ChatGPT

Task Decomposition and Multi-Agent Collaboration in LLMs: Recent Advances

Large Language Models (LLMs) have achieved remarkable general capabilities, but solving complex problems with a single monolithic model can be inefficient or fall short in reasoning depth [1] [2]. A promising paradigm emerging in the last 12–18 months is to **decompose tasks into granular subtasks** and employ a **collection of smaller specialized models or agents** that collaborate to solve them. This approach aims to combine the strengths of multiple experts so that, collectively, they can match or even outperform a single large model in both cost-efficiency and performance [3] [4]. Below, we survey recent strategies for task decomposition, multi-agent collaboration methods, and frameworks built around these ideas, highlighting notable research and open-source developments.

Why Decompose Tasks for Multiple LLMs?

Efficiency and Cost: A single state-of-the-art LLM is powerful but computationally expensive. Recent evidence shows that *smaller LLMs fine-tuned for specific tasks can rival larger general models on those tasks* [3]. For example, Amazon scientists observed that a collection of task-focused LLMs (for question-answering, summarization, etc.) can achieve accuracy comparable to a much larger "frontier" model, while offering **70–90% cost savings** due to their reduced size [5] [6]. NVIDIA's research likewise argues that using **Small Language Models (SLMs) (~<10B parameters)** for well-defined subtasks yields *10–30× lower inference cost* with no loss in quality, and often *better reliability*, compared to invoking a 175B model for every little query [7] [4]. The key insight is that most agent or assistant workloads (formatting JSON, retrieving an order status, answering a focused query, etc.) only require a narrow slice of an LLM's ability. In such cases, *"agentic"* architectures that route subtasks to right-sized models are far more efficient than treating a giant model as a Swiss Army knife [4] [8]. By **specializing smaller models for specific functions**, we also reduce the likelihood of extraneous text or hallucinations in their output, since each model is tightly aligned to a format or domain [8] [9]. This reliability translates to fewer failure points in a complex workflow, as one NVIDIA blog put it: *an agent doesn't need a Swiss Army knife when a single sharp tool will do* [8].

Performance and Accuracy: Beyond cost, breaking a complex task into subtasks can **improve accuracy and reasoning**. This is because each subtask can be given a focused context and handled by a model or module that excels at that function. For instance, Wu *et al.* (Apple, 2024) found that encouraging an LLM to first perform a *problem decomposition phase* (laying out intermediate steps or subgoals) before the *problem solving phase* led to better results than a single-stage prompt [10]. In their "Divide-and-Conquer" study, they actually **distilled** the decomposition ability into a smaller model, showing it generalized well across tasks [11]. The smaller *planner* model could output a step-by-step plan, which a larger solver model then executed – achieving comparable outcomes with lower inference cost overall [12]. This supports the idea that certain aspects of reasoning (like planning a solution path) don't require massive knowledge parameters, just the ability to apply general strategies [13].

Researchers have also proposed that multiple reasoning passes or explicit subtask prompts can surpass one-shot reasoning. Techniques like **Chain-of-Thought (CoT)** prompting and **self-ask** (iteratively asking and answering sub-questions) allow even moderate-sized models to tackle complex queries they would otherwise fail. A notable example is **Tree-of-Thoughts** (Princeton, 2023), where an LLM explores a tree of possible solution steps and uses search algorithms to find a consistent answer – effectively decomposing the reasoning into many small steps and backtracking as needed. Such approaches have been competitive with much larger models on puzzles and planning problems by leveraging logical decomposition instead of brute-force "intelligence." In summary, *allocating subtasks to specialized reasoning paths or models often yields better accuracy on each component*, which in turn improves the end-to-end result when the answers are composed.

Multi-Agent Collaboration Frameworks

A vibrant area of recent research focuses on **LLM-based multi-agent systems**, where each agent is an LLM (often a smaller or mid-sized one) assigned a specific role or expertise. These agents communicate and coordinate to solve a complex task together. This paradigm draws inspiration from human organizations and the concept of *collective intelligence*: a team of specialized minds can outperform any solitary genius on complex, multifaceted problems [14].

