ORGANIZATION OF SELF-CONTROLLED AGENTS FOR GENERAL MATRIX MULTIPLICATION OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents have evolved towards greater autonomy with the advancement of model context protocols. Self-controlled agents, such as Codex and Claude Code, highlight the need for novel organizational frameworks that facilitate agent-level autonomy. In this paper, we propose a tree-based orchestration system, TrAgent, which utilizes a PUCT-style search to dynamically allocate agent actions while maintaining autonomy. This approach offers three key benefits: (i) full agent autonomy for critical tasks like planning and tool use, (ii) a generalized mechanism for inter-agent experience sharing, and (iii) scalability as the number of agents increases. We demonstrate the system's effectiveness through the general matrix multiplication kernel optimization, achieving 80% of the performance of the cuBLAS code. Additionally, the system exhibits a scaling phenomenon as the number of agents increases. Our approach provides a solution for organizing increasingly autonomous agents.

1 INTRODUCTION

Early Large Language Model (LLM) agents are often configured as collections of fixed roles or modules (*e.g.*, a planner, a tool invoker, a coder), orchestrated by hand-crafted workflows or conversation patterns. Recent developments, however, indicate a shift toward agents that can independently plan, use tools, maintain memory, search, and manage versioned artifacts. In particular, agents such as Codex (OpenAI, 2025) and Claude Code (Anthropic, 2025a;b), show that agents can autonomously perform many of the tasks that earlier frameworks delegated to explicit controllers. Model Context Protocols (MCP) further standardize how agents connect to external data and tools (GitHub, 2024; Anthropic, 2024), and an ecosystem of MCP servers (*e.g.*, Context7 for up-to-date code documentation (Upstash, 2024a;b)) and search utilities (*e.g.*, DuckDuckGo Instant Answers (Duckduckgo, 2023)) continues to expand. We refer to this emerging paradigm as *self-controlled agents*: systems in which the agent itself makes most task-level decisions while selectively interfacing with the environment through general connectors and skills.

This evolution raises new challenges for organizing self-controlled agents. Existing multi-agent or workflow-based systems (*e.g.*, conversational orchestration (Wu et al., 2023; Li et al., 2023)) typically rely on explicit role assignments and context passing. We observe three limitations when these patterns are applied to self-controlled agents. First, fine-grained top-down control can inadvertently suppress agent autonomy. Second, coordination through shared prompts is limited by context lengths. Third, many controllers do not scale gracefully with increasing agent capability and system size, yielding diminishing returns as the number of agents grows.

To address these challenges, we propose **TrAgent**, a tree-based orchestration method that employs a PUCT-style search to coordinate self-controlled agents. Inspired by recent work like AlphaEvolve (Novikov, 2025) and , TrAgent represents decision states as nodes and agent proposals/actions as edges; selection–expansion–evaluation–backup cycles allocate exploration budgets while preserving per-agent autonomy. Priors and values are derived from agent judgments and performance signals; backups propagate experience so that subsequent exploration is increasingly informed. This design (i) preserves full agent autonomy on critical tasks (planning, tool use), (ii) provides a generalized, value-based mechanism for inter-agent experience sharing, and (iii) scales with the number and strength of agents via structured, budgeted search.

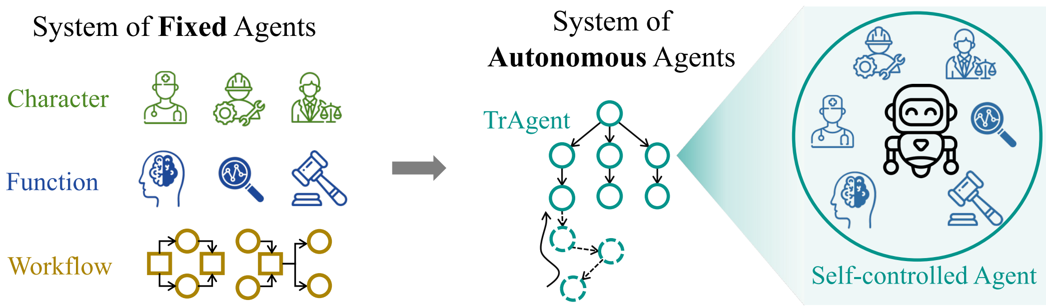

Figure 1: We propose TrAgent, a system designed for self-controlled agents that preserves the autonomy of each agent's actions. Additionally, it leverages shared experiences for inter-agent communication and scales effectively as the number of agents increases.

