Concept Paper

AgenticRS: Agentic Recommender Systems

Jinxin Hu [*,†] , Hao Deng [†] , Lingyu Mu , Hao Zhang , Shizhun Wang , Yu Zhang , Xiaoyi Zeng

Posted Date: 25 March 2026

doi: 10.20944/preprints202603.2003.v1

Keywords: recommender systems; agent; large language model

 doi:10.20944/preprints202603.2003.v1

Concept Paper

AgenticRS: Agentic Recommender Systems

Jinxin Hu[1,*,†], Hao Deng [1,†], Lingyu Mu [2], Hao Zhang [1], Shizhun Wang [1], Yu Zhang [1] and Xiaoyi Zeng [1]

[1] Alibaba International Digital Commerce Group, Beijing, China
[2] University of Chinese Academy, Beijing, China
* Correspondence: jinxin.hjx@alibaba-inc.com
† Equal contribution.

Abstract

Large-scale recommender systems in industry commonly adopt a mature multi-stage architecture of recall, coarse ranking, fine ranking and re-ranking, and have evolved from collaborative filtering to deep neural networks and large pre-trained models. Nevertheless, both multi-stage pipelines and unified One Model designs remain essentially static: models are treated as black-box components, and system optimization chiefly relies on manually proposed hypotheses and iterative engineering, making it difficult to achieve continual, autonomous evolution under highly heterogeneous data and multi objective business constraints. This paper introduces the notion of an Agentic Recommender System (AgenticRS), which reorganizes decision units in recommendation through an agentic perspective. We distinguish between functional agents and model agents, and specify three constraints—functional closed loop, independent evaluability and evolvable decision space—to determine which modules should be designed as agents. For model agents, we present two complementary self evolution mechanisms: reinforcement learning based optimization when the action space is well-defined, and large language model (LLM) based generation and selection of new model and training schemes when the design space is complex and experience-driven. We further separate individual evolution and compositional evolution to describe both the improvement of a single agent and the evolving selection, connection and cooperation among multiple agents, and propose a layered Inner and Outer reward design to balance local capability optimization with global objective alignment. The proposed framework aims to provide a concise and general design foundation for transforming static recommendation pipelines into multi-agent, self-evolving recommender systems.

Keywords: recommender systems; agent; large language model

1. Introduction

Over the past two decades, large-scale recommender systems have made significant progress in both system architecture and model capability, and have become core infrastructure for internet products [2,11,14,19,20,23]. At the model level, recommendation techniques have undergone several key technological iterations as shown in Figure 1. The earliest generation is represented by collaborative filtering [16], which mines similar users or items from the user–item interaction matrix and directly models the intuition that users with similar preferences tend to like similar content. As data volume grew and sparsity increased, pure neighborhood methods exhibited limitations in scalability and generalization, and the second generation shifted to matrix factorization [8] and its variants. By learning low-dimensional latent vectors to represent users and items, these methods embed the sparse rating matrix into a dense latent space, significantly improving prediction accuracy and scalability. At the same time, factorization machines [15] and their extensions explicitly encode feature interactions as inner products, becoming an important foundation for industrial CTR prediction. With the advent of deep learning, recommendation models experienced a third leap forward. Embedding-based architectures combined with multilayer perceptrons have been widely adopted for click-through

 doi:10.20944/preprints202603.2003.v1

rate and conversion prediction, such as Wide&Deep [1], DeepFM [4] and xDeepFM [10]. These models introduce nonlinear interactions and automatic feature combination on top of traditional linear models, greatly reducing the burden of manual feature engineering. Subsequently, to better capture user behavior sequences and content semantics, sequence models and attention mechanisms were systematically introduced into recommendation: RNNs [5], CNNs and Transformers [7] are used to model the temporal evolution of user interests, while graph neural networks capture high-order user–item relations and propagation effects, further enhancing the modeling of complex behavior patterns [14,28]. At the system level, industrial practice has evolved from early single-model or simple two-stage architectures to a highly engineered multi-stage pipeline. In this pipeline, the recall stage first selects several candidate subsets from a massive item pool and a coarse-ranking model then performs a low cost pre-filtering. Then, a fine-ranking model follows with richer features and more complex structures to assign refined scores. Finally, re-ranking and policy modules adjust diversity, freshness and business constraints in a unified manner. This layered architecture of recall, coarse ranking, fine ranking and re-ranking has become the standard in search, feeds, e-commerce and short-video applications.

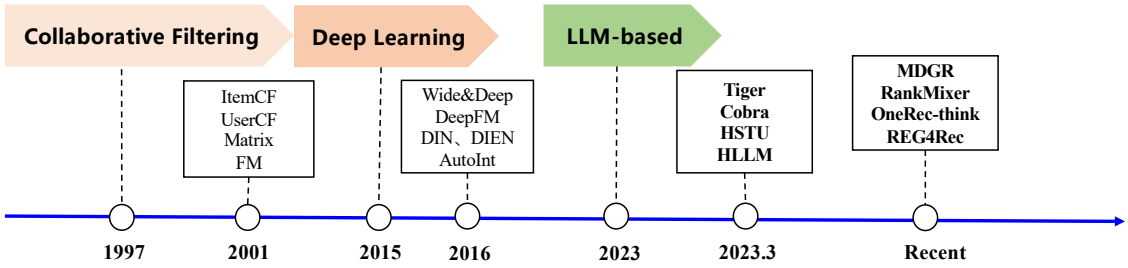

Figure 1. Technological Evolution of Recommendation Systems.