Role Assignment and "Persona" Agents: One intuitive way to decompose a task is by splitting it according to distinct roles or skills. For example, if the task is to build a website automatically, one can imagine a "UX Designer" agent to decide the layout and text, an "Artist" agent to generate needed images, a "Developer" agent to produce the code, and perhaps a "QA tester" agent to verify everything [15] [16]. This is exactly what Burak Gozluklu's team demonstrated in an AWS blog: they created a persona-based *agentic workflow* that handled personalized website generation with multiple smaller LLMs instead of one giant model [17] [16]. The *Personalizer* agent (an average-sized text model with some reasoning ability) designs the site content and style using company policies and user info (with retrieval-augmented prompts to keep it factual) [18] [19]. Next, an *Artist* agent (using a text-to-image model) produces visuals as described. Finally, a *Front-End Developer* agent takes the design spec and generates the actual HTML/CSS/JS code [20] [21]. By chaining these specialized models, they not only saved cost, but also gained modular control – e.g. they could easily swap out the Artist for a better image model, or pinpoint which stage failed if the end result had issues [22] [23]. This persona-oriented decomposition mirrors how a human team would collaborate, and indeed many recent frameworks explicitly use this **"AI team" metaphor**.

Several open-source projects in 2023–2024 explored this idea of multi-agent role-play. **MetaGPT** (DeepWisdom, 2023) is one of the first frameworks to simulate an entire software startup with multiple GPT-based agents [24] [25]. In MetaGPT, a single line requirement from the user is handed to a "CEO" agent, which then delegates to a "CTO" or product manager agent to do planning, who in turn coordinates "Engineer" agents to write code, "Tester" agents to verify, etc. – all implemented by prompting LLMs with role-specific instructions [24] [26]. The system provides standard operating procedures (SOPs) and communication protocols so that agents stay in scope and exchange information effectively (for example, the Engineer only writes code and asks for clarification if needed, the Tester only reports bugs) [26] [27]. This structured *assembly-line paradigm* breaks down a complex software project into manageable chunks, handled via dialogue between the specialized agents. **ChatDev** (Zhou et al., 2023) similarly showed that two GPT-4 instances emulating a software manager and a developer can collaborate through conversation to produce non-trivial programs (including documentation) in a matter of minutes, something a single model

prompted in isolation could struggle with. These projects were early proofs that *"LLM societies" can tackle larger tasks through division of labor.*

Another influential framework is **CAMEL (Communicative Agents for Mind Exploration)** by Li et al. (NeurIPS 2023) [28] [29]. CAMEL introduced a simple but effective *role-playing method*: two chat agents (originally using OpenAI GPT-4 or LLaMA-7B) are assigned roles (e.g. "AI user" and "AI assistant" with a specific skill) and given an inception prompt describing a task goal. The agents then **autonomously dialog** with each other to break the problem down and solve it cooperatively, without human messages in between. This yielded intriguing results – for example, a pair of CAMEL agents with 7B-parameter models was able to achieve high-quality multi-hop question answering and even perform 128k-length text retrieval [30], by virtue of one agent asking for partial info and the other retrieving or summarizing as needed. In essence, the conversation itself served as the "decomposition strategy": the agents would naturally ask clarifying questions, divide the work, and verify each other's outputs. The CAMEL paper open-sourced their framework, and it spurred a community (camel-ai.org) dedicated to scaling up multi-agent cooperation experiments (with projects like **Workforce** for orchestrating many agents on a single task, and **OASIS** for simulating social networks of agents) [31] [32].

Orchestration and Tool Use: Not all multi-agent approaches use symmetric agents – some use a dedicated *orchestrator* or facilitator. Microsoft's **HuggingGPT** (April 2023) is a notable example where a powerful LLM (ChatGPT) was used as a central planner that *decomposed a user request into sub-tasks* and then routed those to various specialist models from HuggingFace Hub [33]. For instance, if the user asks, "Analyze this image and write a story about it," the orchestrator LLM breaks this into (1) image analysis (handled by a vision model), then (2) story generation (handled by a text model), and finally composes the results. HuggingGPT demonstrated that a collection of **many small expert models**, each competent in a narrow domain (vision, speech, math, etc.), can be *steered by an LLM* to accomplish multi-modal or complex goals that one model alone could not. This falls under what Wu *et al.* (2025) describe as *"functional model collaboration,"* where different models contribute different abilities (e.g. an OCR model + an LLM together solve a document understanding task) [34]. Open-source libraries like **AutoGen** (Microsoft, 2023) generalize this idea by providing an API for LLMs to engage in asynchronous conversations and tool usage. AutoGen essentially lets developers script a group of agents (some powered by LLMs, some as tools/APIs) and handles the message passing between them [35]. The *AutoGen paper (Wu et al., 2023)* showcased agents that call each other or external APIs when needed, enabling more flexible task plans than a single static prompt.