We focus our empirical study on general matrix multiplication (GEMM) kernel optimization. GEMM is a compelling testbed because it blends algorithmic structure (*e.g.*, tiling, data reuse, scheduling) with code-level implementation choices, and has outsized practical impact in HPC and deep learning (Goto & van de Geijn, 2008; Van Zee & Van De Geijn, 2015; Chen et al., 2018a; Zheng et al., 2020). The task requires both reasoning over design alternatives and producing executable kernels, offering a balanced evaluation of reasoning, coding, and performance under realistic constraints. We demonstrate that TrAgent substantially improves over single self-controlled agents and a random baseline on GEMM optimization, approaching roughly 80% of a strong vendor library across representative settings.

2 Related Work

2.1 LLM for GEMM Optimization

Classical vendor-tuned libraries such as cuBLAS and MKL embody decades of expert engineering; principles like cache-aware blocking and packing remain foundational (Goto & van de Geijn, 2008). Automatic empirical tuning projects (*e.g.*, ATLAS) demonstrated that search can recover performant schedules (Whaley et al., 2001). Modern auto-tuning compilers decouple algorithms from schedules and search over high-dimensional spaces: Halide (Ragan-Kelley et al., 2013), TVM (Chen et al., 2018a), AutoTVM (Chen et al., 2018b), and Ansor/FlexTensor (Zheng et al., 2020; 2022). In parallel, learning-based approaches inform optimization decisions and performance estimation for code (Cummins et al., 2017; Mendis et al., 2019). Foundational libraries and frameworks such as BLIS (Van Zee & Van De Geijn, 2015) and autotuning systems like OpenTuner (Ansel et al., 2014) provide complementary baselines and techniques. Compiler infrastructure advances (*e.g.*, MLIR (Lattner et al., 2021)) further enable domain-specific transformations. Recent LLM-driven agents bring tool use, memory, search, and self-reflection into the loop (*e.g.*, Toolformer (Schick et al., 2023), Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023)), and multi-agent frameworks enable conversational coordination (*e.g.*, AutoGen (Wu et al., 2023), CAMEL (Li et al., 2023)). Our single-agent baselines align with this line of work, while our system of self-controlled agents focuses on organizing multiple self-controlled agents to scale performance.

2.2 Tree-Search-Based Agent Systems

Tree search is a powerful abstraction for reasoning and acting, and recent work has adapted it to language agents (Zhou et al., 2023; Yao et al., 2023; Besta et al., 2024). Beyond breadth- and depth-first exploration, Monte Carlo Tree Search (MCTS) has emerged as an effective controller for LLM agents in planning and reasoning tasks. LLM-MCTS (Zhao et al., 2023) treats the LLM both as a commonsense world model (providing priors over states) and as a policy heuristic to guide rollouts, substantially improving search efficiency and performance on multi-step tasks. In scientific reasoning, Monte Carlo Thought Search (Sprueill et al., 2023) integrates MCTS with thought-level exploration to surpass chain-of-thought baselines on complex catalyst design queries. For code-centric decision-making, GIF-MCTS (Dainese et al., 2024) leverages MCTS to generate, improve, and fix code world models that support model-based planning.

These systems commonly instantiate prior estimates from LLM judgments and use visit statistics to balance exploration and exploitation under budget limits, akin to PUCT (Silver et al., 2017). Our approach follows this principle while emphasizing autonomy-preserving orchestration: the tree controller allocates where to explore, but individual agents decide how to reason and which tools to invoke (cf. ReAct (Yao et al., 2022)). Compared to centralized auto-tuning controllers, tree-search-based agent systems provide structured, value-guided coordination that scales with agent capability and system budget. Orthogonal to tree search, AlphaEvolve (Novikov, 2025) demonstrates an evolutionary pipeline that iteratively proposes code edits and leverages external evaluators; both styles share the idea of structured exploration with feedback, while differing in how candidates are generated and selected.

