluation against task specifications defined in the control or simulation layer. This prevents ambiguous or partial inputs from implicitly advancing the task.

Separating dialogue state from conversational history enables consistent reasoning about task position even when the

interaction includes digressions, reformulations, or clarification requests. It also supports recovery from invalid inputs without relying on the language model to reconstruct prior context.

Dialogue state is treated as a persistent system artifact that may be stored across turns or sessions using any suitable state storage mechanism (e.g., in-memory stores, databases, or session services), which is intentionally left abstract in this architecture.

E. Stage Transitions and Control Logic

Structured tasks are modeled as staged processes governed by explicit transition rules rather than inferred conversational flow. Each task defines a finite set of stages, allowable transitions between stages, and conditions under which transitions may occur. These elements are treated as part of the task specification and are evaluated by a deterministic control layer at runtime.

Stage definitions are external to the language model and may be configured per task or use case. A stage specification encodes the set of valid stages, permitted successor stages, entry and exit conditions, and constraints on permissible system actions and disclosures at each stage. This allows the same conversational runtime to support different staged tasks without modifying the underlying control logic.

Transition evaluation is performed exclusively by the control layer. User inputs and LLM-generated responses may propose or suggest progression, but transitions are only executed when explicit transition conditions are satisfied. These conditions may include completion of required task elements, validation of user inputs against the task logic or simulation layer, or satisfaction of domain-specific constraints.

Invalid transitions, such as premature advancement or attempts to bypass required stages, are blocked by design. When a transition request is rejected, the system remains in the current stage and generates responses that are consistent with the active stage's disclosure and action constraints. This ensures that task progression remains valid even under adversarial, ambiguous, or unexpected user behavior.

By separating transition authority from language generation, the architecture enforces staged progression deterministically while preserving conversational flexibility. The LLM is responsible for surface realization and interaction quality, whereas the control layer maintains procedural integrity throughout the task lifecycle.

F. Constraint Enforcement and Bounded Disclosure

Constraint enforcement is implemented through stage-specific policies managed by the control layer. These policies define which system actions are permitted, which information may be disclosed, and which responses are prohibited at a given point in the interaction.

While avoiding encoding constraints as prompt instructions alone, the controller regulates what contextual information is provided to the language model and which response types are allowed. Future-stage artifacts, evaluative conclusions, or solution templates are withheld until the task reaches the appropriate stage, ensuring that the model cannot disclose information it has not been authorized to access.

When a user request violates active constraints, the controller blocks the corresponding action and triggers a recovery response. The recovery response is generated within the bounds of the current stage and redirects the interaction without advancing state or revealing restricted information. This preserves procedural integrity under adversarial, ambiguous, or premature user requests.

G. Integration of the Simulation or Task Logic Layer

Task logic is implemented in a simulation or rule-based layer that operates independently of the language model. This layer encodes domain-specific requirements such as completeness conditions, validity checks, dependency constraints, and progression criteria.

User inputs requiring structured interpretation are routed to the simulation layer for evaluation. The simulation returns deterministic signals such as completion flags, constraint violations, or required revisions. These signals are consumed by the controller and translated into bounded response contexts for the language model.

This separation allows domain logic to evolve without modifying prompt templates or conversational behavior. It also reduces reliance on the language model for correctness judgments, limiting its role to explanation and feedback consistent with the simulation's outputs.

H. Interaction Flow

Each interaction cycle follows a fixed control sequence. User input is first received and recorded without immediate state mutation. The controller evaluates whether the input proposes a state transition, requests restricted information, or requires validation against task logic.

If validation is required, the input is passed to the simulation layer. The resulting signals are incorporated into the dialogue state and used to determine the next permissible system action. Only after this decision is made does the controller construct a bounded response context for the language model.

The language model generates a natural language response within the constraints imposed by the controller. Conversational output does not directly modify task state. State updates occur only through explicit controller approval, ensuring that language generation cannot unilaterally advance task progression.

Algorithm 1 summarizes the deterministic runtime loop used to govern staged progression and disclosure.

The CHECKRESPONSE step in Algorithm 1 verifies that the LLM-generated response r_t does not reference disallowed disclosures or propose actions outside $\mathcal{A}_t$. Verification may be implemented through keyword filtering, structural pattern matching, or a secondary classifier. When a violation is detected, RECOVERYRESPONSE generates a stage-consistent fallback that redirects the user without modifying state or revealing restricted information.