Autonomous Task Agents: Alongside formal research, 2023 saw a surge of community-built "autonomous agents" like **AutoGPT** and **BabyAGI**. These are open-source projects (originating on GitHub) that gained viral attention for allowing an LLM to **iteratively plan and execute subtasks** towards a high-level goal. Interestingly, AutoGPT is described as a *"multi-agent framework"* in that it creates multiple sub-agents (all essentially instances of GPT-4) which talk to each other or themselves to break down tasks [36]. In practice AutoGPT often runs as a single looping agent that generates a to-do list of subtasks, executes them one by one (using tools or code it writes), and adjusts the plan based on results – effectively simulating a team of one. Despite their unpredictable performance, these projects demonstrated the *feasibility of task decomposition without human oversight*: given a goal like "Research and write a blog post on cloud computing," AutoGPT would spawn steps like "(1) search for cloud computing basics, (2) summarize key points, (3) draft an outline, (4) write the post," using the LLM at each step. The takeaway is that even a single LLM, when looped with memory, **can play the roles of planner, executor, and reviewer by self-prompting**, which hinted at the power of decomposition. This influenced later academic works to formalize how multiple LLMs can coordinate in a more reliable and structured way.

Modular Architectures and Specialized Subsystems

Moving from high-level frameworks to specific architectural innovations, several recent papers propose *modular systems* where different components (often implemented by LLMs or fine-tuned variants) handle different subtasks in a pipeline or feedback loop. The goal is typically to address known weaknesses of monolithic LLM reasoning (such as planning accuracy or factual consistency) by introducing modules that can check, critique, or refine the outputs of others.

Brain-Inspired Planner (MAP): A standout example is the **Modular Agentic Planner (MAP)** introduced by Mondal, Webb, et al. (2024) [1] – which was published in late 2025 in *Nature* under the title *"A brain-inspired agentic architecture to improve planning with LLMs."* This system explicitly takes inspiration from cognitive neuroscience: the human prefrontal cortex divides planning into distinct executive functions. MAP implements a set of **specialized LLM-based modules** that correspond to these functions: a *Task Decomposer* that breaks a high-level goal into sub-goals, an *Actor* that proposes actions to achieve a given sub-goal, a *Monitor* that filters out invalid or nonsense actions, a *State Predictor* that simulates the world state after an action, an *Evaluator* that scores progress toward the goal, and an *Orchestrator* that decides when the goal is met or when to stop [37] [38]. These modules interact in a loop (with the Orchestrator guiding the overall sequence), essentially forming a closed-loop planning algorithm distributed across multiple LLM "agents." The payoff is significant: on challenges like the Tower of Hanoi puzzle, graph traversal problems, and StrategyQA, MAP *outperformed strong baselines* including standard chain-of-thought prompting, self-consistency, **Tree-of-Thought** search, and even a multi-agent debate approach [39] [40]. For example, MAP was more reliable in avoiding illegal moves (thanks to the Monitor module) and knowing when it had reached a correct solution (thanks to the Orchestrator) [41]. Crucially, the authors demonstrated that **using smaller, more cost-efficient models for the modules still retained the performance gains** [42] [43]. In fact, they report that a MAP built with a *70B Llama 2* model achieved strong results, and even a prototype with a yet-smaller "Llama3-70B" maintained the improvements over a single GPT-4 solving the same tasks [40]. This indicates that strategic decomposition can compensate for a moderate drop in model size by leveraging multiple focused reasoning steps. The MAP study provides a clear blueprint (and even pseudocode in the appendix) for how one can **factor an LLM's reasoning process into distinct subtasks** and achieve emergent planning capabilities that the base model didn't have.

Hierarchical Multi-Agent Reasoning: Another important development is addressing the *inefficiencies* that can arise in multi-agent systems – e.g. redundant work or unnecessary communication between agents. Zhang *et al.* (2025) propose **D^{3}MAS (Decompose, Deduce, Distribute)**, a hierarchical coordination framework for LLM-based agents [44]. They observed that naive multi-agent setups often suffer from agents duplicating the same retrievals or reasoning steps, leading to nearly 50% redundant information in communications [45]. D^{3}MAS tackles this with a three-layer design: a **Task Decomposition layer** that first splits the problem and assigns sub-problems to different agents (filtering out irrelevant work early), a **Collaborative Reasoning layer** where agents share partial inferences to cover different "paths" of reasoning, and a **Distributed Memory layer** that stores and provides non-redundant facts so agents don't repeatedly fetch the same knowledge [44] [46]. These layers are implemented with a structured graph that passes messages in a controlled way, ensuring each agent only gets the minimal information needed from others. The result is a dramatic reduction in wasted effort – knowledge redundancy was cut by ~46% – and a boost in overall problem-solving accuracy by **8.7–15.6%** on challenging reasoning tasks [44] [47]. In experiments on HotpotQA (a multi-hop QA task) and MMLU (a knowledge test), D^{3}MAS outperformed both a single-agent chain-of-thought and previous multi-agent methods (like Microsoft's AutoGen or a DAG-based agent network) [48]. This work highlights that *effective task decomposition isn't only about splitting the work,*

