Agentic Recommender Systems: Foundations, Perspectives, and Lessons from Large Scale Deployments

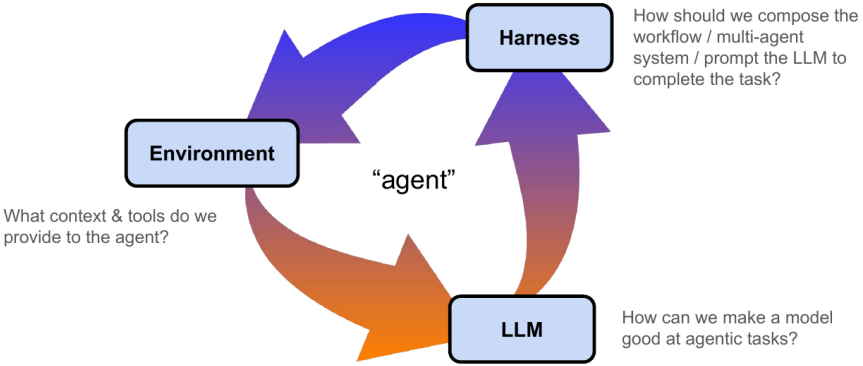
Reza Yousefi Maragheh[1] Yashar Deldjoo[2] Ben Coleman[3] Jason Cho[4] Chi Wang[5]

[1]Walmart Global Tech, [2]Polytechnic University of Bari, [3]Google DeepMind, [4]PretaskAI [5]AG2AI

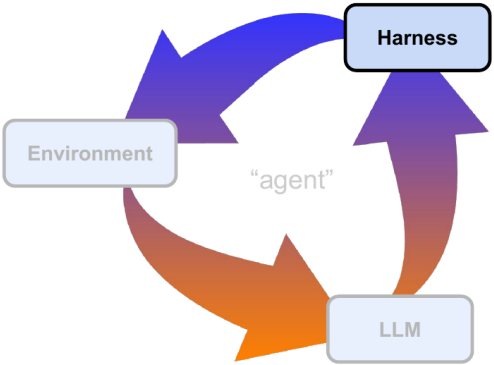
SIGIR 2026

Self Evolving Agents

What's in an agent?



What's in an agent?



Can often solve agent problems in multiple places.

Given a problem or research idea, it's useful to think about where to do the work.

Today, we'll mostly see harness stuff.

Roadmap: Memory, Experiment Loops, & Feature Engineering

1. **Agent memory** - how do we make sure the agent remembers everything it needs to know?

2. **Experiment loops** - how do we get the agent to pursue good research directions?

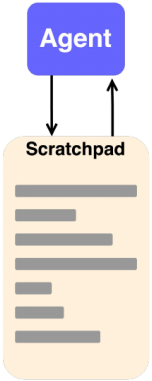
Then, let's look at a very concrete recsys application: feature engineering

In all three cases, we'll see how modifying the harness can solve problems.

Agent Memory

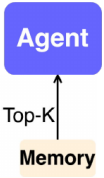
Context Accumulation & Retrieval

Agents accumulate* LOTS of context as they run. How to use the right context?



Some research tries to build a scratchpad like this

Other research tries to represent the context (e.g. as embeddings) and retrieve entries that are relevant to the task at hand

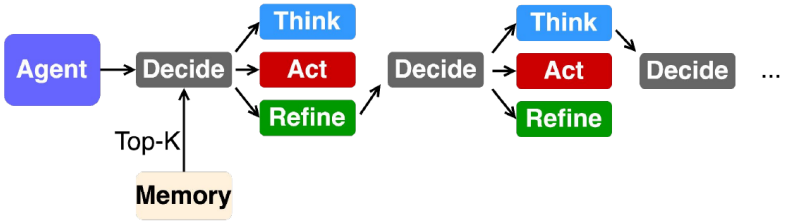


*Human researchers have accumulated a lot of context too. Think about all the stuff on arXiv!

