Beyond Generalist LLMs: Specialist Agentic Systems for Structured Code Workflow Execution

Harris Borman [*1] **Herman Wandabwa** [*1] **Fusun Yu** [1] **Sandeepa Kannangara** [1] **Justin Liu** [1] **Anna Leontjeva** [1] **Ritchie Ng** [1]

Abstract

Large Language Models (LLMs) have accelerated the adoption of software development agents, now widely available as Integrated Development Environment (IDE) extensions and standalone applications. While these agents are typically general-purpose, it remains unclear whether specialist agents justify their additional development effort. We investigate this question in the context of business process automation, focusing on the transformation of Business Process Model and Notation (BPMN) diagrams into executable agentic workflows. Since BPMN specifies explicit control-flow semantics, we focus on deterministic workflows in which a fixed process model and inputs uniquely determine the executed path. We introduce a specialist workflow for this task and compare it against generalist agents such as Roo and Cline. Our results show that the specialist solution produces agents that outperform generalist baselines by approximately 9–20 percentage points in tool-use exactness, 2–4× in penalty-adjusted latency, and 3× fewer tool-call errors, while reducing generation token cost by over 95% and eliminating repair iterations. We also find that generalist agents generate code inconsistently in both functionality and quality, limiting their suitability for industrial settings where reliability and maintainability are essential.

1. Introduction

The emergence of LLMs has accelerated the rise of autonomous software agents (Ferrag et al., 2025). These AI-driven agents now appear as IDE extensions and stand-alone no-code assistants, enabling even non-programmers to build simple applications within minutes (He et al., 2025). Most operate as generalists, leveraging foundational LLMs to perform a wide range of tasks. For example, an open-source coding assistant like Roo Code can plan, write, and debug code across domains directly in a developer's editor (Sapkota et al., 2025). Similarly, multi-agent frameworks such as FLOW, AFLOW, AutoGen, or MetaGPT coordinate several LLM agents with predefined roles to solve complex problems in a general way (Niu et al., 2025; Zhang et al., 2025b; Wu et al., 2024; Hong et al., 2024). These systems have been applied to tasks ranging from web browsing and data analysis to game design and UI creation (Fourney et al., 2024). Prior work has largely focused on what we define as "generalist systems", a system of agents that are able to complete a wide range of tasks, with architectures suited to free exploration of various ideas to complete a task in an unspecified manner (Sapkota et al., 2026). This adaptability has driven adoption, including IDE extensions such as Roo and Cline (Sapkota et al., 2025; Cline, 2025). These systems are useful because they can create a complete system from a single prompt with minimal user intervention, allowing users with limited technical knowledge to build a functioning system from a simple idea (Sapkota et al., 2026).

However, these systems have drawbacks. Their generalist design often demands extensive planning and increases token and cost overhead, especially during rapid or repeated development. They may also lack awareness of company-specific best practices, such as style guides or preferred methods. While experienced users can impose constraints and refine outputs, this remains imperfect and does not guarantee consistency across many generations, as discussed in Section 3. This inconsistency can increase technology debt and complicate future updates and maintenance (Aljohani & Do, 2025).

An alternative paradigm is what we define as a "specialist system". These systems use a well-defined and constrained agentic workflow designed for a specific class of tasks. Rather than asking an LLM to generate solutions from scratch, specialist systems encode expert knowledge into a templated workflow. Within this scaffold, the LLM per-

[1]Commonwealth Bank of Australia, Sydney, Australia. Correspondence to: Harris Borman <harris.borman@cba.com.au>, Herman Wandabwa <herman.wandabwa@cba.com.au>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

forms localised reasoning, making small, context-sensitive adjustments within predefined steps rather than constructing the full solution from first principles.

These systems usually require greater upfront effort because the workflow must be manually designed and validated. This makes them less suitable for short-lived use cases such as demos or proof-of-concept experiments. However, they are better suited to repeated or large-scale deployment. By producing more consistent outputs, they can reduce technical debt and simplify debugging and integration with external systems. Their structured design also enables tighter context management, allowing developers to control what information is exposed to the LLM, reduce unnecessary token usage, and improve performance at scale.

Recent studies show that although LLMs are increasingly equipped with extended context windows, their utilisation of this capacity remains uneven (An et al., 2025). Empirical evidence also suggests that performance tends to degrade as more of the context window is consumed (Modarressi et al., 2025; Laban et al., 2025). To address this, we propose a context management strategy that restricts the active context to the minimum information required for each subtask. In generalist systems, this is difficult because it requires prior knowledge of the information needed for each subtask during workflow construction, and such designs are often task-specific and do not transfer easily across domains. Restricting context in this way can reduce redundancy and improve model performance relative to systems that retain excess context indiscriminately, as shown in Section 4.3. Given that business processes can scale well beyond the complexity of our benchmark workflows, effective context management is critical for maintaining performance in large-scale deployments.

Efficient context management also offers potential cost benefits. This is especially important in business environments where standard operating procedures may evolve frequently during development and post-deployment phases and agentic workflow generation tools may be executed repeatedly for the same task. Under these conditions, reducing token volume can yield substantial savings (Mei et al., 2025; Laban et al., 2025). While the per-instance reduction may seem small, the cumulative impact at scale can be significant. This creates value for specialist workflows that use manually constructed context management to minimise the information exposed to the LLM in a targeted way while maintaining performance and reducing cost.

To operationalise these specialist workflows and the scoped context they enable, we adopt a modular code generation approach in which the source of decomposition is external to the model. Rather than discovering plans through prompts, roles, or library modules as in prior multiagent systems (e.g., AutoGen, MetaGPT, DSPy, SWE-agent) (Wu et al., 2024; Hong et al., 2024; Khattab et al., 2024; Yang et al., 2024), we compile an industry standard BPMN 2.0 process model into a ReAct style control graph. Each BPMN node provides a typed tool contract and a node local policy, while the control plane enforces branches and joins, scopes context per node, and applies contract derived runtime validation with targeted retries. This specifications-to-agent compilation yields an auditable workflow structure, minimises unnecessary context exposure, and avoids rediscovering plans at runtime. The design is motivated by three principles: constrained execution via BPMN-derived control flow, targeted context management that limits irrelevant information exposure, and modular decomposition that supports more reliable tool-level reasoning. Against this background, our study examines whether a specialist BPMN-grounded workflow can offer practical advantages over more generalist agentic systems. The following sections describe how this design is instantiated using BPMNs, outline the experimental methodology and benchmark design, and present results comparing specialist and generalist agentic workflows.

1.1. Business Processes

To ground our study, we focus on business process automation and adopt BPMN as the structural representation of workflows. BPMN is widely used in enterprise modelling and provides a practical interface through which non-technical users can specify process logic in a standardised form (Köpke & Safan, 2024; Nour Eldin et al., 2025; Toxtli & Li, 2025; Berti et al., 2024). This makes it a suitable domain for studying how LLM-based agentic systems can transform human-authored process specifications into executable workflows.

In this work, we consider the task of converting BPMN-defined workflows into operational agentic pipelines. This setting is appropriate for our comparison between specialist and generalist systems because BPMN provides an explicit and auditable representation of process structure, allowing us to assess how effectively each approach preserves intended workflow logic while supporting executable automation.

2. Literature Review

2.1. BPMN and Traditional Workflow Execution

Business Process Model and Notation (BPMN) is a standard for modelling structured business workflows. Its graphical notation is both human-readable and machine-executable, which enables organisations to define, automate, and monitor processes effectively (White, 2004; Chinosi & Trombetta, 2012; Dumas et al., 2018). Traditional BPMN engines operate in deterministic, rule-based settings involving human tasks, service calls, and decision gateways (Weske, 2019).

While robust in predictable environments, they provide limited support for dynamic or context-sensitive decision-making (Van Der Aalst et al., 2020). Prior work has tried to address this through agent-based automation (Wooldridge & Jennings, 1995), adaptive workflow systems (Reichert & Weber, 2012), context-aware frameworks (Rosemann et al., 2008), and decision-centric models (Batoulis et al., 2015). BPMN extensions have also been proposed to support more adaptive workflows (Braun et al., 2014), but execution remains largely constrained by static semantics (Mendling et al., 2018). As a result, traditional BPMN execution still struggles with unstructured data, runtime variability, and ambiguous decision logic (Marrella, 2019).

2.2. LLMs and Agentic Workflows

Recent advances in Large Language Models (LLMs) have expanded their role from passive predictors to agents capable of reasoning, planning, and executing tasks from natural language instructions (Wei et al., 2022). Agentic workflows build on this by combining reasoning with tool use to enable autonomous task completion with limited human oversight (Schick et al., 2023). Architectures such as ReAct (Yao et al., 2023b), Tree-of-Thoughts (Yao et al., 2023a), and PAL (Gao et al., 2023) illustrate this shift toward adaptive task coordination and decision-making (Yang et al., 2023). Research has also explored external memory, planning modules, and tool integration (Wu et al., 2024). Systems such as AFLOW (Zhang et al., 2025b) and MaAS (Zhang et al., 2025a) further show the promise of agentic AI for complex tasks such as retrieval, analysis, and decision-making (Singhal et al., 2023). However, these systems also raise challenges around traceability, control, and integration with structured workflow representations such as BPMN (Mialon et al., 2023; Deng et al., 2023). Most remain general-purpose frameworks, with less attention given to specialist workflows for narrowly defined structured automation tasks.

2.3. BPMN and LLM Integration

Initial work on BPMN and LLM integration has focused mainly on modelling rather than execution. Representative directions include generating BPMN diagrams from text, conversational refinement of process models, and workflow mining (Köpke & Safan, 2024; Nour Eldin et al., 2025; Toxtli & Li, 2025; Berti et al., 2024). These approaches improve accessibility for non-experts, but they generally treat BPMN as a static artefact rather than a basis for executable agentic behaviour.

More recent work has moved toward agentic automation, where LLMs synthesise workflows and execute tasks across tools and APIs (Jain et al., 2024; Zeng et al., 2023; Ye et al., 2023). Despite this progress, a key gap remains in combining BPMN's formal process structure with the flexibility of LLM-driven agents. In particular, reliably executing BPMN-defined workflows while preserving semantic rigour and handling unstructured inputs remains an open challenge.

3. Methodology

To evaluate the efficiency and performance gains of a specialist agent, we designed one for converting BPMN-specified workflows into ReAct agents. We evaluated each system using metrics that capture both the agent generation process and the performance of the generated agents, enabling comparison of system efficiency and output quality.