Algorithm 1 Deterministic Controller Loop for Staged Conversational Tasks

Inputs: user message u_t, current state s_t, task spec $\mathcal{T}$, validator/simulator $\mathcal{V}$, LLM $\mathcal{M}$
Output: response text r_t, updated state s_{t+1}

$i_t \leftarrow \textsc{ClassifyIntent}(u_t, s_t, \mathcal{T})$ ▷ e.g., ask, revise, request_restricted, propose_transition
$x_t \leftarrow \textsc{ExtractStructuredInputs}(u_t, s_t, \mathcal{T})$ ▷ optional parsing
$v_t \leftarrow \mathcal{V}(x_t, s_t, \mathcal{T})$ ▷ returns completion, violations, missing fields
$\mathcal{A}_t \leftarrow \textsc{ComputeAllowedActions}(i_t, v_t, s_t, \mathcal{T})$
$\mathcal{D}_t \leftarrow \textsc{ComputeAllowedDisclosures}(s_t, \mathcal{T})$
$c_t \leftarrow \textsc{BuildBoundedContext}(s_t, v_t, \mathcal{A}_t, \mathcal{D}_t)$
$r_t \leftarrow \mathcal{M}(c_t)$
$ok \leftarrow \textsc{CheckResponse}(r_t, \mathcal{A}_t, \mathcal{D}_t)$
if $ok =$ **false then**
  $r_t \leftarrow \textsc{RecoveryResponse}(s_t, v_t, \mathcal{A}_t)$
end if
$s_{t+1} \leftarrow \textsc{CommitStateUpdate}(s_t, u_t, v_t, r_t, \mathcal{T})$
return r_t, s_{t+1}

IV. Illustrative Instantiation: Customer Discovery as a Staged Conversational Task

This section presents an illustrative instantiation of the proposed architecture using customer discovery as an example of a staged conversational task. The purpose is to demonstrate how deterministic task governance, bounded disclosure, and stage-aware interaction are operationalized in a concrete domain, not to evaluate task outcomes or user performance.

A. Task Structure and Stages

Customer discovery is modeled as a staged conversational task in which progression is constrained by explicit requirements at each stage. The staged structure reflects the need to prevent premature solution formulation while ensuring that users engage with each step of the discovery process in sequence (see Fig. 2).

In the illustrative instantiation, the task is decomposed into a finite set of stages, such as problem articulation, hypothesis formulation, interview planning, evidence gathering, and reflection. Each stage is associated with a well-defined set of permissible system actions, allowable disclosures, and completion conditions. These definitions are external to the language model and are maintained by the control layer as part of the task specification.

Stage progression is not inferred from conversational cues alone. Instead, advancement occurs only when the control layer determines that the conditions for exiting the current stage have been satisfied. For example, a transition from problem articulation to hypothesis formulation may require that the user has provided a sufficiently specific problem statement, as validated by the task logic or simulation layer. Attempts to advance without meeting these conditions are rejected, and the system remains within the current stage.

Treating stages as explicit system entities as opposed to conversational conventions ensures that task structure is preserved across multi-turn interaction. This approach allows conversational flexibility within a stage while preventing stage skipping, uncontrolled transitions, or disclosure of information that is inappropriate for the user's current position in the task.

TABLE II. Example Stage Specification for Customer Discovery under Deterministic Governance

Field	Specification (illustrative)
Stages	S1 Problem articulation; S2 Hypothesis formulation; S3 Interview planning; S4 Evidence gathering; S5 Reflection
State variables	`stage_id`; `artifacts` (problem_statement, hypotheses, interview_questions, notes); `flags` (validated, ready_to_advance); `violations`
Required artifacts	S1: problem statement draft; S2: testable hypothesis; S3: interview guide; S4: evidence log; S5: updated belief + next actions
Allowed actions	Ask clarification; critique current artifact; request missing fields; summarize progress; propose next micro-step within stage; attempt transition (controller decides)
Disallowed disclosures	No downstream templates or "answers" for future stages; no solution proposals before S5; no auto-advancing to S3+ without required artifacts
Exit criteria	S1: problem statement meets specificity checks; S2: hypothesis is falsifiable and linked to problem; S3: interview guide covers objectives and avoids leading questions; S4: evidence log meets completeness; S5: reflection includes decision and rationale
Validator outputs	`completion_flag`; `missing_fields`; `violations`; `allowed_actions`; `allowed_disclosures`