Think-Act-Refine Baseline

Simple Memory Refinement Harness

Wanted to evaluate ideas (to use in applications) against a simple harness



Think: Generate a reasoning trace about the memory & task

Act: Stop messing around with memory and just do the task

Refine: Decide whether to discard some memories from the top-K

Answer uses the whole think-act-refine chain as context

But how do we know if it's any good?

At the time, no easy way to compare these ideas
(so we built one)

Tianxin Wei

Shoutout to our student researcher!

Google DeepMind 2025-11-27

Evo-Memory: Benchmarking LLM Agent Test-time Learning with Self-Evolving Memory

Tianxin Wei[1,†], Noveen Sachdeva[2], Benjamin Coleman[2], Zhankui He[2], Yuanchen Bei[1], Xuying Ning[1], Mengting Ai[1], Yunzhe Li[1,†], Jingrui He[1], Ed H. Chi[2], Chi Wang[2], Shuo Chen[2], Fernando Pereira[2], Wang-Cheng Kang[2] and Derek Zhiyuan Cheng[2]
[1]Work done while at Google DeepMind, [†]University of Illinois Urbana-Champaign, [2]Google DeepMind

Statefulness is essential for large language model (LLM) agents to perform long-term planning and problem-solving. This makes *memory* a critical component, yet its management and evolution remain largely underexplored. Existing evaluations mostly focus on static conversational settings, where memory is passively retrieved from dialogue to answer queries, overlooking the dynamic ability to accumulate and reuse *experience* across evolving task streams. In real-world environments such as interactive problem assistants or embodied agents, LLMs are required to handle continuous task streams, yet often fail to learn from accumulated interactions, losing valuable contextual insights, a limitation that calls for *test-time evolution*, where LLMs retrieve, integrate, and update memory continuously during deployment. To bridge this gap, we introduce Evo-Memory, a comprehensive streaming benchmark and framework for evaluating *self-evolving memory* in LLM agents. Evo-Memory structures datasets into sequential task streams, requiring LLMs to search, adapt, and evolve memory after each interaction. We unify and implement over ten representative memory modules and evaluate them across 10 diverse multi-turn goal-oriented and single-turn reasoning and QA datasets. To better benchmark experience reuse, we provide a baseline method, ExpRAG, for retrieving and utilizing prior experience, and further propose ReMem, an *action–think–memory refine* pipeline that tightly integrates reasoning, task actions, and memory updates to achieve continual improvement.

Keywords: LLMs, Agentic Memory, Test-time Learning, Self-evolving Agents, Lifelong Intelligence

How to "Evo-Memory" your benchmark

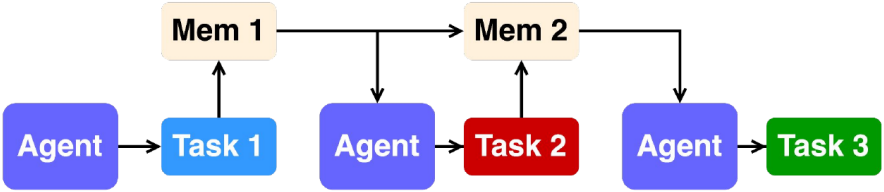
Step 1: Order the tasks in your benchmark into a sequence

(e.g., do MMLU problem #1 then #2, rather than in parallel)

Step 2: Allow the agent to maintain some state in between tasks

Step 3: Compare static performance with streamed-task performance

(the delta can be attributed to memory / experience reuse)



Think-Act-Refine performs surprisingly well

Performance Gains from Mem. Methods (Ours)

Consistent performance gains across models.

More significant gains on long-horizon complex tasks.