3.1. Workflow Selection

We evaluated our approach using ten deterministic workflows of varying complexity. The workflows were manually constructed by the authors to reflect business process automation tasks across multiple domains, including e-commerce, cost optimisation, risk, and information retrieval. Although this workflow set was not derived from an established benchmark, its construction was informed by foundational business process modelling research. Specifically, we drew on research on business process families and variants (Rosa et al., 2017; Delgado et al., 2022), process model quality and comprehension (Mendling et al., 2010; Figl, 2017), and representative BPMN model generation (Skouradaki et al., 2016) to guide workflow diversity and structural complexity.

The workflows were manually designed because we are not aware of an established benchmark for evaluating the conversion of BPMN-style workflows into executable agentic systems. Existing workflow datasets and benchmark resources generally focus on process discovery, conformance checking, event logs, or model analysis rather than end-to-end evaluation of workflow-to-agent translation and execution fidelity (van der Aalst & Carmona, 2022; IEEE Task Force on Process Mining, 2025; Burattin, 2016). Table 1 in the Appendix summarises each workflow, including the number of nodes (tasks, gateways, and events) and edges (sequence flows), which we use as indicators of structural complexity.

The workflows were selected according to two criteria. First, they span a broad range of structural complexity, from 9 to 52 nodes, allowing evaluation across processes of different sizes and control-flow depth. Second, all workflows are deterministic, in the sense that each execution path is governed by predefined labels and conditions and yields a directly specifiable expected outcome. This design lets us isolate the core capability studied in this paper and evaluate it systematically through exhaustive path coverage. We acknowledge that author-constructed workflows may introduce design bias and that restricting the evaluation to deterministic processes limits generalisability, particularly

to settings involving ambiguity, stochasticity, or open-ended human decision making. The results should therefore be interpreted as evidence for structured, deterministic process settings rather than as a claim of universal representativeness.

3.2. Agentic System Design

Although these workflows could be executed using a fixed Directed Acyclic Graph (DAG), and loops can be achieved via LangGraph, we instead evaluate the ability of our system to construct a ReAct-based solution (Yao et al., 2023b). A ReAct agent conventionally plans actions and calls tools through an interactive loop rather than following a strictly predefined execution path. We chose this formulation for three reasons:

1. **Adaptability beyond fixed execution:** ReAct agents can generalise across a broader range of tasks and respond more flexibly to changing execution conditions. In practical settings such as customer service chatbots, workflows may require extracting information from user messages, calling multiple tools, handling corrected inputs, or revising earlier decisions. These scenarios benefit from an agent that can re-plan during execution rather than follow a rigid script (Leocádio et al., 2024).

2. **Scalability to realistic workflows:** Compared with static DAG execution, agent-based control is better suited to workflows that extend beyond fixed linear paths, particularly when user inputs, state changes, or partial task completion require dynamic coordination across steps.

3. **Ease of authoring:** The logic of a ReAct agent is expressed in natural language through prompt instructions, which can make workflows easier for non-technical subject matter experts to understand and modify. Minor changes to behaviour can therefore be made by editing instructions rather than altering code, which can speed up iteration and development.

3.3. Proposed System

Our proposed system (Figure 1) takes a BPMN-defined workflow and API specifications and automatically generates a working ReAct-style agent through the following steps:

1. **Workflow Parsing:** The BPMN diagram is parsed into discrete steps, identifying tasks, decision nodes, and required API calls. This yields a structured representation of the workflow logic (including branches and conditions) that the language model can reason about.

2. **API Service Generation:** From the provided API specification, the system generates a reusable client module that wraps external calls and abstracts low-level details, reducing tool implementation complexity and code duplication.

3. **Tool/Context Creation:** Using the parsed workflow steps and the API service, the system generates code for each tool corresponding to a workflow node.

4. **Iterative Refinement (Agent Self-Verification):** Generated tool code is executed and validated against expected behaviour. On failure, the system enters a refinement loop: the language model analyses the error and revises the implementation. This loop continues until execution succeeds or successive iterations cease to make substantive progress. Each component is thereby either validated or identified as unresolved.

5. **Agent Assembly and Deployment:** Once all tools are verified, the system composes a natural language prompt encoding the workflow logic and generates a main function that instantiates the ReAct agent behind a FastAPI service. The resulting system—prompts, tools, and API endpoints—is then ready for end-to-end testing.

4. Evaluations and Results

4.1. Baseline Systems and Experimental Setup

We evaluated our system against two automated coding agents, **Roo** and **Cline**, available as Visual Studio Code extensions (Microsoft). All systems received identical BPMN workflows, API specifications, and backend LLM. Roo and Cline were allowed to operate unconstrained with their default approaches, as preliminary experiments showed that imposing additional design constraints degraded output quality. The base prompt is shown in Figure 5, where the `<BPMN>` tags contain the raw BPMN 2.0 XML for each workflow.

After each system declared completion, we conducted end-to-end verification on sample inputs covering all workflow branches. If an agent failed, we fed the error back and allowed iterative self-repair until the issue was resolved or the system could no longer make progress. Our system's agents passed all end-to-end tests on the first attempt without manual intervention (Section 4.3.1), unlike those generated by Roo and Cline.

We also evaluated AutoGen, MetaGPT, and FLOW (Wu et al., 2024; Hong et al., 2024; Niu et al., 2025), but none produced functional solutions for any workflow. MetaGPT's outputs were typically incomplete (e.g., generating only prompts and tools but not the full agent). AutoGen, even

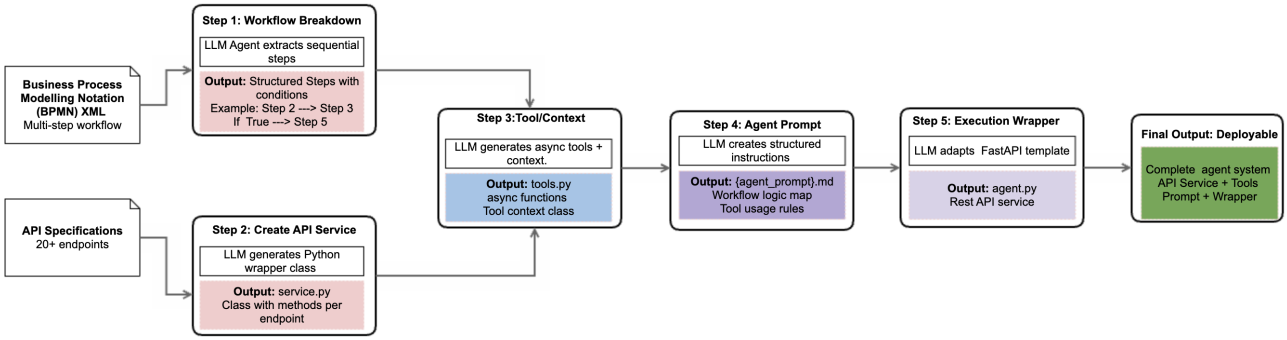

Figure 1. Proposed System Architecture

with its GraphFlow controller, failed to autonomously generate the required files without manual agent wiring. FLOW produced high-level task decompositions but not complete, executable code. As none completed even a single workflow end-to-end, we excluded them from the comparative evaluation, focusing on Roo and Cline as the only baselines that successfully completed the tasks.

4.2. Evaluation Metrics

We evaluate each system across two tasks:

1. the agent generation process, and
2. the functional performance of the generated agents.

For each system under comparison (Roo, Cline and our system), we generated at least 10 agents that successfully compiled and executed according to the BPMN-defined specifications for each of the ten workflows. Each of the 10 agents was then evaluated on a dataset comprising all possible combinations of control-flow flags, yielding comprehensive test coverage across all execution paths. As summarised in Table 1, the workflows vary in structural complexity from 9 nodes and 10 edges (Weather) to 52 nodes and 60 edges (Cart), spanning diverse business domains. This diversity ensures that our evaluation captures a wide range of real-world scenarios, providing a robust testbed for assessing the capabilities of each system in generating functional agents from BPMN specifications.

Achieving the target of 10 successful agents per workflow per system required a total of 371 generation attempts across all three systems, of which 300 succeeded and 71 failed outright. Our system and Cline each needed 117 attempts to obtain their 100 successful agents (85.5% success rate), while Roo required 137 attempts (73.0% success rate). The 300 successful agents were collectively evaluated on 30,543 individual test cases (approximately 102 per run), covering all possible control-flow paths in each workflow. A generation attempt was classified as a *failure* when the coding agent was unable to produce a functional agent at all, for example due to API errors, empty output, or code that could not be executed end-to-end. This is distinct from *repair iterations* (discussed in Section 4.2.1), where the agent initially produced failing code but self-corrected after receiving error feedback. Failure rates varied across workflows: simpler workflows such as Social, Spam, and News were completed without any failures by all three systems, whereas Weather (46.4% failure rate; 56 attempts for 30 successes) and Tournament (42.3%; 52 attempts) proved most challenging. To assess cross-run consistency, we computed the coefficient of variation (CV) of the Tool-Use Exactness score across the 10 independent runs per workflow: our system exhibited the lowest variability (CV = 0.63), followed by Cline (0.86) and Roo (1.05), indicating that our approach produces more consistent agents across independent generations.

4.2.1. Agent Generation Evaluation

The efficiency of the agent generation process was assessed using two primary metrics: the number of *repair iterations* and *token usage*.

1. **Repair Iterations:** We measured the number of refinement loops required to produce a valid agent. For our system, this corresponds to the number of tool-refinement cycles needed to achieve successful execution. For Roo and Cline, this includes code errors and errors in the FastAPI service generated to invoke the agent. Notably, our system's templated approach to developing a FastAPI service that wraps the agent resulted in no errors in this category. We report this as the average number of repair iterations required to complete a single successful generation. Repairs performed in generations where the task was not successfully completed were not included.
2. **Token Usage and Cost:** We tracked the total number of tokens consumed from the initial invocation

through the successful generation of a complete agent, recording input, output, and total token counts. Our system's token usage was measured using Langfuse tracing, while Roo and Cline reported their own token consumption.

4.2.2. Agent Run Evaluations

Each generated agent was evaluated on the following metrics using GPT-4.1 for consistency.

1. **Process Adherence:** Assessed via an LLM-as-a-judge evaluation. The judge received (i) the workflow specification defining the expected tool-call sequence, (ii) the input request, and (iii) the agent's execution trace. Each tool call was categorised as *correct*, *missed* (required but not called), *excess* (called but not required), or *out-of-sequence*. The judge returned a binary adherence verdict (`deviation: true/false`) with a step-by-step reasoning trace.