2.3 Model Context Protocols and Tooling Ecosystem

Self-controlled agents increasingly rely on standardized interfaces to connect with external data and tools. The Model Context Protocol (MCP) provides an open specification for exposing tools and context to LLM applications (GitHub, 2024; Anthropic, 2024). On top of MCP, servers such as Context7 offer up-to-date, versioned documentation retrieval for code assistants (Upstash, 2024a;b), while privacy-focused search services (*e.g.*, DuckDuckGo Instant Answers) complement agent workflows with lightweight factual lookups (Duckduckgo, 2023). Agentic coding systems (*e.g.*, Codex (OpenAI, 2025) and Claude Code (Anthropic, 2025a;b)) further illustrate the trend toward protocol-driven tool use, persistent memory, and versioned development, motivating orchestration frameworks that preserve agent autonomy while coordinating system-level search.

3 Method: Tree-Based Orchestration with PUCT

We organize multiple self-controlled agents using a tree-based search that preserves autonomy while enabling global coordination, inspired by Aygün et al. (2025). Each tree node represents a decision state s; each outgoing edge corresponds to an agent-proposed action a (*e.g.*, a kernel transformation or schedule update). We maintain standard statistics per edge (s, a): visit count $N(s, a)$, accumulated value $W(s, a)$, mean value $Q(s, a) = \frac{W(s,a)}{N(s,a)}$, and a prior $P(s, a)$ provided at expansion.

Selection. From the root, selection recursively chooses the child that maximizes a PUCT objective until reaching a leaf (consistent with the PUCT formulation popularized in AlphaZero (Silver et al., 2017)):

$$\arg\max_a \; Q(s,a) \; + \; c\, P(s,a)\, \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}\,, \tag{1}$$

where $c > 0$ controls exploration (Silver et al., 2017). In practice, we use a shaped prior $\tilde{P}(s, a)$ (defined below) that incorporates parent-level experience:

$$\arg\max_a \; Q(s,a) \; + \; c\, \tilde{P}(s,a)\, \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}\,. \tag{2}$$

Expansion and Evaluation. If the selected node is not terminal and is not fully expanded, we request proposals from the responsible agent(s) to create new edges with priors $P(s, a)$. The leaf is evaluated to obtain a scalar value $V \in [0, 1]$ and (optionally) refined priors. In GEMM optimization, we derive V from measured elapsed time (lower is better) relative to a strong baseline; for instance,

$$V \;=\; \mathrm{clip}\Big(1 - \frac{\mathrm{time(candidate)}}{\mathrm{time(baseline)}}\,,\; 0,\; 1\Big)\,. \tag{3}$$

This normalization keeps values comparable across tasks and budgets.

Backup. Along the selected path, we update

$$N(s,a) \leftarrow N(s,a) + 1, \quad W(s,a) \leftarrow W(s,a) + V, \quad Q(s,a) \leftarrow \frac{W(s,a)}{N(s,a)}\,. \tag{4}$$

Algorithm 1 Pseudocode of the TrAgent Algorithm

Input: Task $task$, initial state $root\_state$, budget T; exploration constant c; shaping hyperparameters $m \in (0,1)$, $k > 0$, $r \geq 0$, $e > 0$
Output: Best action sequence discovered under budget

```
function TrAgent(task, root_state, T)
    init tree with root_state
    for t = 1..T do
        path ← SelectWithPUCT(tree)                          ▷ uses shaped prior
        leaf ← ExpandWithAgentProposals(path)
        V, priors ← EvaluateLeafWithAgent(leaf, task)
        Backup(path, V, priors)
    end for
    return BestActionSequence(tree)
end function

function SelectWithPUCT(tree)
    node ← tree.root
    while node is fully expanded and not terminal do
        a* ← arg max_a Q(node, a) + c·PriorTilde(node, a)· sqrt(N(node)) / (1+N(node,a))
        node ← child reached by a*
    end while
    return path to node
end function

function Backup(path, V, priors)
    for all (s, a) in path do
        N(s,a) ← N(s,a) + 1; W(s,a) ← W(s,a) + V; Q(s,a) ← W(s,a)/N(s,a)
        EXP(s,a) ← m · EXP(s,a) + (1 − m) · g(V)             ▷ parent-side experience
        if priors updated at s then
            P(s,·) ← priors(s,·)
        end if
    end for
    UpdateParentPriorsAlong(path)
end function

function PriorTilde(s, a)
    λ_s ← N(s)/(N(s) + k)
    return Normalize((1 − λ_s) P(s,a) + λ_s (N(s,a) + r EXP(s,a) + e))
end function
```