Claude 3.7 Sonnet

Method	AlfWorld		BabyAI		PDDL		ScienceWorld		Avg.	
	S	P	S	P	S	P	S	P	S	P
Baseline	0.18	0.49	0.51	0.66	0.17	0.39	0.10	0.53	0.24	0.52
History	0.50	0.73	0.48	0.66	0.65	0.85	0.32	0.74	0.49	0.74
ReAct	0.51	0.75	0.57	0.72	0.75	0.91	0.44	0.77	0.57	0.79
Amem	0.48	0.73	0.46	0.64	0.62	0.84	0.33	0.73	0.47	0.73
SelfRAG	0.52	0.75	0.46	0.64	0.65	0.84	0.31	0.74	0.49	0.74
Mem0	0.51	0.74	0.48	0.66	0.65	0.84	0.37	0.76	0.50	0.75
DC-Cu	0.50	0.74	0.50	0.67	0.62	0.84	0.33	0.75	0.49	0.75
DC-RS	0.50	0.74	0.52	0.68	0.62	0.84	0.34	0.74	0.50	0.75
AWM	0.49	0.73	0.53	0.68	0.60	0.82	0.34	0.74	0.49	0.74
ExpRecent	0.66	0.83	0.63	0.73	0.53	0.76	0.49	0.82	0.58	0.79
ExpRAG	0.74	0.89	0.62	0.72	0.72	0.89	0.46	0.76	0.63	0.82
ReMem	**0.92**	**0.96**	**0.73**	**0.83**	**0.83**	**0.95**	**0.62**	**0.89**	**0.78**	**0.91**

Takeaway

Lesson: harness modifications can be really powerful

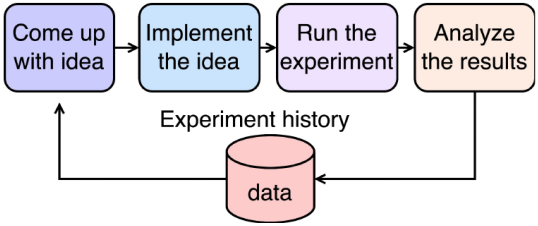
- Simple setups beat complex methods if you get the abstractions right

- For memory retrieval + processing, “think-act-refine” seems like a solid workflow decomposition

Experiment Loops

AutoResearch & Evolution Workflows

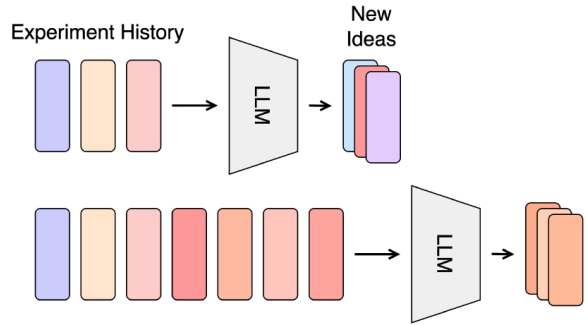
Experiment loops are another useful harness - think of AlphaEvolve, AutoResearch, etc