but also about coordinating the pieces optimally. Too many agents or poorly organized subtasks can introduce overhead (e.g. communication latency or integration complexity) that negates the benefit. D³MAS provides a quantitative framework to think about this: using k smaller LLMs in parallel adds an overhead roughly on the order of $O(k^m)$ for integration (with m between 1 and 2 in their analysis), so one wants k small and m minimized for the approach to beat a single model [49] [50]. The sweet spot is a handful of agents with clear division – something also echoed in the Amazon blog cautioning against *over-engineering* decomposition when a simple solution would do [51] [52].

Verification and Self-Correction Modules: Another line of work has focused on augmenting LLMs with *checker* or *critic* agents to improve reliability. For instance, one agent can generate a solution and another can evaluate or find flaws in it, sending feedback for refinement. The concept of *multi-agent debate* (inspired by a 2017 OpenAI idea) was tested in the LLM era – e.g. two smaller models arguing about the answer to a question and a third judging who is correct. While debate alone has had mixed success, it paved the way for adding explicit verification steps. A notable recent result is **MALT: Multi-Agent LLM Training** by Motwani *et al.* (2024) [53]. MALT is a post-training strategy rather than an architecture: they fine-tune a base LLM into three specialized versions – a *Generator* (produces candidate reasoning chains/answers), a *Verifier* (checks the correctness of the generator's solution), and a *Refiner* (proposes improvements or fixes) [54]. During training, these roles were engaged in a loop to generate a diverse set of reasoning paths (a "multi-agent search tree") and each path was graded against ground-truth answers, giving feedback signals to each role model [54]. This yielded a set of role-conditioned models that, at inference time, can collaborate sequentially: Generator proposes a solution, Verifier flags errors, Refiner revises it. Impressively, MALT using three **8B-sized models** cooperatively was able to *significantly outperform* a single 8B model of the same family on complex benchmarks – achieving 7–15% higher accuracy on math word problems (MATH dataset), logical reasoning (GSM8K), and commonsense QA [55]. In other words, three smaller models each focusing on a part of the reasoning process ended up solving problems better than one model doing it all in one go. This underscores a general theme in recent research: **divide and conquer** not just at inference, but even during *training*, can unlock higher performance without increasing model size. By letting an LLM specialize in "being a critic" or "being a resolver," we can train it on signals (like catching an arithmetic mistake) that a one-shot model would never see in its loss function.

Related approaches include using an ensemble of diverse strategies and then picking the best answer. This is akin to having multiple agents propose answers and a coordinator choose one (sometimes called *Self-Consistency* when done with multiple chain-of-thought samples [43]). The "When One LLM Drools, Multi-LLM Collaboration Rules" report (Feb 2025) went so far as to argue that *multi-LLM collaboration is an essential path toward compositional intelligence*, noting that collaboration allows representation of diverse knowledge and perspectives that a single model might not capture. The report categorized methods by how much the models share information (from independent voting to full message exchange) and pointed to *compositional generalization* as a key benefit [56] [57]. In essence, multiple smaller models can each contribute pieces of the puzzle – whether by proposing different hypotheses, cross-checking each other's work, or each handling a different modality of data.

Open-Source Implementations and Community Discussions

Many of the ideas above are not just theoretical – they have been implemented in open-source libraries or discussed in community forums, accelerating their adoption. We've already mentioned **MetaGPT**, **CAMEL**, **AutoGPT**, and **AutoGen**, all of which have repositories or at least detailed technical blogs available. These tools lower the barrier for developers and researchers to experiment with multi-agent orchestration. For

example, MetaGPT's GitHub provides ready-made "agent role" templates (PM, Architect, Coder, Tester, etc.) so you can plug in your own API keys and watch a team of GPT-4 agents build a simple game for you. CAMEL's open library allows defining any number of agents and custom "inception prompts" to set their persona and goals, which has led enthusiasts to try everything from agent-based storytelling to AI negotiating games.