In recent years, with the rise of pre-training and large model techniques, recommendation models have been entering a new stage characterized by large-scale pre-training and generative modeling [3,6,9,12,13,21,22,24,27]. Pre-trained text and multimodal models have been integrated into recommendation pipelines to provide unified semantic representations for product descriptions, short video content, and user-generated text, significantly improving the ability to handle cold-start cases, cross-modal understanding, and intent interpretation. On the other hand, generative recommendation and One-Model style architectures attempt to use a single large model to jointly perform candidate generation, ranking, and multi-task prediction, compressing functions that were previously distributed across multiple recall routes and ranking stages into a single end-to-end trainable parameter space [27]. In many real-world scenarios, such methods have already demonstrated performance gains over traditional multi-model solutions. However, from the perspective of system design and evolution, current large-model-based recommendation approaches still face several critical issues. First, large models are typically embedded into existing pipelines as larger monolithic components, for example by replacing a ranking stage or adding a generative module in the recall stage. While this improves local predictive accuracy, it does not change the optimization and evolution mechanism of the overall recommendation system, which still relies on humans to propose hypotheses, manually modify configurations, and launch experiments. Second, One-Model style unified architectures, while enhancing expressiveness, further increase system coupling and reduce interpretability. When functions such as recall, ranking, and multi objective prediction are compressed into a single model, any local issue can only be fixed along the way through global architectural changes and full retraining, making it difficult to target specific sub-capabilities for controlled local evolution. In addition, One-Model essentially attempts to use a single unified model to cover all user groups and scenarios, whereas real-world business data are highly heterogeneous. To maintain overall performance, updates of the single large model are often dominated by long sequences and high frequency patterns, leading to insufficient modeling

 doi:10.20944/preprints202603.2003.v1

of cold-start users, short-sequence users, and long-tail items. Consequently, even with substantially increased model capacity, the overall recommendation system remains, in essence, a static pipeline driven by manual iteration.

To this end, we argue that simply stacking larger models into existing pipelines is no longer sufficient to break through the evolutionary bottleneck of recommender systems. Both the multi-stage architecture of recall, coarse ranking, fine ranking and re-ranking, and the attempts to use a single One Model to undertake all functions share a common property: the system structure is essentially fixed at design time, and each module is treated as a relatively static functional block whose subsequent improvements rely mainly on human engineers to propose hypotheses and modify models or policies. The models themselves may become increasingly large, but the entire recommender system still behaves more like a passive production line than an active entity that can sense environmental changes, reorganize its internal structure, and continuously improve itself. To address these limitations, we revisit the development direction of recommendation technology from a system-level perspective. As shown in Figure 2, instead of viewing the recommender system merely as a pipeline composed of a set of fixed modules, we abstract those model components that possess a closed decision loop into agents that can continuously learn and reshape themselves, and thereby construct a new Agentic Recommender System (AgenticRS) [17,26]. The motivation behind this proposal lies primarily in three aspects:

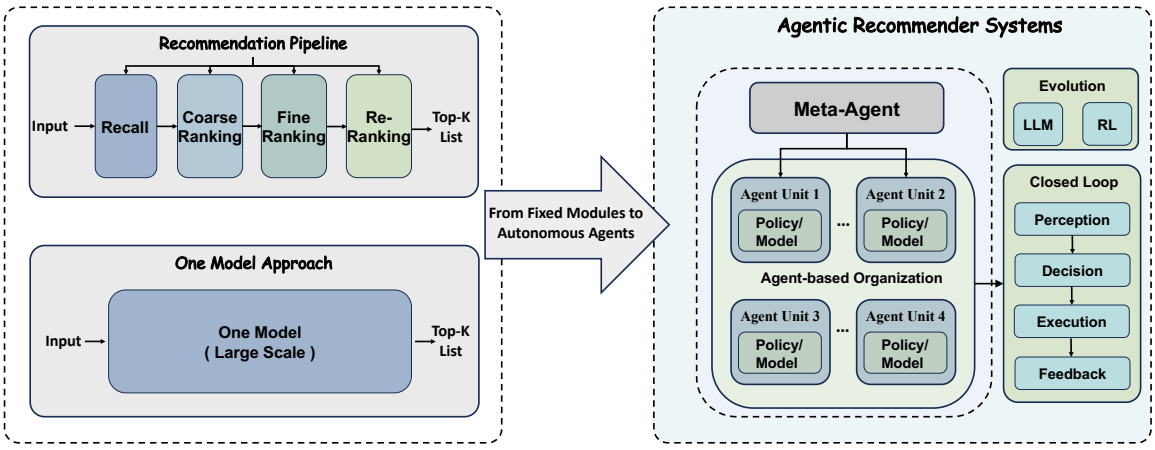

Figure 2. The Agentic Recommender Systems paradigm.