Challenges in Evolution

Context Pollution & Loss of Diversity

Context pollution decreases diversity of thought - less creative over time



Self-reinforcing feedback loop since LLM produces its own context

Exploration Bottlenecks

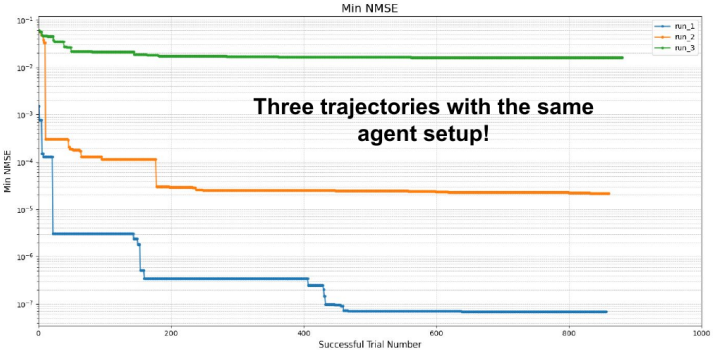
Stagnation in Long-Horizon Runs

```
- Results: nmse: 0.0476, ood_nmse: 0.00347. The performance degradation compared to the static symmetric model (`ood_nmse`: 0.
- Candidate tested if quadratic drag (`~v*abs(v)`) is a near-equilibrium phenomenon by multiplying it with a spatial gating fu
- Results: nmse: 0.0477, ood_nmse: 0.00388. The degradation in generalization compared to the model with an ungated quadratic
- A novel hysteresis model was tested by making the saturation rate of the position-dependent damping term (`~tanh(k*abs(x))`)
- Results: nmse: 0.0476, ood_nmse: 0.00347. The performance degradation compared to the static symmetric model (`ood_nmse`: 0.
- Candidate made the amplitude of the successful saturating position-dependent damping term (`~v*tanh(k*abs(x))`) dependent on
- Results: nmse: 0.04657, ood_nmse: 0.00335. The performance degradation compared to the static model (ood_nmse: 0.00167) sugg
- Candidate regularized the unbounded quadratic drag term (`~v*abs(v)`) with a velocity-dependent rational function (`~1/(1+k*
- Results: nmse: 0.0487, ood_nmse: 0.00322. The worsened generalization (vs. 0.00167 from the model with an unbounded term) in
- Candidate replaced the absolute position `abs(x)` argument in the saturating damping term `~v*tanh(k*abs(x))` with the magni
- Results: nmse: 0.0476, ood_nmse: 0.00371. The significant performance degradation (vs. 0.00167 from the `abs(x)` model) refu
- Candidate tested a symmetric, speed-dependent skew (`~abs(v)/(1+k*v**2)`) in the argument of the saturating damping term (`~
- Results: nmse: 0.0489, ood_nmse: 0.00332. Generalization worsened compared to the model with purely position-dependent satur
- Candidate tested if the saturation rate of the position-dependent damping term (`~tanh(k*abs(x))`) was dynamically modulated
- Results: nmse: 0.0476, ood_nmse: 0.00414. Generalization worsened compared to the static saturation model (ood_nmse: 0.00167
- Candidate tested a bounded, hysteretic damping model by introducing a velocity-dependent skew (`~abs(x) + c*tanh(d*v)`) to t
- Results: nmse: 0.0434, ood_nmse: 0.00361. The significant performance degradation compared to the static symmetric model (`~
- Candidate tested for a viscoelastic restoring force by modulating the linear spring constant with a bounded function of velo
- Results: nmse: 0.0489, ood_nmse: 0.00294. The performance degradation (vs. 0.00167 from the best-performing model with purel
- Candidate tested if quadratic drag (`~v*abs(v)`) is a large-displacement phenomenon by multiplying it with a `tanh(k*x**2)`
- Results: nmse: 0.0493, ood_nmse: 0.00276. The performance degradation (vs. 0.00167 from the ungated model) refutes the gatin
```

Eventually, the agent stops trying new things entirely...

This results in high variance

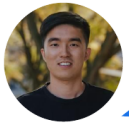
Very noisy process - sometimes you discover something, sometimes you don't



Better: Down + Left

Progress-Aware Consistent Evolution

Introducing: PACEvolve



Minghao Yan

Shoutout to our student researcher!

PACEvolve: Enabling Long-Horizon Progress-Aware Consistent Evolution

Minghao Yan [†1 2] Bo Peng [*†1] Benjamin Coleman [*3] Ziqi Chen [1] Zhouhang Xie [†1 4] Shuo Chen [3] Zhankui He [3] Noveen Sachdeva [3] Isabella Ye [1] Weili Wang [1] Chi Wang [2] Ed H. Chi [3] Fernando Pereira [3] Wang-Cheng Kang [3] Derek Zhiyuan Cheng [3] Beidou Wang [1]

