FEDOT.MAS: Generating Multi-Agent Systems for Complex Tasks with Multi-Stage Validation

Demonstration Track

Dmitry Gilemkhanov
ITMO University
St. Petersburg, Russia
dmitryglhf@itmo.ru

Vadim A. Potemkin
ITMO University
St. Petersburg, Russia
vadim_potemkin@itmo.ru

Maria Zaitseva
ITMO University
St. Petersburg, Russia
mariaz@itmo.ru

Ilia Revin
ITMO University
St. Petersburg, Russia
ierevin@itmo.ru

Anna Kalyuzhnaya
ITMO University
St. Petersburg, Russia
anna.kalyuzhnaya@itmo.ru

Nikolay Nikitin
ITMO University
St. Petersburg, Russia
nnikitin@itmo.ru

ABSTRACT

Complex multi-step tasks such as scientific research are increasingly being performed using LLM-based agents. Existing approaches use fixed agent architectures requiring manual adaptation, or generate individual functions without coordination logic. We present a framework that generates complete multi-agent systems as executable multi-agent system code from natural language descriptions. Our approach employs a seven-stage pipeline with two feedback loops to catch errors at different cost points. On the GAIA benchmark, the framework achieves 49% accuracy on Level 1, 24.4% on Level 2, and 15.4% on Level 3, outperforming the MAS-GPT baseline across all levels. Code and Demo:https://github.com/ITMO-NSS-team/FEDOT.MAS-Demo, https://youtu.be/1w8bBWGHjeQ.

KEYWORDS

Code Generation; Multi-Agent Systems; Task Automation; LLM; Workflow Generation; Validation

ACM Reference Format:
Dmitry Gilemkhanov, Vadim A. Potemkin, Maria Zaitseva, Ilia Revin, Anna Kalyuzhnaya, and Nikolay Nikitin. 2026. FEDOT.MAS: Generating Multi-Agent Systems for Complex Tasks with Multi-Stage Validation: Demonstration Track. 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/OOIG5670

1 INTRODUCTION

The execution of complex, multi-step tasks, particularly in scientific research, increasingly relies on multi-agent systems powered by Large Language Models [3, 8]. Tasks such as hypothesis generation, experimental design, and literature synthesis require not only computation but also sophisticated coordination, dynamic state management, and integration of domain-specific tools. We define task complexity by the number of sequential reasoning steps, required tool integrations, and inter-step data dependencies. In this work, we use the term multi-agent system to refer to LLM-orchestrated pipelines where specialized agents coordinate through generated code.

Recent work on automated agentic system design has explored several directions. ADAS [4] meta-prompts an LLM to propose and refine agent building blocks, while AFlow [10] searches over workflow graphs via Monte-Carlo tree search. SwarmAgentic [11] evolves populations of candidate systems inspired by particle swarm optimization, and GPTSwarm [12] optimizes agent graphs at both node and edge levels. However, these approaches either optimize configurations of predefined primitives or search over prompt spaces—none generates complete, executable multi-agent code from a natural language specification with multi-stage validation. Earlier code-generation methods [2] produce individual functions without coordination logic. MAS-GPT [9] trains an LLM for MAS generation, but relies on the availability of training samples. Our framework closes this gap by generating full Python workflows with a seven-stage pipeline that catches errors before they reach costly execution stages.

We present a system that generates executable multi-agent system code for classes of tasks with multi-stage validation. The generated workflows serve the classes of tasks — groups of structurally similar problems (e.g., 'retrieve and summarize a web page') that share the same agent topology but differ in input parameters. The key idea is to detect errors at multiple levels of abstraction before they propagate to costly stages. The pipeline consists of seven stages with two feedback loops [6]: meta-planning, code generation, static validation, emission, execution, runtime validation, and quality assessment. We evaluate on GAIA [7] (165 complex tasks), demonstrating that multi-stage validation achieves high accuracy without a significant increase in cost (section 3).

2 SYSTEM ARCHITECTURE

The system processes tasks through seven stages with two feedback loops (Figure 1). Stages: (1) meta-planning, (2) code generation, (3) static validation, (4) emission, (5) execution, (6) runtime validation, (7) quality assessment. Feedback loops: static debugging (max 10 iterations), runtime refinement (max 3). Each loop addresses errors at different cost points before expensive execution.

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/OOIG5670

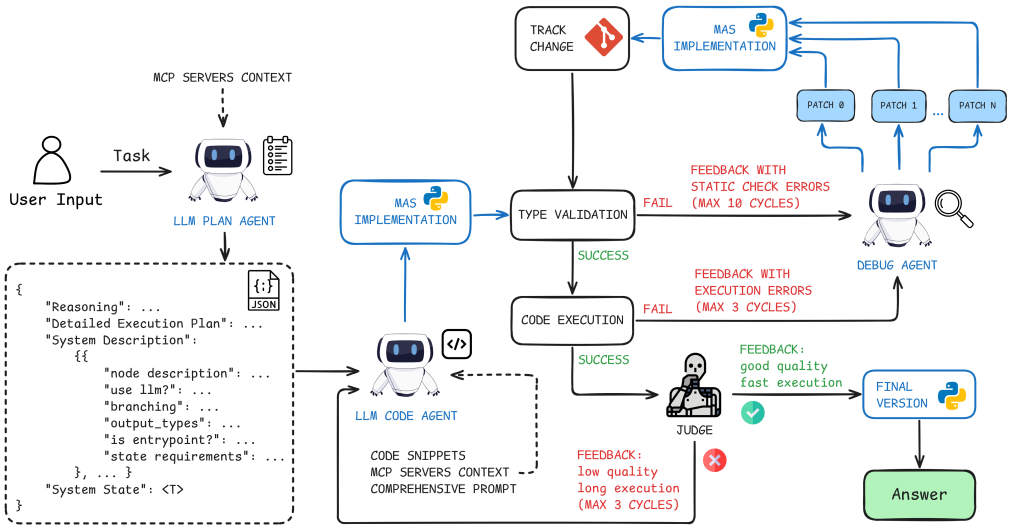

Figure 1: Seven-stage pipeline with two feedback loops catching errors at different cost points.