- **Fine-grained units of evolution.** In current practice, a model is often treated as an indivisible whole, where any modification implies changing the entire architecture, retraining the entire model, and deploying it as a whole. We aim to identify, inside the model, those units that are functionally self-contained, have clear interfaces, and can be independently evaluated and replaced, and to abstract them as agents that can evolve over the long term. In this way, we obtain an evolutionary granularity that is finer and more controllable than treating the whole model as a single unit.
- **Autonomous learning and adaptation.** Under the traditional paradigm, decisions about how to modify a model are almost entirely made by human researchers: engineers detect problems, design changes, implement them, and validate the results. With today's powerful large models and agents that encode extensive world knowledge, a model unit that is designed as an agent naturally emphasizes a closed loop of perception, decision, feedback, and adjustment. Within its own scope of responsibility, it can continuously monitor the environment and its own performance and adapt its internal structure or policies based on feedback, rather than relying solely on external manual intervention. This provides the structural foundation for recommender systems to achieve more automated and sustained optimization during long-term operation.

 doi:10.20944/preprints202603.2003.v1

- **Capability for structured organization under diverse distributions and multiple objectives.** In modern recommendation scenarios, users and content are highly heterogeneous, and different user groups or item subspaces often call for different modeling approaches. At the same time, the system must balance multiple objectives such as short-term clicks versus long-term retention, growth versus safety. We aim to use multiple agents with clearly defined responsibilities to explicitly handle different data sub-distributions and sub-goals, and let the system at a higher level decide which agents to activate in which scenarios and how to combine their outputs. In this way, we structurally mitigate the excessive coupling inherent in One Model style designs and provide clearer organizational units for coordinating complex and potentially conflicting objectives.

From this high-level perspective, an AgenticRS is not simply an extra agent layer wrapped around an existing system. Instead, it treats agents as the new fundamental abstraction for reorganizing the model structure and decision process of recommender systems: which capabilities are separated as independent units, how they coordinate within the system, and how they adjust themselves over time. We argue that only under such an agent-centered system design can large-scale recommender systems truly move beyond the current paradigm and evolve into the next generation of systems that can autonomously adapt over the long term in complex environments.

This paper is a concept and methodology oriented study of system design. Its core objective is to provide a unified abstraction and design framework for constructing an AgenticRS, rather than proposing a specific new model or a single algorithm. To this end, we first present four basic principles for designing an AgenticRS, with the aim of offering a practical blueprint that can guide next generation recommender systems in transforming from static model pipelines into intelligent systems where multiple agents coevolve and collaborate.

2. Related Work

2.1. Recommender Systems

Research on recommender systems has undergone several major phases over the past two decades. Early work was dominated by collaborative filtering, which exploited user–item interaction matrices to infer preferences from neighborhoods of similar users or items [5]. With growing data sparsity and scale, latent factor models such as matrix factorization [8] and factorization machines [15] became the de facto standard, embedding users and items into low-dimensional spaces and modeling feature interactions for tasks like CTR prediction. The success of deep learning further pushed the field toward representation-learning–driven architectures, including MLP-based models such as Wide&Deep [1] and DeepFM [4], as well as sequence- and attention-based methods that capture temporal user behavior and high-order relations via RNNs, Transformers, and graph neural networks [18]. At the system level, industrial deployments have converged on multi-stage pipelines comprising recall, coarse ranking, fine ranking, and re-ranking, which have proven effective for large-scale search, feeds, and e-commerce applications [11,19,20]. More recently, pre-trained and generative models have been introduced into these pipelines to provide unified semantic representations and to enable one-model or generative recommendation paradigms [12,24,25]. However, despite these advances in model capacity and architecture, most systems are still organized as static pipelines whose modules are manually designed and updated, lacking explicit support for autonomous evolution. Our work addresses this gap by re-examining recommender system design from an agentic perspective and proposing a framework in which key decision units are modeled as self-evolving agents.

2.2. Agent

The idea of agent has long been central in artificial intelligence, where agents are typically defined as autonomous entities that perceive their environment, make decisions, and act to maximize some notion of reward over time. Early work in agent-based AI and reinforcement learning formalized this perspective using Markov decision processes and policy optimization, laying the groundwork

 doi:10.20944/preprints202603.2003.v1

for single-agent and multi-agent systems that learn through trial and error in dynamic environments. As computational resources and algorithms matured, research expanded from isolated agents to multi-agent systems that coordinate, compete, or cooperate, with dedicated mechanisms for communication, credit assignment, and joint policy learning. In parallel, the emergence of large language models has sparked a new wave of agentic research: LLMs are increasingly used as general-purpose decision engines that can interpret instructions, reason over unstructured information, and invoke tools in closed perception–action loops. Recent frameworks, such as ReAct-style reasoning-and-acting agents and Reflexion-style self-reflective agents [17,26], demonstrate that agents can improve over time by leveraging external tools and internal memory rather than static model inference alone. This trajectory marks a shift from viewing models as passive predictors to treating them as active, adaptive decision makers embedded in broader systems. Building on this evolution, our work brings the agentic perspective into large-scale recommender systems, focusing on how to define, constrain, and organize agents so that recommendation models and policies can autonomously adapt within complex production environments.