2. **Tool-Use Exactness (TUE) Score:** TUE is a binary per-run indicator equal to 1 when the agent invokes exactly the prescribed tools with zero missed, excess, or out-of-sequence calls, and 0 otherwise. The aggregate TUE score is the percentage of runs achieving a perfect score.

3. **Penalty-Adjusted Latency:** We define latency as the time required per unit of *net-correct workflow progress*, penalising runs that spend time on incorrect tool usage. Let T denote total execution time, C the number of correct tool calls, and M, E, O the counts of missed, excess, and out-of-sequence calls. The effective steps are
$$P = \max(\varepsilon,\ C - (M + E + O)), \tag{1}$$
where $\varepsilon = 1$ avoids division by zero. The penalty-adjusted latency is then:
$$\text{Latency} = \frac{T}{P} \tag{2}$$
Lower values indicate faster, more accurate execution.

4.3. Results

4.3.1. Agent Generation Results

1. **Repair Iterations:** As shown in Figure 2, the specialist system required zero repair iterations across all successful generations, compared to averages of 2.08 for Roo and 1.89 for Cline, reflecting their reliance on iterative refinement loops to achieve functional outputs.

2. **Token Usage and Cost:** The specialist system averaged 49.24k input and 5.82k output tokens per agent, versus 1,484.46k/17.40k for Roo and 1,029.27k/15.36k for Cline (a 2,915%/199% and 1,990%/164% increase respectively), as shown in Figure 2. Combined with zero repair iterations, the specialist system produces a correct agent in a single pass at ≈55k total tokens, compared to over 1,500k for Roo and 1,044k for Cline before accounting for additional tokens consumed in repair cycles. This order-of-magnitude difference makes specialist pipelines substantially more cost-effective at scale.

4.3.2. Agent Run Evaluation Results

- **Tool-Use Exactness (TUE):** Our specialist system achieved the highest TUE score at 57.69%, compared to 48.62% for Cline (+9.1 pp) and 38.11% for Roo (+19.6 pp) (Figure 4(a)). The per-workflow breakdown in Figure 3(a) confirms this advantage holds consistently across all ten workflows regardless of complexity.

- **Tool-Call Errors:** The specialist system averaged 1.27 total errors per run versus 3.19 for Cline and 3.22 for Roo (both 2.5× higher; Figure 4(b)). The dominant error mode was missed tool calls (0.86 specialist, 2.05 Cline, 1.76 Roo), reflecting generalist agents' tendency to return an output once sufficient information was gathered rather than completing the full prescribed sequence. Excess calls were less frequent but still elevated (0.95/Cline, 1.13/Roo vs. 0.37/specialist), and out-of-sequence calls were low across all approaches (≤0.33). These patterns are consistent across workflows (Figure 3(b)).

- **Process Adherence:** The specialist system achieved the highest adherence rate at 54.68%, versus 51.57% for Cline (+3.1 pp) and 42.43% for Roo (+12.2 pp) (Figure 4(d)). The gap widens on more complex workflows (Figure 3(d)), suggesting that generalist systems' tendency to omit tool calls compounds into lower process conformance as workflow complexity increases.

- **Penalty-Adjusted Latency:** The specialist system achieved the lowest latency at 2.49 s per effective step, versus 6.59 for Cline (2.6×) and 9.08 for Roo (3.6×) (Figures 4(c), 3(c)). The elevated generalist latency reflects both longer raw execution times and a higher incidence of tool-call errors, both of which inflate the metric.

Taken together, the specialist system outperforms both baselines on all four metrics, demonstrating the benefits of structured, specification-driven agent generation over general-purpose coding frameworks. These gains likely stem from

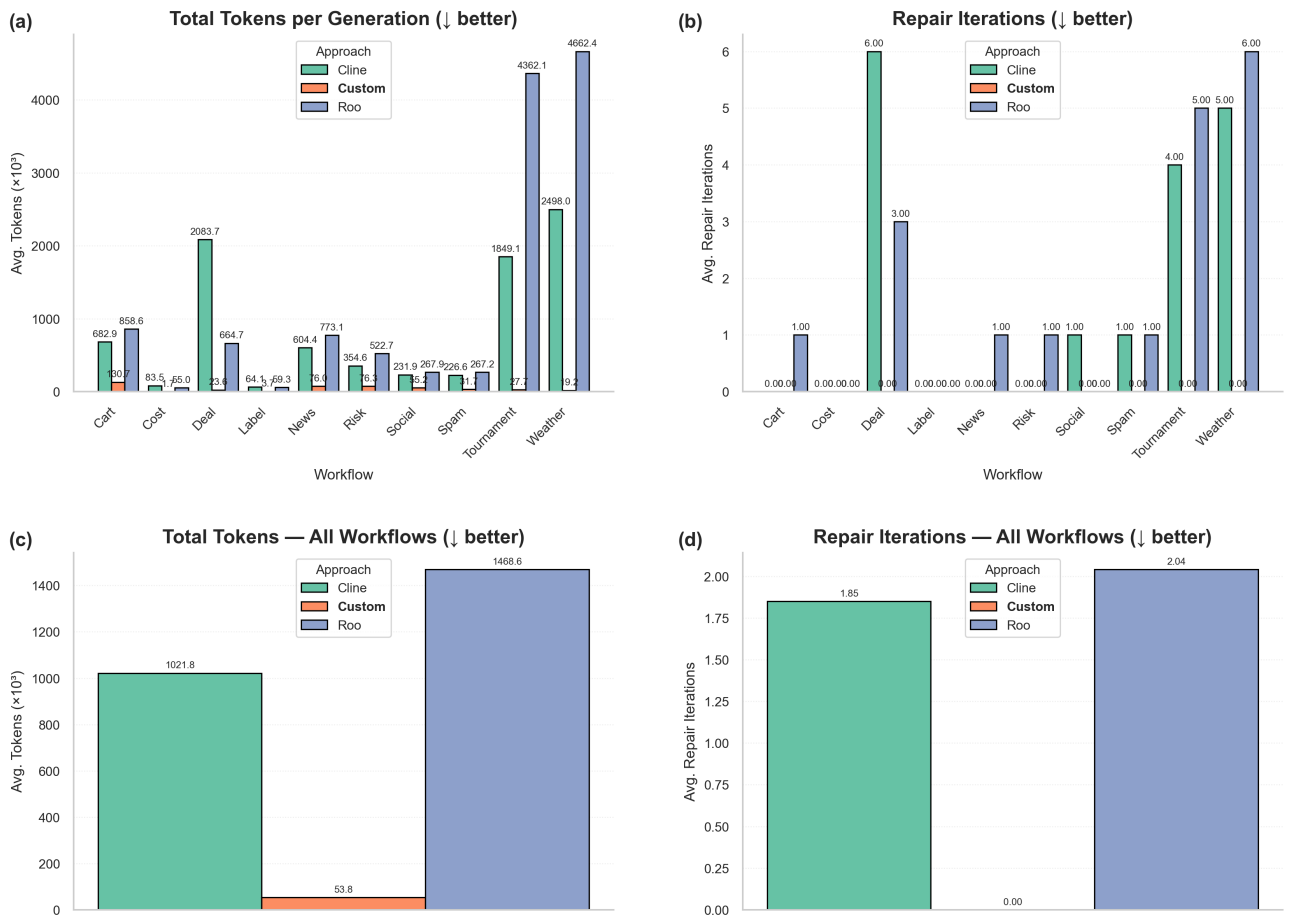

Figure 2. Agent Generation Metrics: Repair Iterations, Token Usage (Input and Output) for the Specialist System(Custom), Roo, and Cline.

the design principles outlined earlier in the paper: constrained execution via externally specified workflow structure, targeted context management, and modular decomposition. Together, these reduce search space and planning overhead, which in turn contributes to fewer tool-call errors and more consistent execution across runs.

5. Conclusion

We presented a specialist agentic system for converting BPMN-defined workflows into executable ReAct agents and evaluated it against Cline and Roo across ten workflows under the same foundation model. The specialist system outperformed both baselines on all four evaluation metrics, achieving higher tool-use exactness and process adherence, substantially lower penalty-adjusted latency, and fewer tool-call errors. It also reduced token consumption by more than 95% relative to the generalist baselines and required no repair iterations. These gains stem from decomposing each workflow into modular prompt, tool, and orchestration components, enabling constrained execution and targeted context management rather than iterative trial-and-error generation.

These improvements are particularly valuable in enterprise settings, where reliability, maintainability, and cost efficiency are critical for large-scale deployment. By externalising workflow structure and reducing unnecessary context exposure, specialist systems can offer a more predictable and operationally manageable alternative to general-purpose agentic coding assistants for structured automation tasks.

The evaluation is scoped to deterministic workflows under realistic low-configuration usage. Although prompt engineering or iterative tuning could improve generalist performance, doing so would require additional user effort and expertise. Future work will examine more heavily optimised baseline configurations and conduct component-level ablations to determine how each structural element contributes to performance, and whether these components can be selectively incorporated into generalist assistants.

6. GenAI Usage Disclosure

In line with our experimental design, LLMs and LLM-powered systems were used to generate and execute the agentic workflows evaluated in this paper. We also used LLMs to enhance our text through light editing tasks such as grammar correction, and sentence restructuring.

References

Aljohani, A. and Do, H. Promptdebt: A comprehensive study of technical debt across llm projects. In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, EASE '25, pp. 371–382, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713859. doi: 10.1145/3756681.3756976. URL https://doi.org/10.1145/3756681.3756976.

An, C., Zhang, J., Zhong, M., Li, L., Gong, S., Luo, Y., Xu, J., and Kong, L. Why does the effective context length of LLMs fall short? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=eoln5WgrPx.

Batoulis, K., Meyer, A., Bazhenova, E., Decker, G., and Weske, M. Extracting decision logic from process models. In *International conference on advanced information systems engineering*, pp. 349–366, 2015.

Berti, A., Kourani, H., and van der Aalst, W. M. P. PM-LLM-benchmark: Evaluating large language models on process mining tasks. *arXiv preprint arXiv:2407.13244*, 2024. doi: 10.48550/arXiv.2407.13244. URL https://arxiv.org/abs/2407.13244.

Braun, R., Schlieter, H., Burwitz, M., and Esswein, W. Bpmn4cp: Design and implementation of a bpmn extension for clinical pathways. In *2014 IEEE international conference on bioinformatics and biomedicine (BIBM)*, pp. 9–16, 2014.