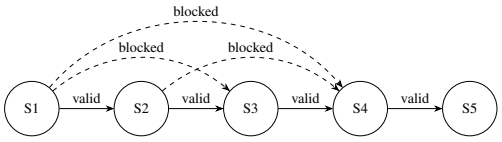

Fig. 2. State transition diagram for the customer discovery instantiation. Solid arrows indicate controller-approved transitions; dashed arrows indicate blocked skip attempts.

Table II provides an illustrative stage specification that externalizes exit criteria, allowed actions, and disclosure limits.

B. Grounded Feedback and Coaching Behavior

The architecture supports coaching behavior by making feedback generation stage-aware and state-dependent. Rather than prompting the model to "behave" as a coach across all turns, the control layer determines the allowable feedback type at the current stage and provides the LLM with stage-bounded context and permissible action options.

At runtime, the controller constructs a constrained response context that includes: 1) the current stage identifier, 2) relevant state variables and artifacts collected so far (e.g., problem statement drafts, hypotheses, interview questions), and 3) an explicit policy describing what the system may and may not disclose in the current stage. The LLM then produces natural language realizations of actions selected or permitted by the controller, such as asking a clarifying question, requesting missing inputs, offering critique on a draft artifact, or summarizing progress.

Grounding is operationalized through controlled access to task artifacts and simulator outputs. When user input requires validation or structured interpretation, the controller routes it to the task logic or simulation layer, which returns deterministic signals such as completion flags, constraint violations, or required revisions. The LLM is not expected to infer these signals from the conversation transcript. Instead, the controller passes the relevant outcomes as response constraints, ensur-

Worked trace: staged customer discovery with bounded disclosure (illustrative).

State (before): `stage_id=S1 (Problem articulation)`, `artifact.problem_statement=underspecified`, `ready_to_advance=false`.

User: "Can you help me write interview questions now so I can test this quickly?"
Intent: `request_future_stage_artifact (S3)`.
Validator: `completion_flag=false`; `missing_fields={target user segment, situation, observable pain}`; `violations={premature stage transition}`.

Allowed actions: ask clarification; critique current problem statement; request missing details.
Disallowed disclosures: interview question templates; hypothesis phrasing; discovery call scripts.

System (LLM realization within bounds):
"Before we design interviews, we need to be precise about the problem. Who specifically experiences this issue, in what recurring situation, and what concrete cost or friction do they face? Try rewriting the problem in one sentence using those elements."

State (after): `stage_id=S1`; `pending={target user segment, situation, pain}`; `ready_to_advance=false`.

Fig. 3. Illustrative trace showing deterministic blocking of premature interview design during customer discovery.

ing that coaching feedback remains consistent with the task specification.

This design yields two practical effects. First, feedback remains aligned with stage intent, since the controller restricts the response space to coaching behaviors appropriate to the current stage. Second, bounded disclosure is preserved, since the LLM does not receive privileged information or future-stage hints unless explicitly authorized by the controller. The result is a conversational experience that remains flexible in language while remaining disciplined in instructional sequencing.

Fig. 3 shows a concrete turn-level example of how stage skipping and premature disclosure are blocked at runtime. This example reflects early-stage customer discovery, where interview design is intentionally blocked until the problem is explicitly articulated.

C. Observed Failure Modes and Mitigations

The architecture is motivated by failure modes commonly observed in prompt-governed multi-turn interaction for staged tasks. In prompt-centric implementations, the model is expected to remember stage position, respect procedural boundaries, and withhold information until the appropriate time. In practice, these behaviors are unreliable under long-horizon conversation, ambiguous user prompts, or user attempts to accelerate the process.

A frequent failure mode is premature solution disclosure, where the model provides downstream answers (e.g., proposed solutions, scripts, or conclusions) before earlier stages have been completed. Another is stage skipping, where the model implicitly advances the task after a single user message, compressing intermediate stages or omitting required artifacts. Conversational drift is also common, where the interaction gradually shifts toward generic advice that is weakly related to the staged task, reducing coherence and diminishing the value of accumulated state. Finally, constraint violations can occur when the model produces outputs that conflict with domain rules, such as recommending actions that the task specification explicitly forbids at a given stage.