Abstract

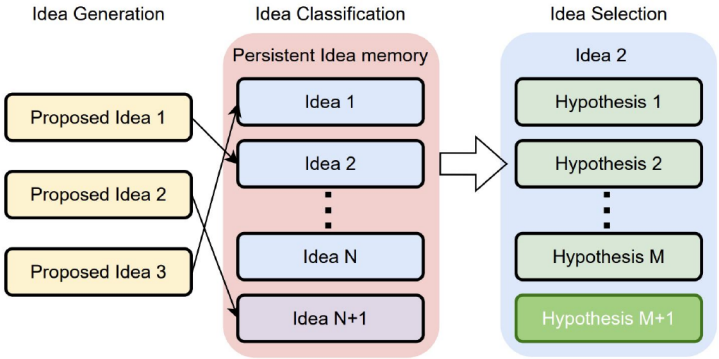
Large Language Models (LLMs) have emerged as powerful operators for evolutionary search, yet the design of efficient search scaffolds remains ad hoc. While promising, current LLM-in-the-loop systems lack a systematic approach to managing the evolutionary process. We identify three distinct failure modes: *Context Pollution*, where experiment history biases future candidate generation; *Mode Collapse*, where agents stagnate in local minima due to poor exploration-exploitation balance; and *Weak Collaboration*, where rigid crossover strategies fail to leverage parallel search trajectories effectively. We introduce **Progress-Aware Consistent Evolution** (PACEvolve), a framework designed to robustly govern the agent's context and search dynamics, to address these challenges. PACEvolve combines hierarchical context management (HCM) with pruning to address context pollution; momentum-based backtracking (MBB) to escape local minima; and a self-adaptive sampling policy that unifies backtracking and crossover for dynamic search coordination (CE), allowing agents to balance internal refinement with cross-trajectory collaboration. We demonstrate that PACEvolve provides a

and engineering problems (Novikov et al., 2025; Romera-Paredes et al., 2024; Lange et al., 2024; Cheng et al., 2025b; Lange et al., 2025). They transform evolutionary search by replacing the rigid, random operators of classical algorithms (such as mutation and crossover) with intelligent, context-aware reasoning (Novikov et al., 2025; Romera-Paredes et al., 2024; Lange et al., 2025). Unlike traditional Evolutionary Algorithms (EAs) that evaluate an extensive number of weakly-guided candidates ($> 10^6$ samples) (Fogel, 1988; Holland, 1992), LLM-driven agents leverage in-context evolution history to perform iterative refinement (Novikov et al., 2025). By treating the history as a dynamic knowledge base, these agents can theoretically learn from failures and perform meta-reasoning, shifting the paradigm toward sample-efficient, knowledge-guided optimization (Zhai et al., 2025; Lange et al., 2025).

Our work aims to use these intelligent search priors to unlock state-of-the-art performance in complex research and engineering tasks (Shojaee et al., 2024; Ouyang et al., 2025a; Jordan et al., 2024). However, this new LLM-in-the-loop paradigm introduces significant instability, preventing the search from consistently leveraging the LLM's reasoning capabilities (Xia et al., 2025; Kim et al., 2025). Rather than steadily improving, these systems suffer from high variance (§2), often failing to produce reliable improvements due to the combined stochasticity of the LLM and the search process (Comanici et al., 2025; Renze, 2024).

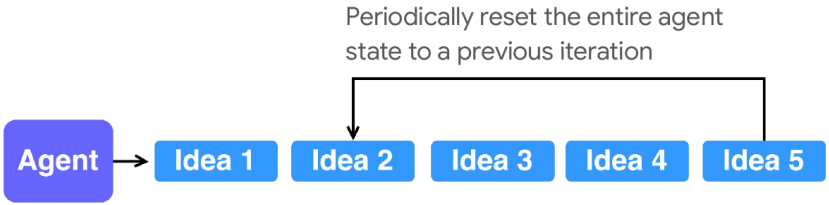
Workflow decomposition strikes again

Another workflow decomposition - separate ideation and selection into subagents



Plus other harness heuristics

We also do a few other things, such as backtracking when the system gets stuck



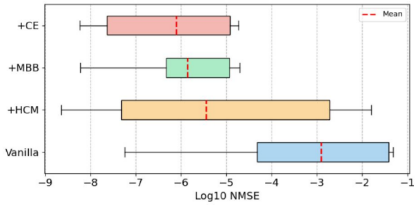
We do this based on a momentum heuristic (details in paper)