Burattin, A. Plg2: Multiperspective process randomization with online and offline simulations. In *Proceedings of the BPM 2016 Demo Track*, volume 1789 of *CEUR Workshop Proceedings*, pp. 1–6. CEUR-WS.org, 2016. URL https://ceur-ws.org/Vol-1789/bpm-demo-2016-paper1.pdf.

Chinosi, M. and Trombetta, A. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1): 124–134, 2012.

Cline. Cline, 2025. URL https://docs.cline.bot/getting-started/what-is-cline.

Delgado, A., Calegari, D., García, F., and Weber, B. Model-driven management of bpmn-based business process families. *Software and Systems Modeling*, 21:2517–2553, 2022. doi: 10.1007/s10270-022-00985-3.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web. In *NeurIPS 2023 Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.06070.

Dumas, M., Rosa, L. M., Mendling, J., and Reijers, A. H. *Fundamentals of business process management*. Springer, 2018.

Ferrag, M. A., Tihanyi, N., and Debbah, M. From llm reasoning to autonomous ai agents: A comprehensive review, 2025. URL https://arxiv.org/abs/2504.19678.

Figl, K. Comprehension of procedural visual business process models: A literature review. *Business & Information Systems Engineering*, 59(1):41–67, 2017. doi: 10.1007/s12599-016-0460-2.

Fourney, A., Bansal, G., Mozannar, H., Tan, C., Salinas, E., Erkang, Zhu, Niedtner, F., Proebsting, G., Bassman, G., Gerrits, J., Alber, J., Chang, P., Loynd, R., West, R., Dibia, V., Awadallah, A., Kamar, E., Hosn, R., and Amershi, S. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. Pal: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

He, J., Treude, C., and Lo, D. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Trans. Softw. Eng. Methodol.*, 34(5), May 2025. ISSN 1049-331X. doi: 10.1145/3712003. URL https://doi.org/10.1145/3712003.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

IEEE Task Force on Process Mining. Process mining event logs. https://www.tf-pm.org/resources/logs, 2025. Accessed 2026-04-21.

Jain, A., Paliwal, S., Sharma, M., Vig, L., and Shroff, G. Smartflow: Robotic process automation using LLMs. *arXiv preprint arXiv:2405.12842*, 2024. doi: 10.48550/arXiv.2405.12842. URL https://arxiv.org/abs/2405.12842.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024.

Köpke, J. and Safan, A. Introducing the BPMN-chatbot for efficient LLM-based process modeling. In *Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with the 22nd International Conference on Business Process Management (BPM 2024)*, volume 3758 of *CEUR Workshop Proceedings*, Kraków, Poland, 2024. CEUR-WS.org. URL https://ceur-ws.org/Vol-3758/paper-15.pdf.

Laban, P., Hayashi, H., Zhou, Y., and Neville, J. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*, 2025.

Leocádio, D., Guedes, L., Oliveira, J., Reis, J., and Melão, N. Customer service with ai-powered human-robot collaboration (hrc): A literature review. *Procedia Computer Science*, 232:1222–1232, 2024. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2024.01.120. URL https://www.sciencedirect.com/science/article/pii/S1877050924001200. 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023).

Marrella, A. Automated planning for business process management. *Journal on data semantics*, 8(2):79–98, 2019.

Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z.-Z., Zhang, D., Zhou, C., Mao, J., Xia, T., Guo, J., and Liu, S. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334.

Mendling, J., Reijers, H. A., and van der Aalst, W. M. P. Seven process modeling guidelines (7pmg). *Information and Software Technology*, 52(2):127–136, 2010. doi: 10.1016/j.infsof.2009.08.004.

Mendling, J., Weber, I., Aalst, W. V. D., Brocke, J. V., Cabanillas, C., Daniel, F., Debois, S., Ciccio, C. D., Dumas, M., Dustdar, S., et al. Blockchains for business process management-challenges and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 9 (1):1–16, 2018.

Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

Microsoft. Visual studio code. https://code.visualstudio.com/. Accessed: 2025-09-17.

Modarressi, A., Deilamsalehy, H., Dernoncourt, F., Bui, T., Rossi, R. A., Yoon, S., and Schütze, H. Nolima: Long-context evaluation beyond literal matching, 2025. URL https://arxiv.org/abs/2502.05167.

Niu, B., Song, Y., Lian, K., Shen, Y., Yao, Y., Zhang, K., and Liu, T. Flow: Modularized agentic workflow automation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sLKDbuyq99.

Nour Eldin, A., Assy, N., Anesini, O., Dalmas, B., and Gaaloul, W. Nala2bpmn: Automating BPMN model generation with large language models. In Comuzzi, M., Grigori, D., Sellami, M., and Zhou, X. (eds.), *Cooperative Information Systems*, volume 15506 of *Lecture Notes in Computer Science*, pp. 398–404. Springer, Cham, 2025. doi: 10.1007/978-3-031-81375-7_27. URL https://link.springer.com/chapter/10.1007/978-3-031-81375-7_27.

Reichert, M. and Weber, B. *Enabling flexibility in process-aware information systems: challenges, methods, technologies*, volume 54. Springer, 2012.

Rosa, M. L., van der Aalst, W. M. P., Dumas, M., and Milani, F. P. Business process variability modeling: A survey. *ACM Computing Surveys*, 50(1), 2017. doi: 10.1145/3041957.

Rosemann, M., Recker, J., and Flender, C. Contextualisation of business processes. *International Journal of Business Process Integration and Management*, 3(1):47–60, 2008.

Sapkota, R., Roumeliotis, K. I., and Karkee, M. Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic ai, 2025. URL https://arxiv.org/abs/2505.19443.

Sapkota, R., Roumeliotis, K. I., and Karkee, M. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *Information Fusion*, 126:103599, February 2026. ISSN 1566-2535. doi: 10.1016/j.inffus.2025.103599. URL http://dx.doi.org/10.1016/j.inffus.2025.103599.

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2302.04761.

Singhal, K., Azizi, S., Tu, T., Mahdavi, S. S., Wei, J., Chung, H. W., Scales, N., Tanwani, A., Cole-Lewis, H., Pfohl, S., et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

Skouradaki, M., Andrikopoulos, V., and Leymann, F. Representative bpmn 2.0 process model generation from recurring structures. In *Proceedings of the 23rd IEEE International Conference on Web Services (ICWS)*, pp. 468–475. IEEE, 2016.

Toxtli, C. and Li, W. Automating automation: Using LLMs to generate BPMN workflows for RPA integration. In *Computational Science and Computational Intelligence*. Springer, Cham, 2025. doi: 10.1007/978-3-031-86623-4_18. URL https://link.springer.com/chapter/10.1007/978-3-031-86623-4_18.

van der Aalst, W. M. P. and Carmona, J. (eds.). *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*. Springer, Cham, 2022. doi: 10.1007/978-3-031-08848-3.

Van Der Aalst, W. M. P., Bichler, M., and Heinzl, A. Challenges in business process intelligence. *Business & Information Systems Engineering*, 62(6):535–536, 2020.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Weske, M. *Business Process Management: Concepts, Languages, Architectures*. Springer, Berlin, Heidelberg, 3rd edition, 2019.

White, S. A. Introduction to bpmn. *Ibm Cooperation*, 2(0): 0, 2004.

Wooldridge, M. and Jennings, N. R. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Yang, S., Nachum, O., Du, Y., Wei, J., Abbeel, P., and Schuurmans, D. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*, 2023a. URL https://arxiv.org/abs/2305.10601.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Ye, Y., Cong, X., Tian, S., Cao, J., Wang, H., Qin, Y., Lu, Y., Yu, H., Wang, H., Lin, Y., Liu, Z., and Sun, M. Proagent: From robotic process automation to agentic process automation. *arXiv preprint arXiv:2311.10751*, 2023. doi: 10.48550/arXiv.2311.10751. URL https://arxiv.org/abs/2311.10751.

Zeng, Z., Watson, W., Cho, N., Rahimi, S., Reynolds, S., Balch, T., and Veloso, M. Flowmind: Automatic workflow generation with LLMs. In *Proceedings of the 4th ACM International Conference on AI in Finance (ICAIF '23)*, New York, NY, USA, 2023. ACM. doi: 10.1145/3604237.3626908. URL https://arxiv.org/pdf/2404.13050.

Zhang, G., Niu, L., Fang, J., Wang, K., BAI, L., and Wang, X. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=imcyVlzpXh.

Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X.-H., Chen, J., Zhuge, M., Cheng, X., Hong, S., Wang, J., Zheng, B., Liu, B., Luo, Y., and Wu, C. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