Experience mechanism and parent-level shaping. Vanilla PUCT uses fixed priors $P(s,a)$ (*e.g.*, from a policy head) and visit counts to drive exploration. In TrAgent, the parent node aggregates *experience* from its children and uses it to shape priors over time. Let $\mathrm{EXP}(s,a)$ denote an exponentially-smoothed success indicator for edge (s,a), updated during backup via

$$\mathrm{EXP}(s,a) \leftarrow m\,\mathrm{EXP}(s,a) + (1-m)\,g(V)\,, \quad g(V) \in [0,1]\,, \tag{5}$$

where $g(V)$ can be chosen as V (normalized value) or a banded indicator (*e.g.*, $\mathbb{1}\{V \geq \theta\}$). We then *blend* static priors with empirical evidence at the parent:

$$\tilde{P}(s,a) \;=\; \mathrm{Normalize}\big((1-\lambda_s)\,P(s,a) \;+\; \lambda_s\,(\,N(s,a) + \rho\,\mathrm{EXP}(s,a) + \varepsilon\,)\big)\,, \tag{6}$$

where $\lambda_s \in [0,1]$ increases with the parent visit count $N(s) = \sum_b N(s,b)$ (*e.g.*, $\lambda_s = \frac{N(s)}{N(s)+k}$), $\rho \geq 0$ weights experiential signals, and $\varepsilon > 0$ avoids degeneracy. Equation equation 6 treats $N(s,a)$ and $\mathrm{EXP}(s,a)$ as parent-side evidence, yielding a shaped prior $\tilde{P}(s,a)$ that gradually transitions from static policy mass to data-driven preferences as experience accrues. Substituting $\tilde{P}$ into Eq. equation 1 yields Eq. equation 2, improving stability (early exploration) and credit assignment (later exploitation) relative to vanilla PUCT, while preserving agent autonomy.

Autonomy-preserving design. We minimize over-control by (i) restricting the orchestrator to selection/backup and lightweight interfaces, (ii) letting agents decide how to use tools (compilation, profiling, search, memory, reflection), and (iii) feeding back outcomes (elapsed time, diagnostics) instead of prescriptive micro-instructions. The system therefore coordinates *what* to explore without dictating *how* agents reason.

Budgets and termination. Given a budget T ("rounds") of selection–expansion–evaluation–backup iterations, the search returns the best action sequence discovered. We also allow early stopping when an improvement threshold is not met over a patience window.

Connection to experiments. Our evaluation (Fig. 2) plots elapsed time versus rounds (§4), where one round equals a full PUCT iteration. We ablate c, tree depth/width, and autonomy features (reflection and memory toggles) to study efficiency and stability.

4 Experiments

We evaluate our tree-based orchestration system on GEMM kernel optimization, comparing against (i) a single self-controlled agent under two MCP variants, (ii) a random search baseline, and (iii) our system instantiated with two model families (codex-style and claude-code-style). We report elapsed time versus rounds, where rounds denote PUCT tree iterations (selection–expansion–evaluation–backup), averaging results over five runs with standard deviations.

4.1 Task Specification (Specification-Driven Development)

We adopt a specification-driven development (SDD) protocol to define the GEMM task and its evaluation. The objective is to optimize a GPU kernel for matrix multiplication $C = AB$ (FP16) under a clear, reproducible contract.