3. Principle 1: The Definition and Boundaries of Recommender Agents

In traditional recommender systems, the boundaries among modules, services, and models are predominantly dictated by engineering implementations. Components such as retrieval, ranking, policy control, and experimental platforms are typically implemented as mutually independent services or function libraries; their respective responsibilities and boundaries largely reflect code partitioning and team division of labor. Under the AgenticRS paradigm, adhering to the conventional intuition that one module equates to one agent will inevitably lead to severe over-agentification. On the one hand, the system becomes saturated with nominal agents whose responsibilities and capability boundaries are difficult to delineate clearly. On the other hand, it becomes exceedingly challenging to investigate the collaborative relationships among these agents and their mechanisms of self evolution over time. Therefore, prior to adopting an agentic perspective, a fundamental yet easily overlooked question must be addressed: Within a recommender system, what types of modules ought to be designed as agents? This section will present a taxonomy of recommender agents, outline the constraints they must satisfy, and discuss how to determine the appropriate granularity of agents within complex systems.

3.1. Types of Recommender Agents: Functional vs. Model-Centric

From a holistic perspective of recommender systems, we categorize agents within recommendation scenarios into two primary types: functional agents and model-centric agents. (i) **Functional agents** are defined by a closed-loop business function, typically spanning multiple underlying models or tools to manage the end-to-end cycle of "decision proposal, workflow organization, and result interpretation." Typical instances include Strategy Design and Experiment Orchestration Agents, which propose policy configurations and manage traffic allocation, and Traffic Scheduling and User Segment Allocation Agents, which route requests across various recommendation pipelines based on user characteristics. Conversely, (ii) **Model-centric agents** are defined by independently evolvable models or sub-models. Their core competency lies in executing specific predictive or representation learning tasks while continuously optimizing within a stable systemic context. Representative examples include specific retrieval models (e.g., interest-based, content-based, or social retrieval), specialized sub-towers within a main ranking model (e.g., long-sequence or cold-start towers), and internal sub-structures such as routing modules in Mixture-of-Experts (MoE) layers. Ultimately, functional agents focus on system-level decision-making and process orchestration, whereas model-centric agents emphasize granular modeling and predictive capabilities.

3.2. Three Essential Constraints for Recommender Agents

Not all modules should be designed as agents. We posit that a module should only be regarded as an independent agent within an AgenticRS if it simultaneously satisfies the following three conditions:

 doi:10.20944/preprints202603.2003.v1

Closed-loop Functionality. The module must establish a relatively complete decision-making closed loop within the system, characterized by four key attributes: first, it possesses clear and stable input interfaces for perceiving states or environmental contexts. Second, it performs internal reasoning or computation to output decisions or recommendations, such as scoring, ranking, routing, or configurations. Third, its outputs are executed or adopted by the system, exerting a tangible impact on user exposure or downstream processes. At last, the agent can leverage the feedback from these execution results—such as via training samples, logs, or statistical features—to inform subsequent decisions. In other words, an agent is not merely an utility function. It is a functional unit that persistently participates in the "perception–decision–execution–feedback" cycle.

Independent Evaluability. The behavior of the module must be governed by relatively independent and measurable criteria for success. Specifically there should exist at least one set of metrics or rewards closely aligned with its responsibilities that are primarily determined by the actions of the agent itself. It must be possible to compare different versions of the agent through local A/B testing or offline evaluation without significantly interfering with other agents in the system. Even if final business indicators are a collective responsibility of the entire system the agent must maintain a clear local evaluation perspective. Only when independent evaluability is established can an agent undergo meaningful selection replacement and evolution during system operation.

Evolvable Decision Space. A recommender agent should not be a static configuration but rather an entity capable of autonomous improvement during system operation. This requirement dictates that the agent must be designed with an adjustable decision space. For model-centric agents this involves allowing for modifications to model architectures loss function combinations parameters or input feature selections. For functional agents it entails the ability to adjust decision strategies workflow sequences retrieval or model calling policies and various thresholds or weights. Furthermore there must exist a learning or search mechanism for updating the agent such as reinforcement learning or adaptive optimization that automatically tunes structures and parameters based on feedback signals. This may also involve leveraging large language models and empirical memory to generate new settings or rules which are then preserved if they prove superior through experimental validation. If a module is designed as hard coded logic with no intention or mechanism for updates it is better categorized as an internal tool of an agent rather than an independent agent itself within the agentic perspective.

3.3. Agent Granularity

The three aforementioned constraints provide the principles for identifying potential agents yet within a complex recommender system this definition still allows for a vast hierarchy of candidates ranging from a global retrieval module to a specific retrieval model then down to an internal Mixture of Experts layer or even a single expert and its corresponding loss term. Labeling all these units as agents without discrimination would lead to the previously discussed issue of over-agentification. To address this we propose two supplementary principles regarding agent granularity from a practical perspective.