A. Appendix

A.1. Workflow Agent Runs

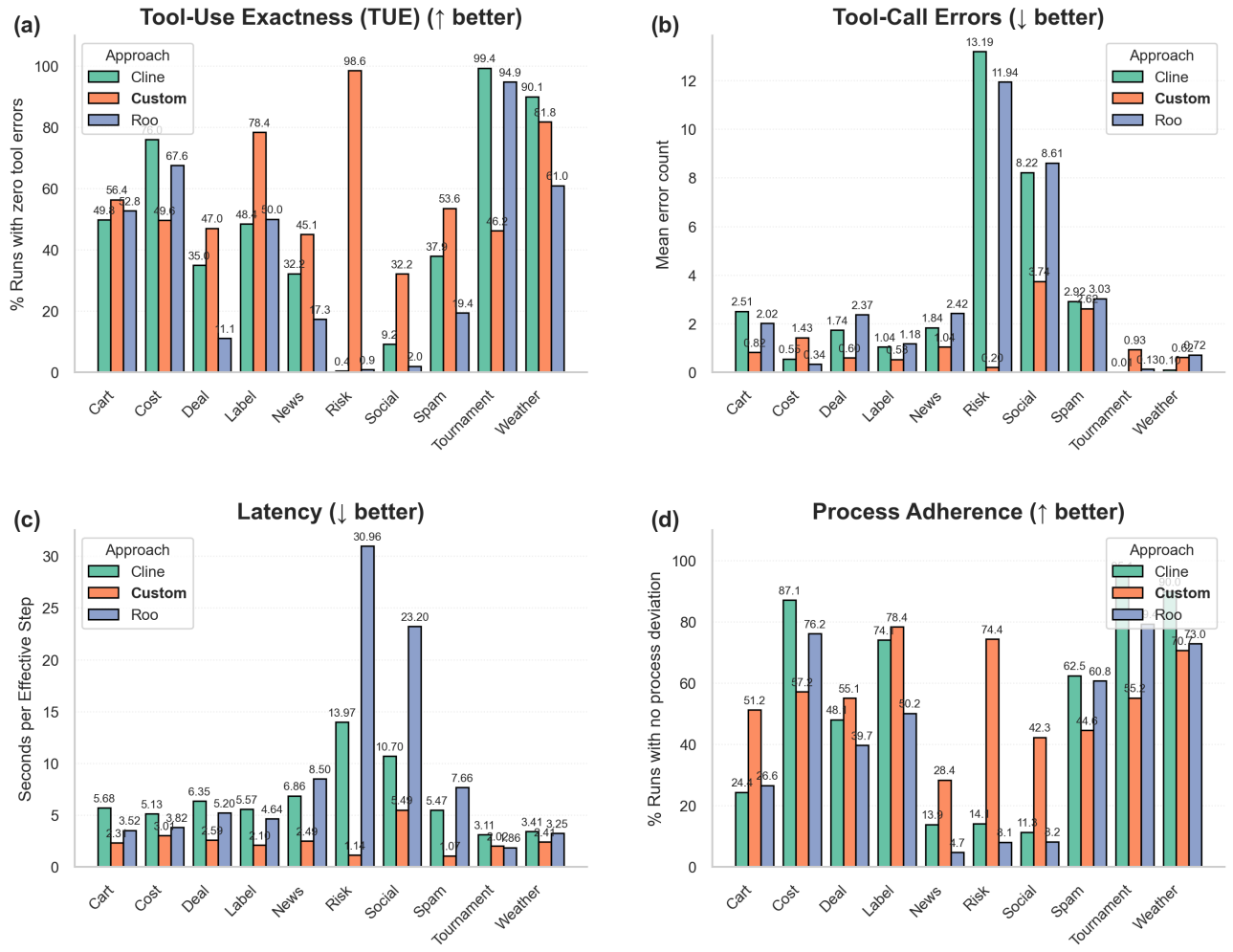

Figure 3. Per Workflow Agent Run Evaluation Metrics: Tool-Use Exactness, Tool-Call Errors, Latency and Process Adherence for the Specialist System

A.2. Workflow Summary Table

Workflow	Brief Overview	Nodes	Edges
Cart	Online retail order fulfillment process covering cart validation, payment authorisation, inventory reservation, shipping, delivery monitoring, and refund/reorder handling.	52	60
Cost	Fraud cost optimisation workflow that evaluates rules, checks fraud flags, payment status, and recoverability to compute operational costs and penalties.	12	15
Deal	Black Friday deal locator that routes based on user preferences (online/in-store), fetches deals, filters by relevance, and presents ranked results.	11	13
Labelling	User disengagement classification workflow that checks eligibility, funding status, and account closure to assign "Disengaged" or "Exclude" labels.	13	16
News	News article collection pipeline that queries multiple providers, validates responses, applies filters, and aggregates results with run metrics.	34	38
Risk	Risk and opportunity classifier that computes risk/opportunity scores, applies threshold checks, assigns priority levels, and persists classification records.	23	25
Social Media	Social media signal collector that fetches posts across platforms, normalises fields, filters by language/region/content type, and builds aggregated datasets.	25	29
Promotion	Customer promotion workflow that fetches profiles, validates eligibility, retrieves order history and loyalty points, segments customers by value tier, calculates discounts, generates recommendations, and sends notifications based on urgency.	18	19
Tournament	Tournament registration process that validates player eligibility, checks slot availability, submits registration, and confirms or rejects enrollment.	14	17
Weather	Weather forecast workflow that fetches weather data, validates retrieval, parses conditions, and displays either severe alerts or regular forecasts.	9	10

Table 1. Summary of the ten deterministic BPMN workflows used for evaluation. Nodes include tasks, gateways, and start/end events; edges represent sequence flows between nodes.

A.3. Key Metrics by Approach

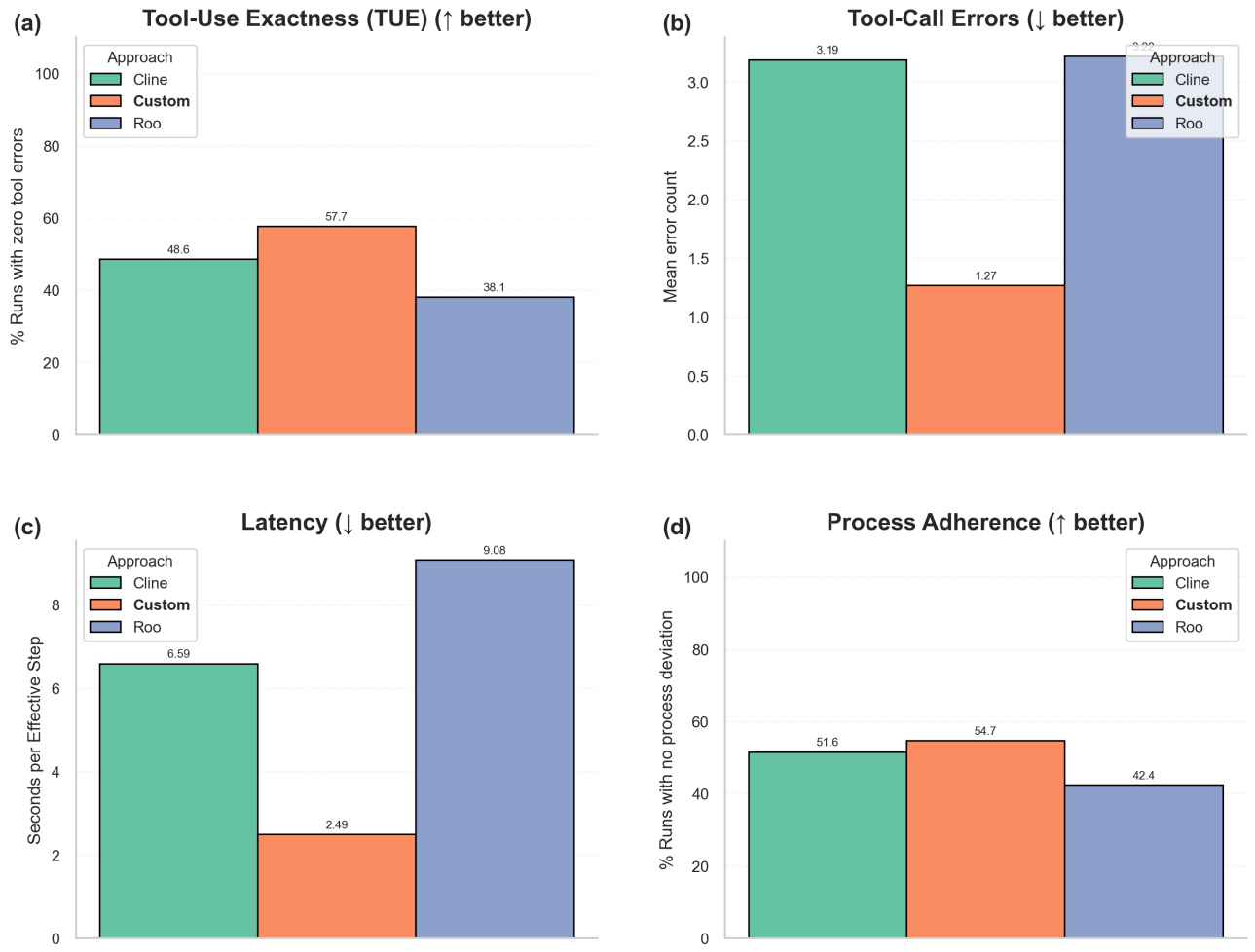

Figure 4. **Key metrics by approach.** Comparison of key metrics across different approaches in agent run evaluation.

A.4. Base Prompt for Agent Generation

```
Create a GenAI powered ReAct agent in a new folder called
{framework}_agent_experiment_{num}.  Do not create a state machine, hardcoded
implementation or any other implementation, your agent should be a GenAI powered
ReAct agent that uses tool calls to complete the task.  Do not ask any questions
about what implementation to use -- you should make all of these decisions yourself.

Your agent folder needs at least 3 files:

 1. agent.py -- a FastAPI service on localhost:7860 with two endpoints:  /chat
    (accepts conversation_id, cif, and message; if no message is required it can
    accept an empty string; returns the end result of the GenAI powered ReAct agent
    designed to execute the workflow) and /get_history (accepts conversation_id;
    returns the complete conversation history and all tools called by the agent
    including tool_name & tool_output, i.e. {"conversation":  [], "tool_calls":  []}).
    It should adhere to the provided API spec.  Ensure this file has a main function
    that uses uvicorn to run the app.

 2. tools.py -- contains all tools for the agent.

 3. prompts/agent_prompt.md -- contains the prompt for the agent.


Use the following to get the necessary API details for the LLM calls for your ReAct
agent:

openai_base_url = os.environ.get("OPENAI_BASE_URL")
openai_api_key  = os.environ.get("OPENAI_API_KEY")
model_name      = os.environ.get("AGENT_LLM")

Your agent should be based on the provided BPMN specification (passed in the <BPMN>
block).  You are not permitted to read any other files to complete this task.  Use
the venv stored at YOUR_VENV_HERE. Do not examine this venv or try to install new
libraries -- if needed they will be automatically installed for you.  Do not attempt
to execute your code after completing it; manual testing will be performed and any
errors will be passed directly to you.
```

Figure 5. Base prompt provided to Roo and Cline for each agent generation attempt. The {num} placeholder was incremented per attempt, and the <API_SPEC> and <BPMN> blocks were populated with the actual API specification and BPMN workflow for each experiment.