PACEvolve Results

SOTA Performance & Variance Reduction

By adopting this workflow decomposition, we can:

reduce the variance a lot and get SOTA results*



*See paper for more results - KernelBench, GPT-speedrun, etc

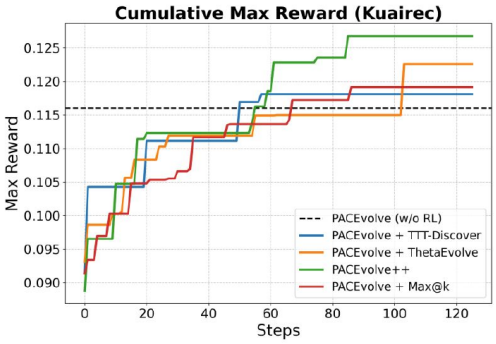
Table 1. Performance comparison on LLM-SR task. We report the best, P75, mean, and worst performance across 10 runs in terms of Log10 Normalized Mean Squared Error (Log10 NMSE). Best results are in **bold**, second-best results are underlined.

Method	Best	P75	Mean	Worst
uDSR	-4.06	-4.06	-3.95	-3.73
LLM-SR	-4.80	-4.03	-4.06	-3.62
OpenEvolve	-7.11	-5.79	-5.40	-4.02
CodeEvolve	-7.26	-5.54	-4.97	-3.99
ShinkaEvolve	-6.35	-5.92	-5.35	-3.42
PACEvolve-Single	-8.23	-6.33	-5.87	-4.71
PACEvolve-Multi	**-8.24**	**-7.64**	**-6.11**	**-4.73**

Can we solve the problem in the LLM rather than the harness?

We tried RL on the ideation model to let the LLM learn from experiment success / failures throughout the evolution run

Reward shaping and stability are challenging* but early results do show gains



*See PaceEvolve++ paper for details

Semantic feature tagging in RecSys

Now let's see a more concrete recsys application: feature engineering

Item Description

Limited edition 40th Anniversary vinyl, great audio quality...

Item ID & Semantic IDs

358, 271, 77, 82 — **Depends on encoder quality, uninterpretable*

Tag Features

Rock → Glam Rock → Proto Punk — **Exploits LLM reasoning, interpretable, NL native*

We want the LLM to produce a hierarchy of useful semantic tags

A naive agent / workflow does not work well

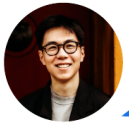
If we just ask an LLM (or agent) to generate tags, we get

- long, one-off tags
  "hardwood desk for computer monitor in home office"
- inconsistent / soft-duplicates
  "camera bag" versus "camera case"
- way too many tags
  Tag cardinality > number of items

Tags with these characteristics are hard to use in production and don't drive semantic generalization.

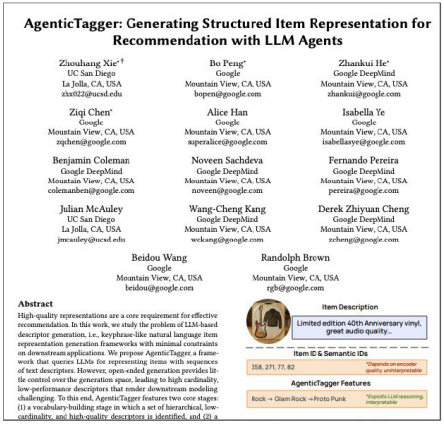
Structured semantic tag generation with AgenticTagger

Introducing:
AgenticTagger



Zhouhang Xie

Shoutout to our student researcher!