Stage 1: Meta-Planning. The planner (GraphMetaAgent) receives a task description and generates a JSON plan specifying agent roles, tool assignments, state graph structure, and data flow. The planner uses Claude Haiku 4.5 with a system prompt describing the primitives of the chosen multi-agent framework, available Model Context Protocol (MCP) tools, and state management patterns.

The plan includes input/output keys for state management and agent descriptions for code generation. Plans map directly to multi-agent framework primitives: agents become nodes, dependencies become edges, and custom TypedDict state (AgentState) handles communication through annotated reducers. The planner considers task complexity (tool count, reasoning steps) and selects appropriate agent architectures (linear pipeline, conditional branching, parallel execution).

Stage 2: Code Generation. The GraphCoderAgent converts validated plans to executable Python code. Code generation uses four templates: (1) *State*—`TypedDict` with annotated reducers for inter-agent communication; (2) *Node*—agent functions with structured outputs and MCP [1] tool bindings; (3) *Graph*—composition with nodes, edges, and entry point; (4) *Tools*—MCP server configuration for web search, file analysis, and browser automation.

Stages 3–7: Validation and Execution. **Static Validation** (Stage 3) runs `py_compile`, `ruff`, and type checker locally without LLM calls or API requests, catching syntax and import errors before expensive agent execution. The GraphDebugAgent fixes errors in patch mode (line-level changes, 80–90% token savings vs. full rewrite, max 10 iterations). **Emission** (Stage 4) writes code with version tracking. **Execution** (Stage 5) runs the workflow asynchronously with timeout protection (600s). **Runtime Validation** [5] (Stage 6) checks results and triggers targeted refinement for semantic errors (max 3 cycles)—catching issues static analysis cannot detect: wrong API parameters, data transformation errors, file format assumptions. **Quality Assessment** (Stage 7) uses an LLM judge to evaluate answer completeness and execution efficiency; low-quality results trigger code regeneration.

3 EVALUATION

We evaluate on GAIA [7], a benchmark with 165 questions across three difficulty levels requiring multi-step reasoning and tool use. We compare against MAS-GPT [9], a recent MAS generation approach. Configuration: Gemini-2.5-Flash via OpenRouter, iteration limits 10/3 (static/runtime), 600s timeout. Table 1 confirms that our approach outperforms MAS-GPT across all difficulty levels: +16.9% on Level 1, +4.6% on Level 2, and +3.8% on Level 3. The average cost of our system is $0.32 per task, which is comparable to the operating costs of state-of-the-art multi-agent systems using language models of a similar scale.

Table 1: Accuracy on GAIA Benchmark (165 tasks: 52/86/27 per level)

Method	*Level 1*	*Level 2*	*Level 3*
Ours	**49.0%**	**24.4%**	**15.4%**
MAS-GPT	32.1%	19.8%	11.6%

4 CONCLUSION

We presented a system for generating complete multi-agent systems as executable code with multi-stage validation. Unlike fixed architectures or individual function generation, our approach produces coordinated workflows that serve task classes—enabling batch execution, inspection, and iterative refinement. The seven-stage pipeline catches errors at different cost points: static validation prevents syntax and type errors before execution, runtime validation enables targeted semantic fixes, and quality assessment ensures answer completeness. Future work includes extending the validation criteria based on execution results.

ACKNOWLEDGMENTS

This work supported by the Ministry of Economic Development of the Russian Federation (IGK 000000C313925P4C0002), agreement No139-15-2025-010.

REFERENCES

[1] Anthropic. 2024. Model Context Protocol: Universal Standard for AI-Tool Integration. https://modelcontextprotocol.io/.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[3] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. MetaGPT: Meta Programming for Multi-Agent Collaborative Framework. *arXiv preprint arXiv:2308.00352* (2023).

[4] Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated Design of Agentic Systems. *arXiv preprint arXiv:2408.08435* (2024).

[5] Martin Leucker and Christian Schallhart. 2009. A Brief Account of Runtime Verification. *The Journal of Logic and Algebraic Programming* 78, 5 (2009), 293–303.

[6] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. *Advances in Neural Information Processing Systems* 36 (2023).

[7] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. GAIA: A Benchmark for General AI Assistants. *arXiv preprint arXiv:2311.12983* (2023).

[8] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. https://github.com/microsoft/autogen.

[9] Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. 2025. MAS-GPT: Training LLMs to Build LLM-based Multi-Agent Systems. arXiv:2503.03686 [cs.CL]

[10] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. AFlow: Automating Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=z5uVAKwmjf

[11] Yao Zhang, Chenyang Lin, Shijie Tang, Haokun Chen, Shijie Zhou, Yunpu Ma, and Volker Tresp. 2025. SwarmAgentic: Towards Fully Automated Agentic System Generation via Swarm Intelligence. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 1778–1818. https://doi.org/10.18653/v1/2025.emnlp-main.93

[12] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language Agents as Optimizable Graphs. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 62743–62767. https://proceedings.mlr.press/v235/zhuge24a.html