The proposed architecture mitigates these failure modes through deterministic governance. Premature disclosure is controlled through stage-specific disclosure policies that limit which information and actions are available to the LLM at each stage. Stage skipping is prevented by enforcing transition checks in the controller, so that advancement can only occur when explicit completion conditions are met. Drift is reduced by anchoring each turn to an explicit task state representation and by routing key evaluations to the simulation layer instead of relying on narrative consistency in model outputs. Constraint violations are handled through runtime checks and rejection of invalid actions, followed by controlled recovery responses that redirect the interaction without changing the active stage.

These mitigations are not framed as guarantees of correctness in the model's linguistic output. Rather, they provide guarantees about procedural integrity: stage position, transition validity, and bounded information disclosure are governed by deterministic system components, while the LLM remains responsible for conversational naturalness within the permitted space.

V. Discussion

A. Design Trade-Offs

The proposed architecture also differs from conventional workflow and orchestration systems commonly used for multistage task execution. Workflow engines and automation platforms are designed to coordinate deterministic system-to-system processes, typically assuming that task progression is driven by predefined triggers, API responses, or execution outcomes in place of open-ended human interaction.

In contrast, the architecture presented in this study addresses staged task progression in the presence of natural language interaction, where user input and model-generated responses introduce uncertainty and variability. The challenge is not merely sequencing actions, but governing conversational progression, state coherence, and information disclosure under probabilistic language generation.

While workflow systems excel at executing predefined pipelines, they do not address failure modes specific to LLM-driven interaction, such as premature solution disclosure, stage skipping, or conversational drift. The proposed control layer operates at a different abstraction level, mediating between conversational inputs and task logic to ensure that staged processes remain valid even when interaction is negotiated through language as opposed to enforced through execution constraints alone.

B. Relation to Prompt-Centric and Agentic Approaches

Recent agentic approaches to large language model (LLM) systems emphasize autonomous planning, action selection, and tool use driven largely by model-generated reasoning traces. In such systems, the LLM is often responsible for determining when to advance between steps, invoke tools, or terminate a task, with control logic embedded implicitly in prompts or emergent reasoning behavior.

The architecture proposed in this study adopts a deliberately different stance on agency. Rather than treating the LLM as an autonomous task manager, the model is positioned as a bounded conversational service whose outputs are mediated by an external control layer. While the LLM may propose actions or suggest progression, authority over dialogue state transitions, stage advancement, and permissible disclosures resides exclusively in deterministic system components.

This design choice reflects a prioritization of procedural integrity over autonomous flexibility. By externalizing task governance, the architecture avoids relying on probabilistic reasoning traces or prompt adherence to enforce staged progression. As a result, conversational fluency is preserved while guarantees over task structure, bounded disclosure, and valid transitions are maintained independently of model behavior.

The proposed approach is not intended to replace agentic systems in domains where autonomous planning is desirable. Instead, it targets settings where the integrity of staged processes is critical and where conversational interaction must remain flexible without allowing the model to unilaterally control task execution.

The distinction between agentic autonomy and deterministic governance is intentional. The proposed architecture does not attempt to improve the model's ability to plan or reason about task execution. Instead, it removes responsibility for procedural correctness from the model altogether. This design choice reflects a different objective than that of agentic systems. The goal is not autonomous task completion, but preservation of staged interaction integrity under conversational uncertainty. This difference in objective leads to a different allocation of responsibility across system components.

C. Generality and Transferability

The proposed architecture is not specific to customer discovery tasks. For example, an entrepreneurship education scaffold could gate hypothesis formulation until a problem statement meets specificity criteria, while a customer validation workflow could restrict solution proposals until sufficient interview evidence has been logged — both mirroring the staged structure demonstrated in Section IV.

Transferability is achieved through task configuration in place of architectural modification. Stages, constraints, and validation logic are defined externally and can be replaced without changing the conversational runtime. This allows the same system to support multiple task types with differing structures and domain rules.

The architecture therefore functions as a reusable governance layer for conversational systems instead of a domain-specific application design.