The formation of replaceable and evaluable decision units. A reasonable agent should generally fulfill several core criteria. It must repeatedly execute a perception-decision-execution-feedback cycle on its own time scale rather than functioning as a one-off function invoked by occasional calls. It should allow for the comparison of different versions through local experiments while keeping the rest of the system constant such as replacing a specific retrieval model or a sub-tower. Furthermore it should correspond to a relatively stable node within the system architecture diagram rather than being a minor detail nested deep within a node. Several examples help clarify this distinction. In a MoE structure a single expert is typically a basic operator because it lacks an independent feedback loop and does not exist as a version that can be independently deployed or rolled back. In contrast the gating logic and expert combination methods of an MoE layer possess a clear decision space and independent evaluation metrics such as load balancing or degree of specialization making them much closer to the definition of an agent. A loss function itself is part of the evaluation criteria rather than a

 doi:10.20944/preprints202603.2003.v1

decision unit with closed-loop capabilities. However one could design a high-level Loss Design or Objective Configuration Agent responsible for reading data and logs proposing new loss combinations and verifying them through experiments while a specific loss term remains merely a tool for the agent to select.

When the communication complexity between two modules significantly exceeds their respective internal decision-making complexity, they should be merged into a single agent. In practical systems another scenario frequently arises where two logically distinct modules are always invoked as a pair and share nearly identical inputs or memory. These modules require the transmission of massive amounts of information between them during every call while the internal decision-making complexity of either module viewed in isolation is significantly lower than the communication complexity between them. Under these circumstances it is more reasonable from an agentic perspective to merge these two modules into a single agent. This allows a unified agent to manage internal states and decisions thereby reducing unnecessary inter-agent communication and avoiding the introduction of excessive pseudo-agents at the system level.

In summary while theoretically any component with an adjustable decision space and the ability to learn from feedback such as Mixture of Experts routing policies or loss design strategies can be encapsulated as an agent in practice a module is only worth promoting to an explicit agent when it forms a relatively independent decision making closed loop and satisfies the requirements for independent evaluation and secure evolution within the system architecture.

4. Principle 2: Evolutionary Methods for Agents

In Principle 1, we established a generalized definition for agents within recommender systems and noted that the scope of model-centric agents is relatively broad. These range from specific retrieval agents, ranking agents, and re-ranking or fusion agents to separable internal substructures of large-scale models such as MoE routing modules. Any component satisfying the criteria of a functional closed loop, independent evaluability, and room for self evolution can be regarded as a model-centric agent. Consequently, these agents are responsible not only for executing specific predictive tasks but also for autonomously improving their architectures, hyperparameters, and training strategies to enhance performance metrics relevant to their duties without relying entirely on manual tuning. This transition raises a fundamental question: by what specific means does a model-centric agent achieve self evolution? Aside from manual parameter adjustment, what systemic methods exist for such evolution? In this section, we categorize the primary methods driving the evolution of model-centric agents into two major types: **reinforcement learning based evolution** and **large language model based evolution.**

4.1. RL-Based Evolution

In many practical scenarios, the modifications surrounding a model-centric agent can be abstracted into a set of structured but low-dimensional decision variables. These include structural selections from a finite set of alternatives such as various backbones, activation functions, normalization methods, and pooling schemes. They also encompass a small number of key hyperparameters including learning rates, regularization strengths, loss function weights, and negative sampling ratios, as well as simple combinatorial strategies for selecting or weighting substructures and controlling the activation of specific branches. Within these contexts, the process of configuring or updating a model-centric agent can be effectively modeled as a reinforcement learning problem:

- **State**. Reflecting information such as current data distributions task objectives resource constraints and the historical performance of the agent.
- **Action**. Making specific choices regarding the structure or training configuration of the agent such as the aforementioned architectural selections and hyperparameter settings.
- **Reward**. Measuring the quality of actions using offline metrics such as AUC NDCG recall and diversity or through the results of small scale online experiments.

 doi:10.20944/preprints202603.2003.v1

The RL-based approach is particularly suitable as the primary evolutionary method for a model-centric agent when specific conditions are met. **First**, the action space must be precisely defined and low-dimensional, meaning the number of adjustable variables is limited or can be discretized into a small set of options where each dimension carries a clear meaning suitable for reinforcement learning actions. **Second**, the effect of each action must be independently measurable, allowing for the completion of a training and evaluation cycle for a given structure or hyperparameter configuration at a reasonable cost, while maintaining a clear causal relationship between the resulting reward signal and that configuration. Under these conditions, the advantage of RL-based updates over traditional hyperparameter search lies in the ability to continuously update strategies despite noisy or delayed returns, utilizing historical experience to reduce ineffective trials and gradually approaching an optimal configuration. Therefore, for model-centric agents with small-scale, simple, and easily parameterized action spaces, we recommend RL-based or other classical search methods as the primary means of evolution.

4.2. LLM-Based Evolution

In another category of scenarios the evolution of a model centric agent is no longer about fine tuning a few structural options or hyperparameters but instead requires significant and non local changes to model architectures training paradigms and task decomposition methods. These include redesigning sub network structures such as how to combine sequential modules graph modules Mixture of Experts specific towers or multi head experts. They also involve reorganizing multi task structures and loss systems including determining which objectives to learn together which parameters to share or separate and how to construct distillation or contrastive losses. Furthermore this evolution entails synthesizing business documentation historical experiments and error analysis to summarize problem patterns and propose new modeling concepts. This type of decision space typically possesses the following characteristics. **(i)** The action space is high dimensional and highly structured making it difficult to precisely enumerate or encode as a finite set of actions. Many decisions involve architectural and training paradigm designs that cannot be fully expressed through a small number of scalars or enumerated options. If forcibly discretized the search space would expand exponentially making it nearly impossible for reinforcement learning or traditional search methods to explore effectively. **(ii)** There exists a vast amount of informal human experience available for utilization. Engineering insights research paper conclusions internal design documents and historical experimental logs contain a wealth of knowledge regarding which structures are most effective in specific scenarios. This knowledge primarily exists in the form of natural language and code rather than a predefined action space.

