PADME: Procedure Aware DynaMic Execution

Extended Abstract

Deepeka Garg
J.P. Morgan AI Research
London, UK
deepeka.garg@jpmorgan.com

Sihan Zeng
J.P. Morgan AI Research
Palo Alto, USA
sihan.zeng@jpmchase.com

Annapoorani L. Narayanan
J.P. Morgan AI Research
London, UK
annapoorani.lakshminarayanan@jpmorgan.com

Sumitra Ganesh
J.P. Morgan AI Research
New York, USA
sumitra.ganesh@jpmorgan.com

Leo Ardon
J.P. Morgan AI Research
London, UK
leo.ardon@jpmorgan.com

ABSTRACT

Executing long-horizon procedures from natural language is challenging for LLM agents due to the lack of structure in free-form instructions like recipes or business workflows, often leading to execution drift or failure. We propose Procedure Aware DynaMic Execution (PADME), a framework that autonomously transforms procedural text into executable graphs capturing dependencies and decision logic. PADME employs a two-phase approach: a *Teach* phase for systematic structuring and an *Execute* phase for dynamic, real-time inputs and environment feedback-driven traversal. This graph-based representation provides an inductive bias that reduces error accumulation and ensures scalability. Empirically, PADME achieves state-of-the-art performance on four benchmarks, including ALFWorld and ScienceWorld, demonstrating that agents equipped with graph-based procedure representations offer a powerful intermediate abstraction for robust and generalizable execution.

KEYWORDS

Autonomous Agents; LLMs; Graph Representation; Workflow Automation.

ACM Reference Format:
Deepeka Garg, Sihan Zeng, Annapoorani L. Narayanan, Sumitra Ganesh, and Leo Ardon. 2026. PADME: Procedure Aware DynaMic Execution: Extended Abstract. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/HIOH2748

1 INTRODUCTION

Real-world procedures involve long horizons and conditional dependencies that cause large language model (LLM) agents to drift, accumulate errors, or fail to maintain coherent execution. While LLM agents can reliably follow short, single-goal prompts, they often lose coherence when reasoning over diverse domains with dependencies that span dozens of steps [1]. Much procedural knowledge resides in free-form text (e.g., SOPs, manuals) designed for humans. Manually rewriting these into formal representations is impractical at scale, while unstructured execution remains fragile due to inconsistent syntax and human-oriented design.

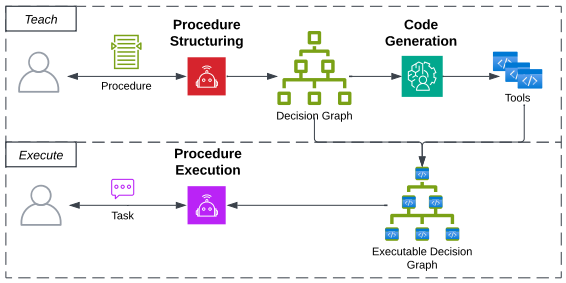

Figure 1: PADME: *Teach* and *Execute* Framework

To address this, we introduce Procedure Aware DynaMic Execution (PADME), a two-phase framework that bridges the gap between free-form text and reliable execution. In the *Teach* phase, a structuring agent autonomously converts procedural text into an executable decision graph. This representation explicitly captures task dependencies and branching choices as decision nodes, providing a verifiable blueprint that is both human-readable for domain experts and machine-readable for agents. Unlike prior works such as AgentKit [2] or SPRING [3], which require manual graph construction, PADME automates the conversion process directly from raw text. Furthermore, while many existing methods are tied to fixed simulators or specific action spaces [4, 5], PADME treats procedural text as the primary knowledge source. By decoupling the semantic structuring of the graph from specific tool bindings, we ensure the logical representation remains stable. In the *Execute* phase, the agent traverses this graph in real-time, adapting to environmental feedback. This modularity allows PADME to generalize across diverse environments and toolsets without requiring the underlying procedure to be restructured.

Our key contribution is a modular two-phase approach that separates *Teach* (automatic conversion of free-form procedures into an executable *decision graph* with dependencies, decision points, and node-level inputs/outputs, pre/post conditions metadata) from *Execute* (context-aware decision graph execution with tool invocation). Figure 1 illustrates the full pipeline. PADME provides domain-agnostic generalization and robust execution across varied tasks and action vocabularies.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/HIOH2748

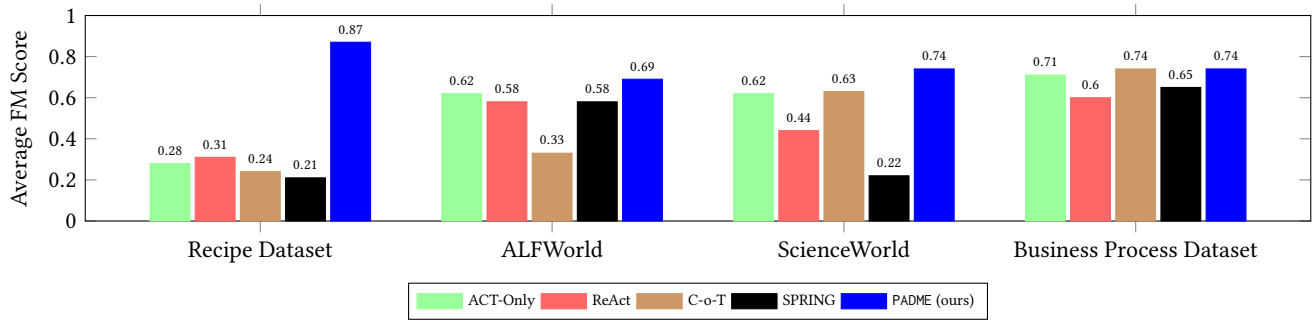

Figure 2: Final Match (FM) scores averaged over five independent trials on each of the four datasets (higher is better).