VI. Limitations and Future Work

A. Current Limitations

The architecture introduces additional system complexity relative to prompt-centric approaches. Explicit controllers, state representations, and simulation layers require greater upfront design effort and tighter integration among system components.

The approach also assumes that task structure can be sufficiently formalized. Domains with highly subjective evaluation criteria or weakly defined progression rules may require hybrid designs that combine deterministic checks with probabilistic judgment.

The architecture does not address optimization of language quality or persuasive effectiveness. Its focus is procedural integrity as opposed to conversational richness.

The deterministic governance model may also introduce rigidity in domains where task progression is inherently exploratory or where valid sequences are not fully specifiable in advance. Creative workflows, open-ended research tasks, or domains with subjective completion criteria may require hybrid designs in which constraint strictness is relaxed or dynamically negotiated.

B. Future Extensions

Future work may explore declarative task specification languages that allow stages, constraints, and transition logic to be defined without custom code. This would reduce the effort required to author new staged tasks.

Empirical evaluation of user experience and learning outcomes under deterministic governance is another natural extension. Comparative studies could examine differences between prompt-centric, agentic, and constraint-driven architectures.

Adaptive control policies may also be investigated, allowing constraint strictness or feedback style to vary based on observed user behavior while preserving deterministic stage boundaries.

VII. Conclusion

This study presented a constraint-driven conversational architecture for supporting structured, staged task interaction with large language models. The proposed design separates probabilistic language generation from deterministic task governance by introducing an external control layer responsible for dialogue state management, stage transitions, and constraint enforcement. By externalizing task logic and progression rules, the architecture avoids relying on prompt-based control alone to preserve procedural integrity during multi-turn interaction.

Through an illustrative instantiation in customer discovery tasks, the study demonstrated how staged processes and bounded information disclosure can be operationalized without embedding task logic directly into language prompts. The architecture enables conversational flexibility while maintaining explicit guarantees over task progression, permissible actions, and disclosure boundaries.

Rather than focusing on outcome evaluation, this work contributes an architectural pattern for LLM-driven conversational

systems that require structured interaction and enforceable task constraints. The proposed approach is intended to be reusable across domains where the integrity of staged processes is critical, and it provides a foundation for future work that may explore empirical evaluation, task configuration languages, or adaptive control policies built atop the same deterministic governance framework.

The central contribution of this work is the articulation of deterministic task governance as an architectural requirement for LLM-driven conversational systems, instead of a behavioral property to be induced through prompting or autonomous reasoning.

FUNDING

This work was supported by the Ateneo de Manila University Research Council under the University Research Council (URC) grant program, Grant No. URC-2026-05, awarded to Joseph Benjamin Ilagan.

ACKNOWLEDGMENT

The authors thank Maria Mercedes T. Rodrigo, Ph.D., the Department of Information Systems and Computer Science (DISCS) of the Ateneo de Manila University, the John Gokongwei School of Management (JGSOM), the Ateneo Business Insights Laboratory for Development (BUILD), and the Ateneo University Research Office (UREO) for institutional support and administrative assistance related to this work.

DECLARATION ON GENERATIVE AI

Generative artificial intelligence tools were used to support language refinement and drafting during manuscript preparation. All scientific content, system design decisions, arguments, and conclusions were developed by the human authors. The authors critically reviewed, edited, and take full responsibility for the accuracy, originality, and integrity of the manuscript.

REFERENCES

[1] D. Jurafsky and J. H. Martin, "Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition," 2009, publication Title: Prentice Hall Series in Artificial Intelligence.

[2] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/hash/f442d33fa06832082290ad8544a8da27-Abstract.html

[3] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, Sep. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3560815

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html?utm_source=transaction&utm_medium=email&utm_campaign=linkedin_newsletter

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models. arXiv 2022," *arXiv preprint arXiv:2210.03629*, vol. 10, 2023.

[7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *Advances in Neural Information Processing Systems*, vol. 36, pp. 68 539–68 551, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[8] Z. Zhang, D. Zhang-Li, J. Yu, L. Gong, J. Zhou, Z. Hao, J. Jiang, J. Cao, H. Liu, and Z. Liu, "Simulating classroom education with llm-empowered agents," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025, pp. 10 364–10 379. [Online]. Available: https://aclanthology.org/2025.naacl-long.520/