In such cases, an LLM-based evolutionary approach is more appropriate. The general workflow involves using a large language model to comprehend and summarize the current agent structure, training configurations, performance bottlenecks, and business documentation. Based on this understanding, the LLM generates candidates for new architectural designs, optimization strategies, and training schemes. These candidates are then converted into concrete implementations through code or configuration generation, entering the standard training and evaluation pipeline. According to the evaluation feedback, superior solutions are retained, and the results are written back into an empirical memory to serve as a reference for the next generation cycle. The value of the LLM-based method lies in its ability to propose modifications within an open-ended and difficult-to-formalize design space by leveraging human experience and semantic information. This allows for the introduction of new structural types, combination methods, or training paradigms that traditional AutoML and pure reinforcement learning often struggle to invent automatically.

4.3. The Relationship Between RL-Based and LLM-Based Evolution: Independence, Alternation, and Parallelism

It is essential to highlight that RL-based and LLM-based evolution are not necessarily two sequential stages, nor are they a simple binary choice. In practical systems, three typical relationships exist between them.

 doi:10.20944/preprints202603.2003.v1

Independent Selection for a Single Agent (Single Dominance). If the adjustable space of a model-centric agent is already well-parameterized into finite actions (such as the architecture or hyperparameter configuration of a retrieval model), RL-based methods or traditional search can serve as the primary driver without the need for an LLM. Conversely, if the main challenge for another agent lies in structural innovation and task decomposition (such as restructuring a complex multi-tower model), it can be driven entirely by an LLM-based approach.

Alternating Use in Long-term Evolution (Phased Collaboration). For example, an LLM-based method can first propose a set of new structures and training paradigms to filter out several candidate architectures. Subsequently, within each candidate structure, RL-based methods can be employed to fine-tune key hyperparameters and local strategies. Alternatively, RL-based methods can first fully explore the local optimum of an existing structure; once the potential for improvement saturates, an LLM-based approach can generate new structural hypotheses to break out of that local optimum.

Different model-centric agents within a system can adopt different evolutionary paths. For instance, RL-based evolution can be applied to several agents with fixed structures that primarily require parameter tuning, while LLM-based evolution is reserved for a few critical agents responsible for architectural design or task decomposition. Within a single agent, certain design aspects can be delegated to an LLM (such as generating candidate structural templates), while other details are handled by RL (such as selecting specific configurations within those templates), creating a "LLM proposes, RL searches" combination.

From a high-level perspective, these two methodologies represent different facets of the same objective: RL-based methods excel at efficient exploration and convergence within known, parameterizable spaces, whereas LLM-based methods are superior at proposing new hypotheses and major revisions within open, structured spaces that rely heavily on empirical knowledge. A mature AgenticRS typically requires both capabilities, combining them flexibly based on each agent's specific responsibilities and decision space.

5. Principle 3: Types of Agent Evolution

Building upon the evolutionary methodologies discussed in the previous principle, a further question arises: when a system contains multiple model-centric agents, at what granularity does this evolution occur? Does it focus solely on optimizing each individual agent, or does it also encompass the optimization of which agents exist and how they connect and collaborate? To address this, we distinguish between two complementary granularities of evolution for model-centric agents:

- **Individual Evolution**: This involves fixing the connectivity between agents and updating only the internal structure, parameters, and training strategies of a single model-centric agent.
- **Compositional Evolution**: Given the existence of multiple model-centric agents, this focuses on optimizing the overall architecture—determining which agents participate, how they are interconnected, and how labor is divided or routed between them.

These two granularities are not mutually exclusive; rather, they appear in an alternating and cyclical fashion within practical systems.

5.1. Individual Evolution

Individual evolution focuses on how a specific model-centric agent can improve itself while keeping the rest of the system relatively stable. Given the agent's input-output interfaces and its defined responsibilities within the system, we allow for changes in its internal network structure (such as layers, sub-tower architectures, feature interaction methods, or normalization schemes), its training configurations (including loss function combinations, optimizers, learning rates, and negative sampling strategies), and its feature selection and processing methods.

At this granularity, the "graph structure" between agents is considered fixed. The calling relationships between modules like retrieval, ranking, and re-ranking remain unchanged, and within a single large-scale model, the connections between sub-towers or MoE layers are treated as temporarily constant. The typical questions addressed by individual evolution include: (1) How can the agent's

 doi:10.20944/preprints202603.2003.v1

architecture be redesigned or fine-tuned within its existing scope of responsibility? (2) How should the training workflow and hyperparameters be adjusted given the fixed data and features? (3) How can version replacement and rollback be performed for this agent without disrupting the stability of the overall pipeline?