Another interesting project is **AgentVerse** (Chen et al., 2024) – a framework by researchers at Microsoft for *facilitating multi-agent collaboration and exploring emergent behaviors*. It supports configurable agent environments (e.g. cooperative vs competitive settings) and has been used to benchmark how multiple LLM agents perform on tasks like negotiation, trading, and collaborative writing [58]. Having such frameworks is useful to systematically study *when* multiple agents are truly better than one. Early findings suggest that on tasks requiring diverse expertise or multi-step planning under uncertainty, carefully orchestrated agents shine, whereas for straightforward tasks, a single fine-tuned model is still best (due to lower overhead). This nuance is echoed by Amazon's caution that too much decomposition can "over-engineer" a solution – the best approach balances the complexity of the task with an appropriate number of agents [51] [52].

The **community has actively discussed** the pros and cons of these approaches. On forums like OpenAI's and Reddit, one common question is how to handle the *coordination failures*: for example, two agents might get stuck in a loop asking each other the same question (a known issue in early CAMEL experiments), or an orchestrator might misidentify which tool to use for a subtask (as seen in HuggingGPT occasionally). Solutions being tried include having a "watchdog" agent that can interrupt and adjust course if agents loop, or integrating a simple heuristic (like timeouts or vote-based termination) to decide when the group should stop iterating. There is also interest in **standard benchmarks** for multi-agent systems. A recent survey by Tran *et al.* (2025) introduced a taxonomy for collaboration mechanisms and noted the lack of unified evaluation methods [56] [59]. Efforts like **MultiAgentBench** (2025) are starting to provide testbeds with scenarios that require coordination or competition, so that different agent architectures can be objectively compared [60].

Outlook: Towards Collective Intelligence

In summary, the last 1–2 years have seen rapid progress in enabling LLMs to **break down tasks and collaborate**, with numerous approaches converging on the idea that **many minds can be better than one**. Task decomposition strategies – from simple prompt chaining to sophisticated planner-executor splits – have proven effective in pushing the performance of smaller models to or beyond the level of a single large model on various benchmarks [10] [55]. Multi-agent systems and modular architectures introduce a form of *specialization* and *communication* that lets AI systems tackle more complex, long-horizon problems than was possible with a single prompt-response cycle. Notably, these advances often come with the benefit of **transparency** and **debuggability**: it's easier to trace which component failed (and perhaps fix or improve it) when the process is divided into clear subtasks, as opposed to a giant black-box model that produced an end result [22] [23].

That said, there are challenges ahead. Coordination overhead and system complexity are real concerns – as one analysis put it, using multiple LLMs adds an extra term to the computational complexity, and if you naively scale up the number of agents, you hit diminishing returns or even regressions in quality [61] [62]. Researchers are actively looking into **optimization of communication** (so agents only share the necessary bits of information) and into **adaptive task allocation** (so that subtasks are assigned to agents only when there is a clear gain). Techniques like the D^3MAS graph-based messaging and Model Swarms' swarm

intelligence optimization are early attempts to make multi-agent setups more *scalable* and *principled*. On the flip side, model collaboration opens doors to continuous learning: Wu *et al.* (2025) envision *"model co-evolution"* where multiple models not only solve tasks together but also **learn from each other over time**, exchanging knowledge or skills to collectively improve [63] [64]. This is an exciting direction akin to an ecosystem of models evolving cooperative behavior – a step closer to a robust collective AI.

In conclusion, the paradigm of **"decompose, then solve (together)"** has gained strong traction, with research showing tangible benefits in accuracy, efficiency, and reliability. From academic papers like *MAP* (which improved multi-step reasoning by merging six small expert agents) [40] to industry deployments like Amazon's agentic workflow for web design (which cut costs by using task-specific LLMs) [65] [66], it's clear that *a carefully orchestrated team of models can accomplish what a single model cannot, or do so more affordably*. As toolkits for multi-agent orchestration mature and best practices (and pitfalls) become better understood, we can expect to see *collections of specialized LLMs/agents working in concert* at the core of many AI systems – embodying the old adage that **"many hands make light work,"** even when those hands are virtual.