AgenticTagger: Generating Structured Item Representation for Recommendation with LLM Agents

Zhouhang Xie*†
UC San Diego
La Jolla, CA, USA
zhx022@ucsd.edu

Bo Peng*
Google
Mountain View, CA, USA
bopen@google.com

Zhankui He*
Google DeepMind
Mountain View, CA, USA
zhankui@google.com

Ziqi Chen*
Google
Mountain View, CA, USA
zqchen@google.com

Alice Han
Google
Mountain View, CA, USA
superalice@google.com

Isabella Ye
Google
Mountain View, CA, USA
isabellasye@google.com

Benjamin Coleman
Google DeepMind
Mountain View, CA, USA
colemanben@google.com

Noveen Sachdeva
Google DeepMind
Mountain View, CA, USA
noveen@google.com

Fernando Pereira
Google DeepMind
Mountain View, CA, USA
pereira@google.com

Julian McAuley
UC San Diego
La Jolla, CA, USA
jmcauley@ucsd.edu

Wang-Cheng Kang
Google DeepMind
Mountain View, CA, USA
wckang@google.com

Derek Zhiyuan Cheng
Google DeepMind
Mountain View, CA, USA
zcheng@google.com

Beidou Wang
Google
Mountain View, CA, USA
beidou@google.com

Randolph Brown
Google
Mountain View, CA, USA
rgb@google.com

Abstract

High-quality representations are a core requirement for effective recommendation. In this work, we study the problem of LLM-based descriptor generation, i.e., keyphrase-like natural language item representation generation frameworks with minimal constraints on downstream applications. We propose AgenticTagger, a framework that queries LLMs for representing items with sequences of text descriptors. However, open-ended generation provides little control over the generation space, leading to high cardinality, low-performance descriptors that render downstream modeling challenging. To this end, AgenticTagger features two core stages: (1) a vocabulary-building stage in which a set of hierarchical, low-cardinality, and high-quality descriptors is identified, and (2) a

Item Description

Limited edition 40th Anniversary vinyl, great audio quality...

Item ID & Semantic IDs

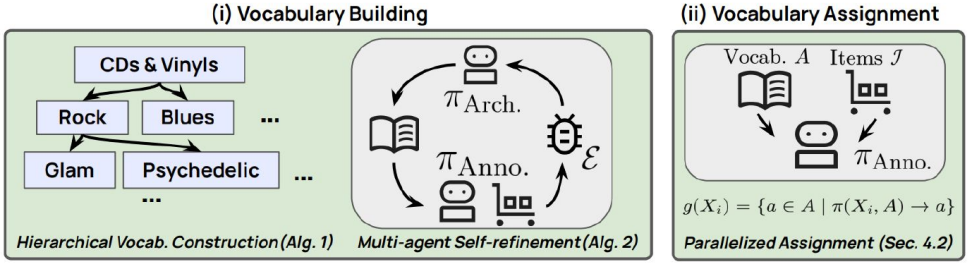
158, 271, 77, 82 *Depends on encoder quality, uninterpretable

AgenticTagger Features

Rock → Glam Rock →Proto Punk *Exploits LLM reasoning, interpretable

AgenticTagger Architecture

The core approach* is again a multi-agent workflow decomposition



*making many simplifications for this talk - see paper for details

AgenticTagger Results

Vocabulary Control & Downstream Gains

Adopting this vocabulary building + assignment setup fixes the problems

Item Context	Free-form	AgenticTagger
Ground Truth *Transformer stock, 820% 3-yr returns*	Skipper Limited, Order Book Growth, Power Transmission.	**L0:** Business & Finance **L1:** Stock Market **L2:** Individual Performance
User History (1) *Defence bidder, 7k% 5-yr returns* (2) *Industrial PSU, 200% 5-mo returns*	(1) Defence PSU, DRDO Bidding. (2) PSU Stocks, Market Returns.	**L0:** Business & Finance **L1:** Stock Market **L2:** Individual Performance, Order Book Analysis

All three of these papers have something in common

Takeaway: We can solve problems in the environment, harness, or model. Multi-agent workflow decomposition is a powerful tool to solve problems in the harness.

- While not truly “multi-agent,” the think-act-refine loop is surprisingly effective at memory retrieval + management
- Decomposing the auto-research loop into ideation + selection helped us resolve context pollution problems
- Decomposing the feature engineering workflow into vocabulary building + assignment helped us resolve tagging problems