At this level, the RL-based and LLM-based methods discussed in Principle 2 can be directly applied. For agents with relatively simple structural spaces—such as a specific retrieval model's backbone or key hyperparameters—RL-based or traditional search methods are ideal for automated tuning. For agents with complex structural spaces that require significant domain expertise, such as the design of a sophisticated multi-tower sub-structure, LLMs are better suited to generate candidate architectures and training schemes, which are then screened through standard training and evaluation.

Ultimately, individual evolution emphasizes that each agent should polish its specific area of responsibility, maximizing local performance within the established system framework.

5.2. Compositional Evolution

Individual evolution assumes that the connections between agents remain largely stable; however, in many critical scenarios, this assumption does not hold. As user distributions, content ecosystems, and business objectives shift, the system must address a different set of questions: given a multitude of model-centric agents, which ones should participate in decision-making, how should they be interconnected, and how should their labor and routing evolve over time? This is what we define as compositional evolution. The "multiple agents" involved can exist at different levels. At the system level, this involves the combination of independent model-centric agents, such as how multiple retrieval models coexist, how a primary ranking model coordinates with auxiliary towers, or how re-ranking and risk models are integrated into the decision pipeline. At the intra-model level, it concerns the composition of sub-agents within a single large-scale model—for instance, determining which towers in a multi-tower structure participate in predicting specific samples and how they fuse, or deciding which expert agents exist in a MoE structure and how routing policies distribute traffic among them. It also includes activating specific sub-agent paths for different targets or scenarios, such as a dedicated sub-pipeline for cold-start users. At this granularity, the object of evolution is no longer the internal parameters of a single agent, but the graph structure formed between agents. Typical questions include:

1. **Selection of Agent Sets:** Which model-centric agents should participate in decision-making for a given scenario? Are there redundant agents that should be decommissioned, or is there a need to introduce new agents into the pipeline?
2. **Connectivity and Fusion Methods:** How should outputs from different agents be combined? Examples include weighted fusion, cascaded filtering, or multi-stage pruning. Within a single model, how should outputs from multiple towers or experts be aligned in the representation space and aggregated effectively?
3. **Routing and Labor Division Rules:** Which path of agent combinations should specific samples follow (e.g., partitioned by user demographics, scenarios, or request types)? Should certain agents be activated only under specific conditions (e.g., a risk model that intervenes only during high-risk requests)?

Similar to individual evolution, the compositional evolution layer can also leverage the two categories of methods outlined in Principle 2. In RL-based compositional evolution, given a set of agents, the process of selecting combinations and routing rules is treated as a policy search problem, where feedback signals are used to optimize traffic distribution and structural choices. In LLM-based compositional evolution, the system synthesizes logs, documentation, and business descriptions to understand which capabilities need to work together in specific scenarios. The LLM then generates new structural or routing schemes—such as introducing new pipelines, adjusting connections between towers, or designing dedicated paths for specific user segments—before evaluating their effectiveness through experiments. Consequently, compositional evolution does not replace individual evolution;

 doi:10.20944/preprints202603.2003.v1

rather, it operates at a higher level to answer the fundamental question of how the system should best utilize these agents.

5.3. Synergy Between Individual and Compositional Evolution

In a practical AgenticRS, individual and compositional evolution typically drive system growth through an alternating and cyclical process rather than being mutually exclusive. A typical collaborative pattern involves the following stages: (1) Individual Evolution within a Fixed Composition Initially, the connections between agents are held constant while each key model-centric agent undergoes local optimization. The goal during this phase is to maximize the specific capabilities of each agent within its current scope of responsibility and operational context. (2) Triggering Compositional Evolution in Response to Change When environmental factors or business objectives shift—such as changes in user distribution, content trends, or target weights—the system initiates a round of compositional evolution. This phase involves adjusting the set of active agents, their interconnection methods, and routing rules. (3) Refining Optimization Post-Reconfiguration. Once the new compositional structure stabilizes, the focus returns to individual evolution. Under this new architecture, each agent faces a fresh set of local optimization challenges, and its performance can be further polished to fit its updated role in the system.

This alternating cycle ensures the system can continuously refine model capabilities at a local level while periodically restructuring how those capabilities are organized at a global level. This dual-layered approach allows the system to remain highly adaptive to environmental changes and evolving business goals over long timescales.

6. Principle 4: The Reward Structure of Agents

In the aforementioned principles, we perceive a model as a structure composed of multiple model-centric agents, spanning from system-level agents like retrieval and ranking to internal sub-agents such as sub-towers and MoE routing modules. When updating a specific model-centric agent, a critical question arises: what type of optimization signal should primarily guide its evolution? Should it focus solely on its own local metrics, or should it align directly with outer-layer or even business-level objectives? To address this, we distinguish between two types of reward signals within the model: **Inner Rewards**, which are local objectives derived from the agent's own perspective or its direct sub-tasks, and **Outer Rewards**, which are objective signals determined by outer-layer agents or overarching business logic and passed inward. These are not static system-level definitions; rather, they represent optimization signals from different hierarchies relative to the specific model-centric agent being observed.