A.5. BPMN Specifications for Each Workflow

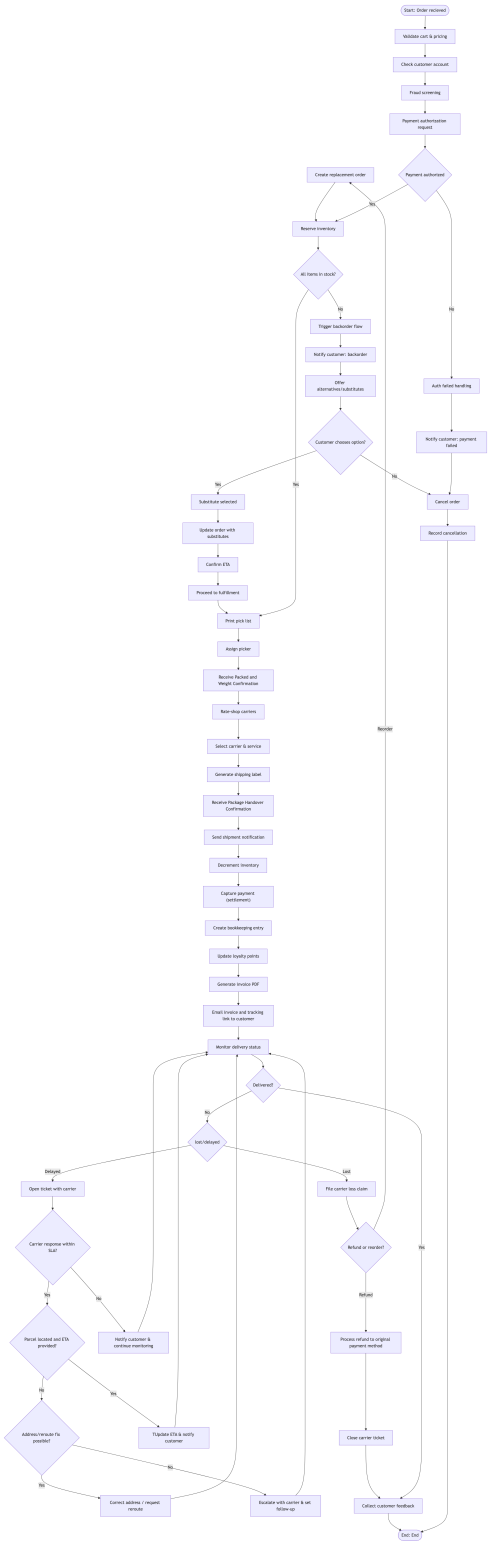

Figure 6. **Online retail cart/order fulfillment workflow.** The process validates the cart, authorises payment, reserves inventory, and routes to either fulfillment/shipping or cancellation/backorder handling depending on payment and stock decisions. After shipping, it monitors delivery and branches into delayed/lost handling, ending in either delivery confirmation, refund, or reorder.

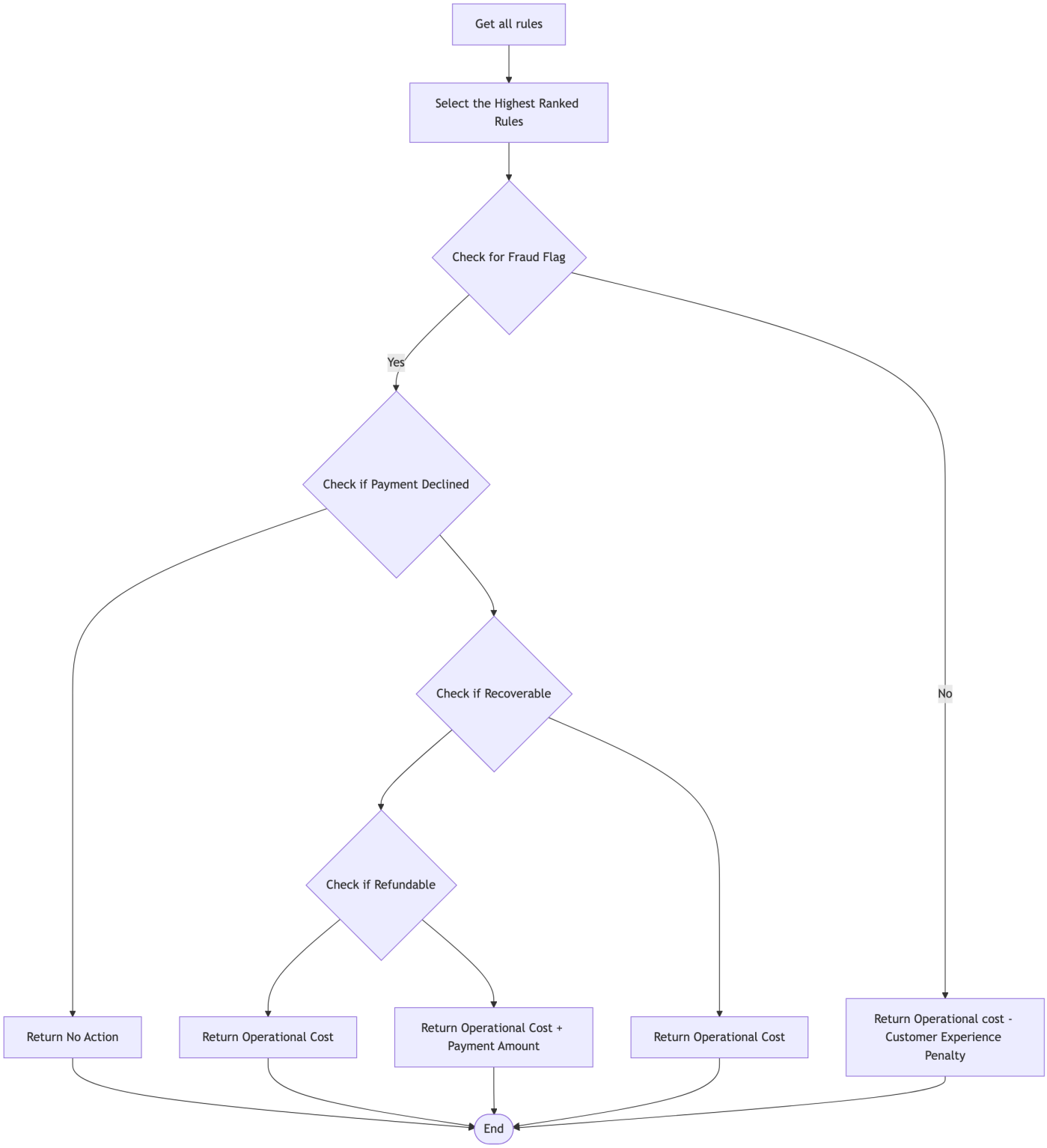

Figure 7. **Costing/estimation workflow.** A deterministic sequence of data-gathering and calculation tasks with decision gateways that select which cost components to apply, followed by aggregation into a final cost/price output.

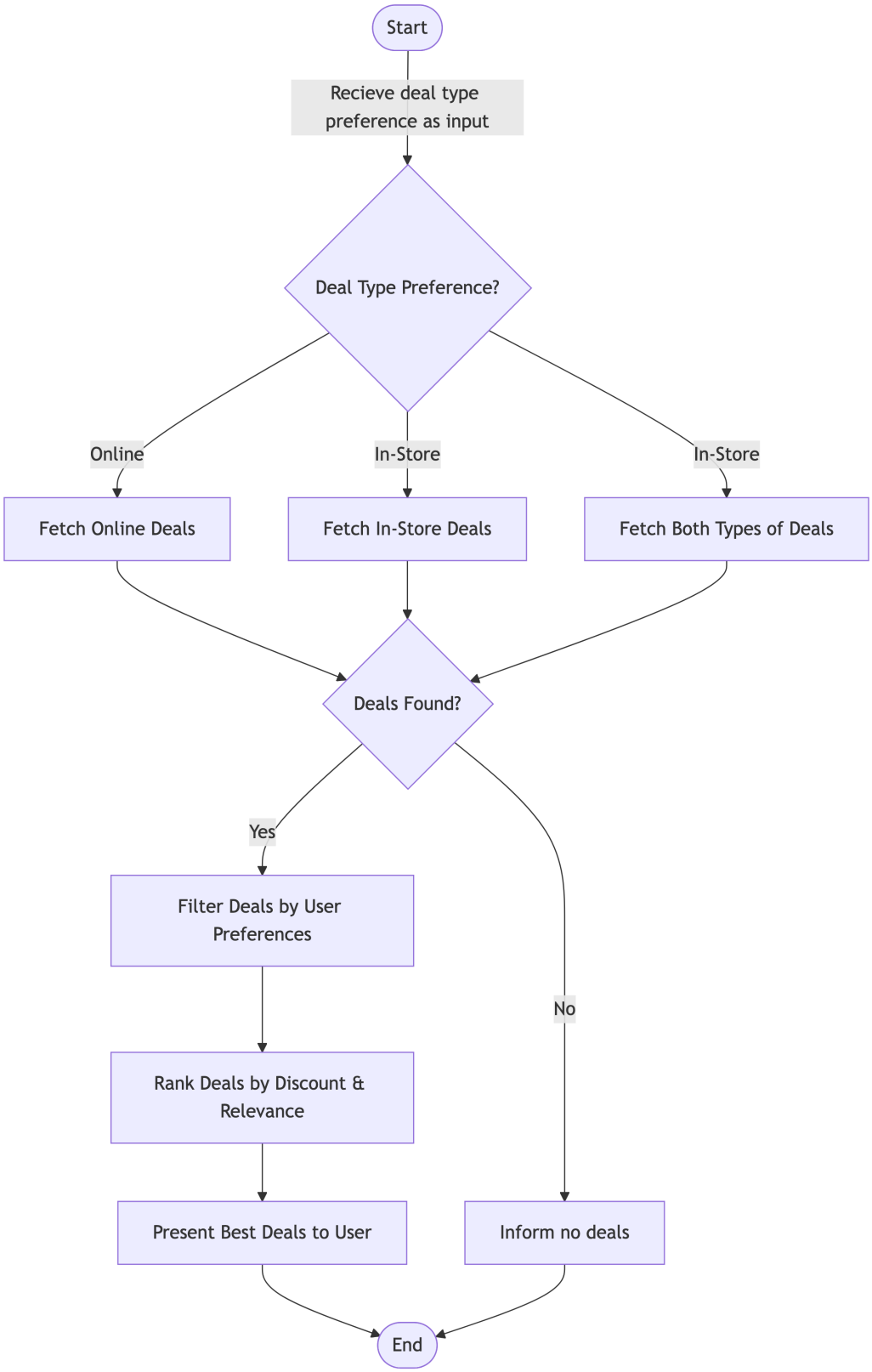

Figure 8. **Deal assessment workflow.** The workflow evaluates a candidate deal through staged checks and scoring steps, using decision gateways to route to accept/reject/escalate outcomes based on thresholds and validation checks.