Sources:

- Burak Gozluklu, *"How task decomposition and smaller LLMs can make AI more affordable,"* Amazon Science Blog, Sept. 2024 [3] [17].
- Zhuofeng Wu *et al.*, *"Divide-or-Conquer? Which Part Should You Distill Your LLM?"* EMNLP 2024 (Apple ML Research) [10] [12].
- Mondal, Webb *et al.*, *"Improving Planning with LLMs: A Modular Agentic Architecture (MAP),"* arXiv 2023 / Nature 2025 [40].
- Heng Zhang *et al.*, *"D^{3}MAS: Decompose, Deduce, and Distribute for Enhanced Knowledge Sharing in Multi-Agent Systems,"* arXiv 2025 [44].
- NVIDIA Research (Belcak *et al.*), *"Small Language Models are the Future of Agentic AI"* (Position paper & blog), Aug. 2025 [8] [67].
- IBM Developer, *"MetaGPT and Multi-Agent Collaboration"* (V. Winland & M. Syed), 2023 [26] [36].
- Motwani *et al.*, *"MALT: Improving Reasoning with Multi-Agent LLM Training,"* arXiv Dec. 2024 (v3 Oct. 2025) [54] [55].
- Tran *et al.*, *"Multi-Agent Collaboration Mechanisms: A Survey of LLMs,"* arXiv Jan. 2025 [68] [56].
- Feng *et al.*, *"Model Swarms: Collaborative Search to Adapt LLM Experts via Swarm Intelligence,"* arXiv Oct. 2024 [69] [70].
- Li *et al.*, *"CAMEL: Communicative Agents for 'Mind' Exploration of LLM Society,"* NeurIPS 2023 [28] [29].
- Xixi Wu *et al.*, *"AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation,"* arXiv Aug. 2023 [35].
- *Additional resources and open-source implementations are cited inline throughout the text.*

---

[1] [40] [43] Improving Planning with Large Language Models: A Modular Agentic Architecture
https://arxiv.org/html/2310.00194v5

[2] [33] [34] [63] [64] Knowledge-Empowered, Collaborative, and Co-Evolving AI Models: The Post-LLM Roadmap
https://www.engineering.org.cn/engi/EN/10.1016/j.eng.2024.12.008

3 5 6 15 16 17 18 19 20 21 22 23 49 50 51 52 61 62 65 66 How task decomposition and smaller LLMs can make AI more affordable - Amazon Science
https://www.amazon.science/blog/how-task-decomposition-and-smaller-llms-can-make-ai-more-affordable

4 8 9 67 How Small Language Models Are Key to Scalable Agentic AI | NVIDIA Technical Blog
https://developer.nvidia.com/blog/how-small-language-models-are-key-to-scalable-agentic-ai/

7 Small Language Models Are the Future of Agentic AI: Here's Why | by MKWriteshere | Towards AI
https://pub.towardsai.net/small-language-models-are-the-future-of-agentic-ai-heres-why-6fe5e4fa9834?gi=25931fc5927b

10 11 12 13 Divide-or-Conquer? Which Part Should You Distill Your LLM? - Apple Machine Learning Research
https://machinelearning.apple.com/research/divide-conquer

14 24 25 26 27 36 What is MetaGPT ? | IBM
https://www.ibm.com/think/topics/metagpt

28 29 [2303.17760] CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society
https://arxiv.org/abs/2303.17760

30 CAMEL: Communicative Agents for "Mind" Exploration of Large ...
https://www.facebook.com/groups/DeepNetGroup/posts/2047328628993342/

31 32 CAMEL-AI Finding the Scaling Laws of Agents
https://www.camel-ai.org/

35 44 45 46 47 48 58 "D"^{3}"MAS": Decompose, Deduce, and Distribute for Enhanced Knowledge Sharing in Multi-Agent Systems
https://arxiv.org/html/2510.10585v1

37 38 39 41 42 Nature publishes brain-inspired agent planning for LLMs, outperforming baselines. | Maksim Pukin posted on the topic | LinkedIn
https://www.linkedin.com/posts/mpukin_llm-ai-agents-activity-7379361814020771840-IM76

53 54 55 [2412.01928] MALT: Improving Reasoning with Multi-Agent LLM Training
https://arxiv.org/abs/2412.01928

56 57 59 60 68 Related papers: Multi-Agent Collaboration Mechanisms: A Survey of LLMs
https://fugumt.com/fugumt/paper_check/2501.06322v1_enmode

69 70 Model Swarms: Collaborative Search to Adapt LLM Experts via Swarm Intelligence
https://arxiv.org/html/2410.11163v1