Objective. Given matrices $A \in \mathbb{R}^{M\times K}$, $B \in \mathbb{R}^{K\times N}$, implement an optimized CUDA kernel named `MyGemmKernel` in a single source file (`result.cu`) that minimizes profiler-reported total kernel cycles (*Elapsed Cycles* from Nsight Compute). Correctness is mandatory.

Inputs and layout. Inputs are positive integers M, N, K defining matrix dimensions. We store A, B, C in row-major order with leading dimensions $\mathrm{lda} = M$, $\mathrm{ldb} = K$, $\mathrm{ldc} = M$.

Constraints. External high-performance libraries (*e.g.*, cuBLAS/cuBLASLt/CUTLASS) are disallowed. CUDA intrinsics, `cp.async`, and Tensor Cores/`wmma` may be used. The kernel name is programmatically verified.

Correctness verification. We compare against a provided reference implementation (CPU or slower GPU) as ground truth. Only FP16 is considered; we require maximum absolute error $\leq 10^{-2}$ and maximum relative error $\leq 10^{-2}$ on element-wise outputs.

Optimization strategies. We encourage standard GPU optimization techniques that reduce *Elapsed Cycles*, including: (i) shared-memory tiling for locality and reuse; (ii) double buffering/pipelining with `cp.async` to overlap global-to-shared transfers with compute; (iii) register/shared-memory balance and occupancy control (*e.g.*, `__launch_bounds__`); (iv) coalesced accesses, loop unrolling, and instruction-level parallelism; and (v) Tensor Cores via `wmma` for FP16. Shared-memory padding should be used as needed to avoid bank conflicts.

Edge cases. Implementations must handle very small matrices (*e.g.*, $M = N = K = 1$) and scale to large shapes within CUDA resource limits. The computation must not introduce `NaN`/`Inf` artifacts.

Implementation notes. Thread-block tiling must respect shared-memory capacity. When using `cp.async`, *wait groups* should synchronize staged tiles correctly. `wmma` fragments should match tile geometry and data types for FP16.

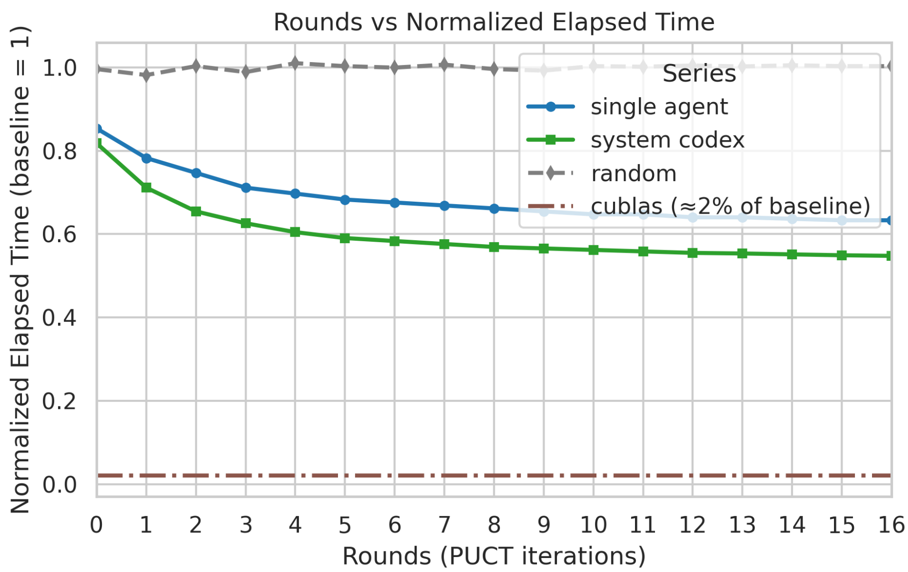

Figure 2: Rounds (PUCT iterations, x-axis) versus elapsed time (ms, y-axis) for single-agent (two MCP variants), random baseline, and our system with codex-style and claude-code-style agents. Lower is better.

Evaluation protocol. We use a provided harness (`evaluate.py`) to compile and validate kernels, and Nsight Compute to extract *Elapsed Cycles*. Results are averaged over five runs per point. All kernels are compiled with consistent flags; we report elapsed time and relative performance versus a strong library baseline for context.