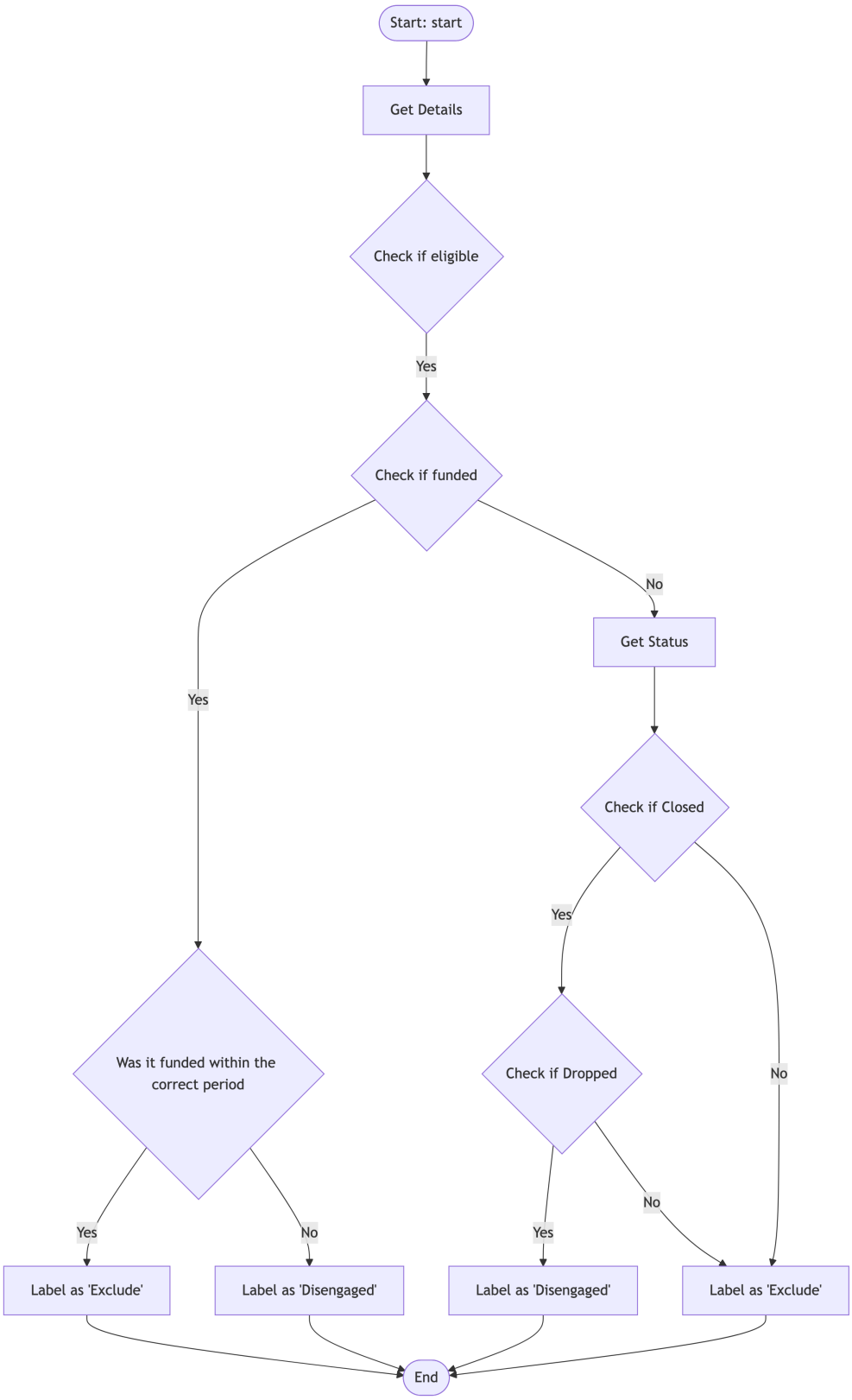

Figure 9. **Labelling/classification workflow.** A rules/criteria-driven labelling pipeline that applies sequential checks and uses gateways to assign labels and/or exclude items, terminating once a label decision is reached.

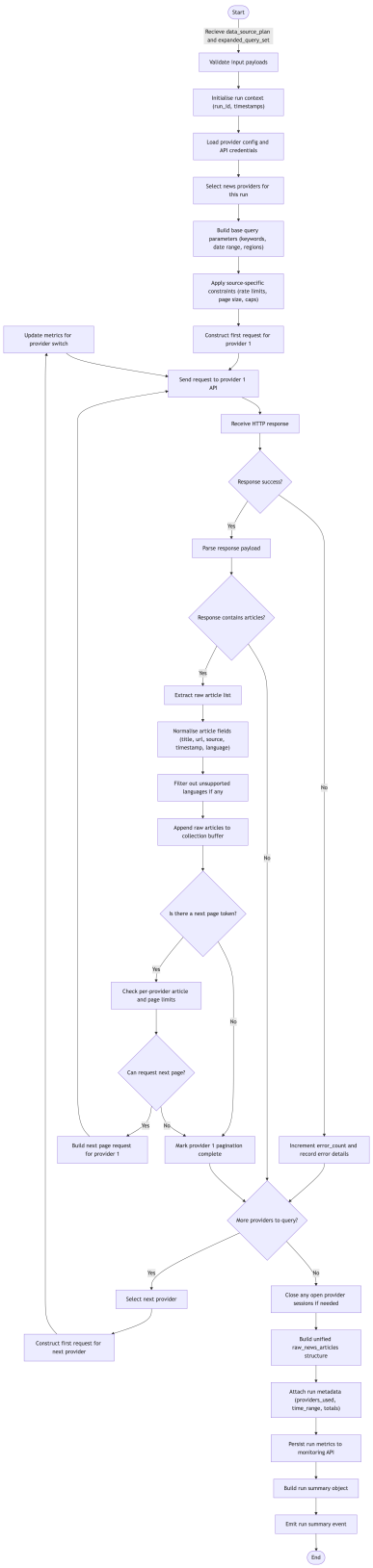

Figure 10. **News processing workflow.** The process ingests an item, performs validation and enrichment steps, then routes through conditional branches (e.g., eligibility/quality thresholds) to produce a final categorisation/output or a discard/escalation outcome.

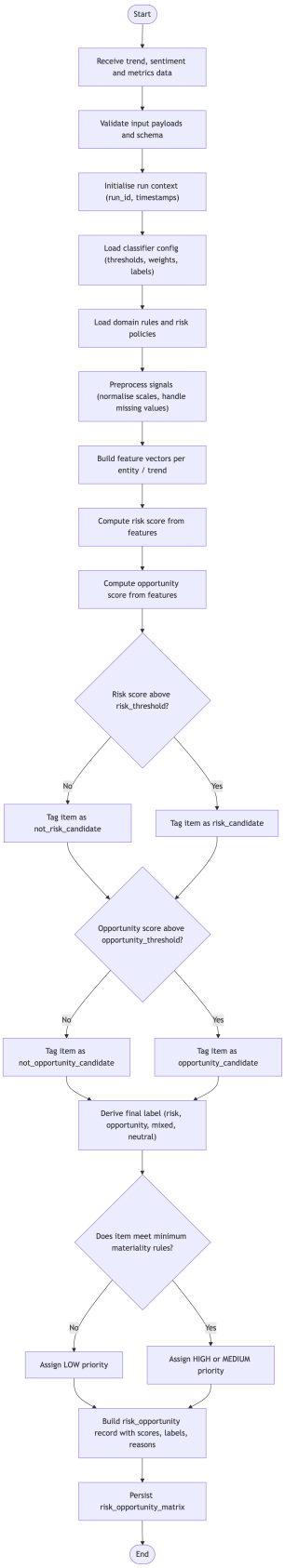

Figure 11. **Risk & opportunity classifier workflow.** The workflow computes risk/opportunity signals, derives a final label (risk/opportunity/mixed/neutral), then routes by a decision gateway into different priority/handling paths (e.g., high/medium vs low) before completing.

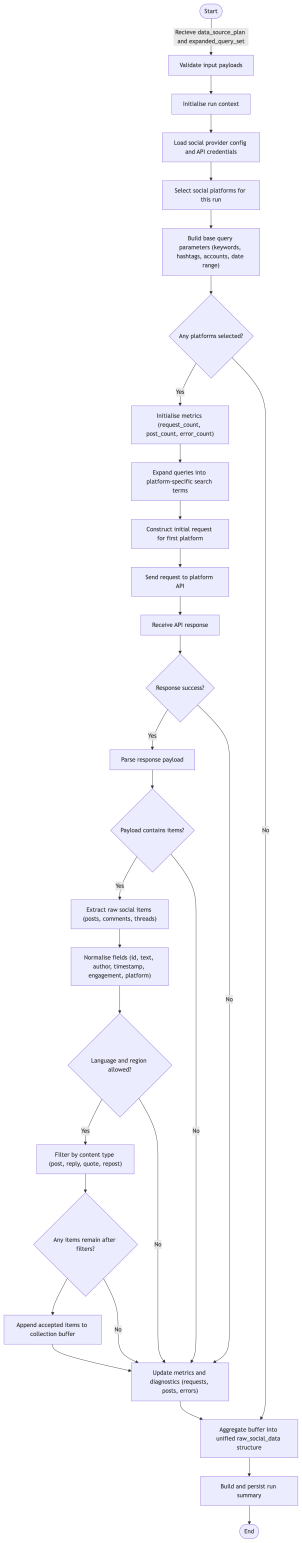

Figure 12. **Social media workflow.** A multi-step pipeline that performs content/user/context checks and then uses decision gateways to choose an action path (e.g., allow, flag, or escalate), ending once an action outcome is produced.