2 PADME METHODOLOGY

PADME consists of two phases: *Teach*, which transforms text into a structured decision graph, and *Execute*, which traverses the graph adaptively.

2.1 Decision Graph Representation

We formalise a procedure as a directed acyclic graph (DAG) $G = (V, E)$, where each node $v \in V$ is a typed operator $f_v : \mathcal{X}_v \to \mathcal{Y}_v$. Edges (u, v) represent data or temporal dependencies. Nodes are categorised into five classes: (1) Human Input (requires user-provided data); (2) Information Processing (performs internal computation or transformation); (3) Information Extraction (pulls in information from external sources such as APIs); (4) Knowledge (provides reference or background material); and (5) Decision (encodes branching based on conditional logic). Unlike unstructured LLM execution, which searches a space of $\Theta(|\mathcal{A}|^T)$ for T steps, the graph constrains traversal to $O(V + E)$, drastically reducing the search space and bounding error propagation to local parameterisation errors.

2.2 *Teach* Phase: Procedure Structuring

The structuring agent uses an LLM to segment procedural text into actionable units. Each segment is converted into a local subgraph and merged into the global DAG G. To make the graph executable, each node (excluding Decision nodes) is equipped with an executable function or tool binding. While the semantic structure remains stable across environments, these bindings, ranging from API calls to generated code snippets, can be updated without altering the underlying logic. Decision nodes are not assigned static functions; instead, the execution agent resolves them dynamically at runtime based on environmental context. This separation allows for human-in-the-loop validation of the procedural blueprint before a single line of code is executed.

2.3 *Execute* Phase: Dynamic Traversal

The execution agent performs a topological traversal of G. Execution is deterministic for standard nodes but pauses at Decision nodes, where the agent uses the current context (upstream outputs and environmental feedback) to resolve the distribution $p(\text{branch} \mid \text{context})$. This separation ensures that the agent maintains a long-horizon anchor via the graph while retaining the flexibility to adapt to real-time inputs. The graph is then expanded along the chosen edge, and traversal continues. This process repeats until no further nodes remain to be executed. This design frames procedure execution as a form of structured reasoning; deterministic, where the graph specifies explicit dependencies, but adaptive, where decision nodes defer choice to runtime.

3 EXPERIMENTS AND EVALUATION

We evaluate PADME using GPT-4 across four domains: Business Process Dataset [6], Recipe Book (extended from [7] by aggregating individual recipes into a unified "recipe book" akin to a chef's SOP manual), ScienceWorld [8], and ALFWorld [9]. However, PADME is model-agnostic and GPT-4 can be replaced with other models. Each benchmark uses domain-specific tool libraries (e.g., Python APIs). While baselines access the tools upfront, PADME induces the decision graph solely from procedural text.

Baselines: We compare PADME against Act-Only [10], ReAct [10], Chain-of-Thought (CoT) [11], and Studying the Paper and Reasoning to Play Games (SPRING) [3].

Metrics: We evaluate PADME and baselines by comparing their predicted action sequences against ground-truth action trajectories. We report four metrics [1]: *Prefix Match Length* (PML), *Prefix Accuracy* (PA), *Sequential Match* (SM), and *Final Match* (FM). PML/PA measure alignment before drift, while SM/FM capture end-to-end success.

Results: Figure 2 summarizes FM scores over five independent runs. PADME demonstrates consistently strong performance across all datasets. We observe distinct shortcomings across different baselines. ReAct falls into reasoning–action loops (e.g., sensor → capture → sensor → capture), while C-o-T is prone to hallucinations when sketching long plans without grounding. By pre-computing the procedure's topology, PADME ensures the *Execute* phase remains streamlined and robust to environment feedback.

4 CONCLUSION

We introduced PADME, a framework that bridges the gap between free-form procedural text and robust agent execution. By automating the conversion of raw text into executable decision graphs, PADME eliminates the need for manual engineering while providing a stable, verifiable blueprint for automation. Our results across four diverse, long-horizon benchmarks demonstrate that decoupling semantic structuring from dynamic execution significantly reduces reasoning drift and error accumulation.

DISCLAIMER

REFERENCES

[1] Ippei Fujisawa, Sensho Nobe, Hiroki Seto, Rina Onda, Yoshiaki Uchida, Hiroki Ikoma, Pei-Chun Chien, and Ryota Kanai. Procbench: Benchmark for multi-step reasoning and following procedure. *arXiv preprint arXiv:2410.03117*, 2024.

[2] Yue Wu, Yewen Fan, So Yeon Min, Shrimai Prabhumoye, Stephen Marcus McAleer, Ruslan Salakhutdinov, Yonatan Bisk, Yuanzhi Li, and Tom Mitchell. Agentkit: Structured llm reasoning with dynamic graphs. In *First Conference on Language Modeling*, 2024.

[3] Yue Wu, So Yeon Min, Shrimai Prabhumoye, Yonatan Bisk, Russ R Salakhutdinov, Amos Azaria, Tom M Mitchell, and Yuanzhi Li. Spring: Studying papers and reasoning to play games. *Advances in Neural Information Processing Systems*, 36, 2024.

[4] So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*, 2021.

[5] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[6] Flavia Monti, Francesco Leotta, Juergen Mangler, Massimo Mecella, and Stefanie Rinderle-Ma. Nl2processops: towards llm-guided code generation for process execution. In *International Conference on Business Process Management*, pages 127–143. Springer, 2024.

[7] Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. Recipenlg: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, 2020.

[8] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader?, 2022. *URL https://arxiv. org/abs/2203.07540*, 2022.

[9] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

[10] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. *URL https://arxiv. org/abs/2210.03629*, 2023.

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