In addition to the primary comparison, we conduct ablations on search parameters (*e.g.*, exploration constant, maximum depth/width) and autonomy features (reflection and memory toggles). We also discuss regimes where overhead dominates (small sizes) and memory-bandwidth limitations reduce attainable improvements.

Finally, we achieve 80% of the cuBLAS version.

5 Conclusion

We presented a tree-based orchestration system for organizing self-controlled agents via a PUCT-style search that preserves autonomy while enabling scalable coordination. On GEMM kernel optimization, the system outperforms single-agent baselines and a random search baseline, approaching a strong vendor library across representative settings. Our analysis highlights how search hyperparameters and autonomy features (reflection and memory) shape effectiveness.

Limitations & Future Work. Our results are limited to GEMM and do not exhaust all hardware or kernel classes; future work should evaluate broader operator suites and heterogeneous devices. Search overhead may hinder small workloads; adaptive budgeting and early stopping heuristics could improve efficiency. As agent capabilities grow, richer interfaces for value estimation and prior shaping may further accelerate exploration. Finally, investigating communication and credit assignment among specialized agents is a promising direction for scaling performance and robustness.

References

Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2014.

Anthropic. What is the model context protocol (mcp)? https://modelcontextprotocol.io/docs/getting-started/intro, 2024.

Anthropic. Claude code overview. https://docs.claude.com/en/docs/claude-code/overview, 2025a.

Anthropic. Claude code: Agentic coding tool (github). https://github.com/anthropics/claude-code, 2025b.

Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston Anton Kast, Cory Y McLean, Peter Norgaard, Zahra Shamsi, et al. An ai system to help scientists write expert-level empirical software. *arXiv preprint arXiv:2509.06503*, 2025.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. arXiv:2308.09687.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Ed Yan, Haichen Shen, Matt Cowan, Leyuan Wang, Yu Hu, Luis Ceze, et al. Tvm: An end-to-end open deep learning compiler stack. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, 2018a.

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018b. arXiv:1805.08166.

Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Zinner. End-to-end deep learning of optimization heuristics. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 277–288. IEEE, 2017.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. *arXiv preprint arXiv:2405.15383*, 2024.

Duckduckgo. Duckduckgo instant answers and features. https://duckduckgo.com/duckduckgo-help-pages/features/instant-answers-and-other-features, 2023.

GitHub. Model context protocol (mcp) – github organization. https://github.com/modelcontextprotocol, 2024.

Kazushige Goto and Robert A van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):1–25, 2008.

Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *NeurIPS*, 2023. arXiv:2303.17760.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Charith Mendis, Jonathan Ragan-Kelley, Saman Amarasinghe, and Michael Carbin. Ithemal: Accurate, portable and fast basic block throughput estimation. In *International Conference on Machine Learning*, pp. 4515–4525. PMLR, 2019.

et al. Novikov. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Lightweight coding agent that runs in your terminal. https://github.com/openai/codex, 2025.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 519–530, 2013.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023. arXiv:2303.11366.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Henry Sprueill, Carl Edwards, Mariefel Olarte, Udishnu Sanyal, Heng Ji, and Sutanay Choudhury. Monte carlo thought search: Large language model querying for complex scientific reasoning in catalyst design. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 8348–8365, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.560.

Upstash. Context7 mcp server (upstash). https://github.com/upstash/context7, 2024a.

Upstash. Context7: Up-to-date documentation for llms and ai code editors. https://context7.com/about, 2024b.

Field G. Van Zee and Robert A. Van De Geijn. BLIS: A framework for rapid instantiation of BLAS functionality. *ACM Transactions on Mathematical Software*, 2015.

R Clint Whaley, Antoine Petitet, and Jack J Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27(1-2):3–35, 2001.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS*, 2023. arXiv:2305.10601.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.14078.

Lianmin Zheng, Cheng Jia, Minmin Sun, Zhao Chen, Christos Kozyrakis, and Tianqi Chen. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 863–879, 2020.

Size Zheng, Shuo Liu, and Yida Wang. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous systems. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 157–170, 2022.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.