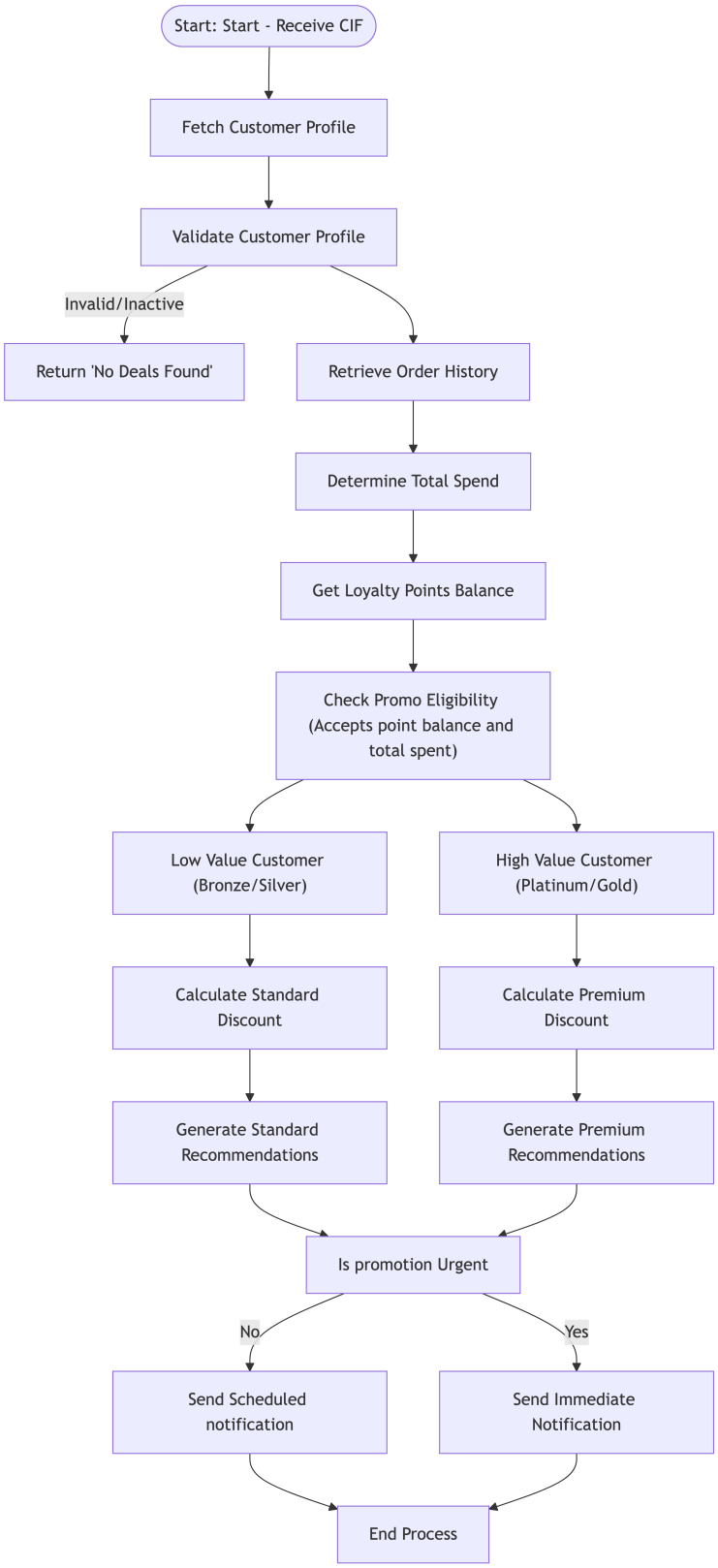

Figure 13. **Customer promotion workflow.** The process fetches and validates customer profiles, retrieves order history and loyalty points, checks promotion eligibility, segments customers into value tiers (high/low), calculates tier-appropriate discounts, generates recommendations, and routes notifications based on promotion urgency.

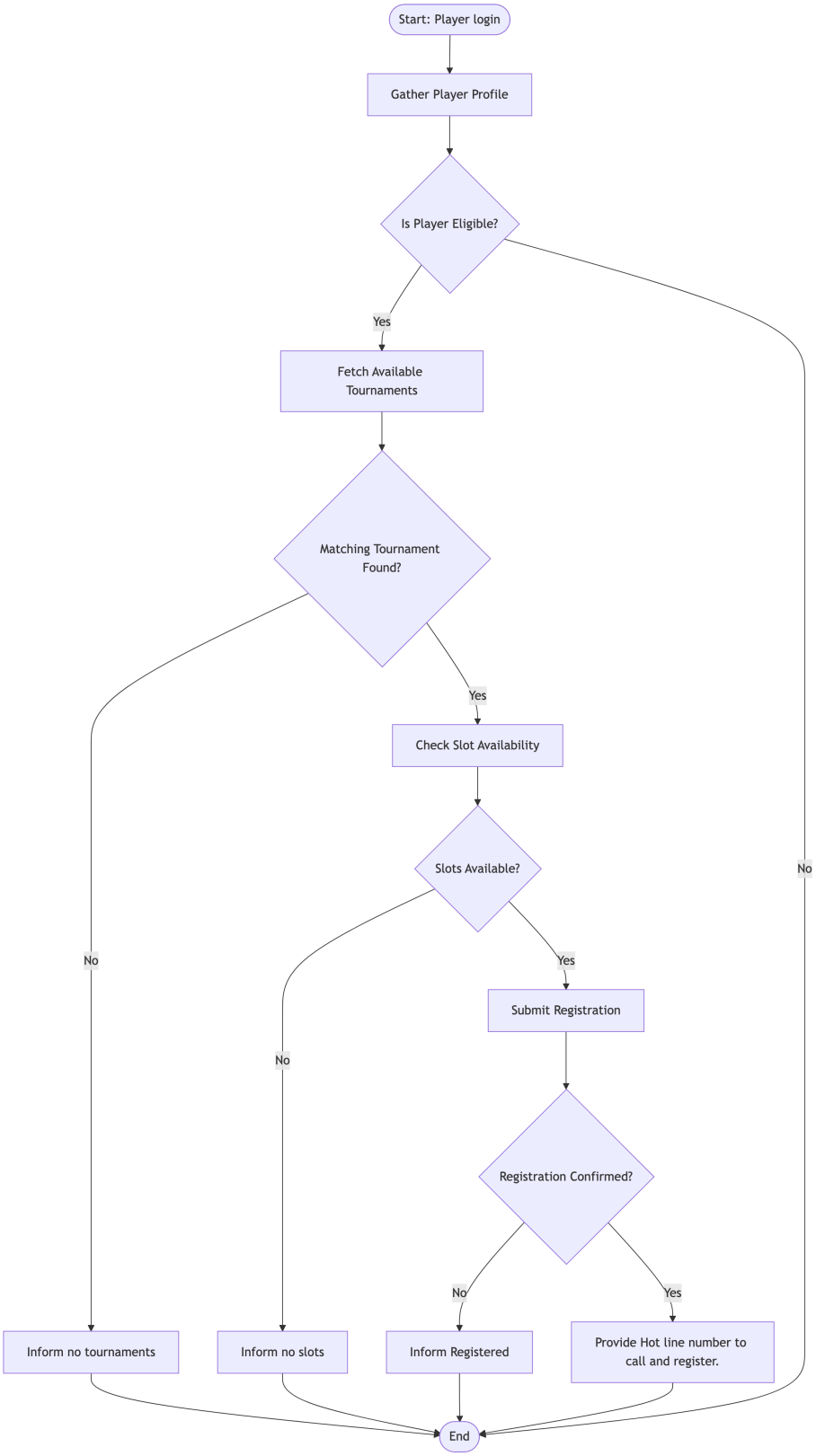

Figure 14. **Tournament/bracket workflow.** The workflow advances items/participants through staged rounds, using decision gateways to determine advancement and termination once a final winner/outcome is determined.

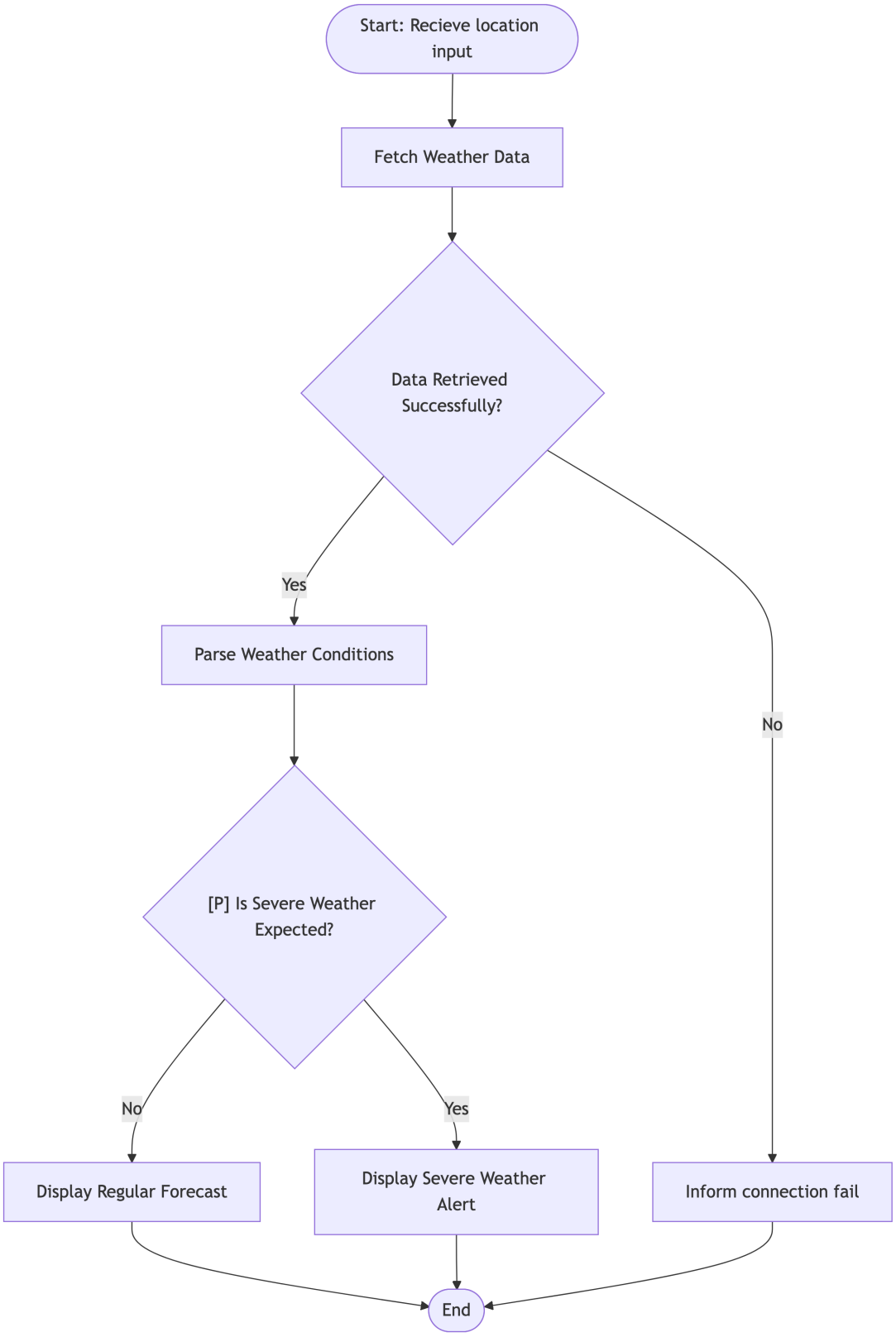

Figure 15. **Weather workflow.** A deterministic sequence of data retrieval and transformation steps with conditional branches (e.g., data availability/threshold checks), culminating in a final forecast/decision output.