6.1. Inner Reward: Local Objectives Oriented Toward the Agent

For a specific model-centric Agent **A**, which could represent an entire model, a single layer, a sub-tower, or a routing module—the Inner Reward refers to evaluation signals directly aligned with its primary duties. Examples include:

- If Agent **A** is a retrieval model, the Inner Reward might consist of recall rates, coverage, or effectiveness in long-tail retrieval.
- If Agent **A** is a CTR prediction sub-tower within a ranking model, the Inner Reward could be the sub-tower's own prediction loss, AUC, or NDCG.
- If Agent **A** is an MoE routing agent, the Inner Reward would focus on load balancing, expert utilization, and routing stability.
- If Agent **A** is a risk identification tower, the Inner Reward would involve accuracy, miss rates, and false-positive rates.

From an internal perspective, Inner Rewards typically exhibit two distinct characteristics. **First**, they represent proximal signals, which are directly determined by the agent's outputs or intermediate results; these signals can be frequently calculated during the training phase, providing dense feedback. **Second**, they offer stable training and clear optimization directions, as they can typically be optimized

 doi:10.20944/preprints202603.2003.v1

efficiently under supervised learning or standard optimization frameworks and are highly correlated with the local capabilities for which the agent is responsible. Consequently, the Inner Reward describes the direction in which an agent should improve within its own specific scope of duty.

6.2. Outer Reward: Target Signals from the Outer Layers

For the same Agent **A**, the Outer Reward represents signals back-propagated from outer-layer modules or the overall system objectives. Examples include:

- For a ranking sub-tower Agent, the Outer Reward could be the aggregate loss of the entire ranking model relative to the final click/conversion goal, or higher-level business proxy metrics.
- For a retrieval Agent, the Outer Reward might be the final business impact contributed by this specific retrieval path, such as the total clicks, conversions, or retention it generates—transmitted back through evaluation and credit assignment mechanisms.
- For a risk tower Agent, the Outer Reward could be the overall system risk event rate or its corresponding cost function, reminding the agent to not only optimize its own identification metrics but also account for its impact on the entire chain.

From an internal perspective, Outer Rewards typically possess the following characteristics: Outer Rewards are characterized first as distal and indirect signals. These are jointly determined by higher-level model architectures or business logic, meaning they cannot be fully controlled by the individual agent alone; consequently, they tend to be noisier, sparser, and subject to longer feedback latencies. Secondly, they primarily express the need for synergy. They require the agent to consider its coordination with upper-layer agents or the entire model while optimizing its own tasks, acting as an aligner that pulls local behavior toward global objectives. Ultimately, viewed from within the model, the Outer Reward describes how an agent must cooperate with its superiors or the system as a whole, beyond simply hitting its own functional benchmarks.

6.3. Most Internal Agents Should Primarily Follow Inner Rewards

In a complex model containing a vast array of model-centric agents (such as different towers, layers, and experts), it is theoretically possible for each to receive Outer Rewards from the broader system. However, we do not want every sub-agent to rely primarily on these Outer Rewards for their updates, for several critical reasons:

1. **Outer signals are too ambiguous for internal agents:** Many high-level metrics (such as final click loss or business proxies) are the aggregate result of multiple modules working in concert. For a single internal agent, these signals are often difficult to decouple; using them as the primary optimization target makes it nearly impossible for the agent to discern its specific direction for evolution.
2. **Inner signals are more stable and easier to train**: Local metrics—such as prediction loss, retrieval quality, or risk identification accuracy—are highly correlated with the agent's specific responsibilities. These can be efficiently trained under standard supervised learning or local optimization frameworks, offering much more controllable convergence properties.
3. **Architectural design already encodes the evolutionary path**: Through the model structure and the defined connections between agents, we have already pre-assigned responsibility. Within such a framework, as long as the majority of sub-agents improve toward their respective Inner Rewards, the system as a whole tends to evolve in a rational and productive direction.

Therefore, from the perspective of engineering practice and optimization stability, we advocate for the following principle. Within a single model, the vast majority of learned parameters and sub-agents should update primarily along the direction of their own Inner Rewards. Outer Rewards should only be used as a lightweight correction or regularization when necessary to prevent severe deviations from the overall objective. This approach avoids the training instability that occurs when all internal modules attempt to align directly with volatile business metrics, while simultaneously fully leveraging the modular advantages inherent in the system's structural design.

 doi:10.20944/preprints202603.2003.v1

6.4. A Small Number of Key Agents Require Stronger Reliance on Outer Rewards

While most internal agents can rely primarily on their Inner Rewards, a small number of agents in critical positions must possess a stronger perception of Outer Rewards to fulfill their coordination and alignment responsibilities. Typical examples include: (1) Decision-Making Agents near the Output Layer: Examples include the "fusion Agent" responsible for combining multiple sub-tower outputs, or modules making the final re-ranking decisions at the end of the sorting pipeline. Since these agents directly determine the system's output, they must be tightly aligned with the final business objectives. (2) Agents in Routing or Resource Allocation Roles: This includes gating Agents in MoE structures or routing modules that distribute traffic across different retrieval paths or sub-towers. Their decisions determine which sub-agents are activated and which samples they process; therefore, they must use Outer Rewards to sense the overall impact. (3) Key Agents Handling Business-Level Constraints: These are sub-networks that directly carry the burden of risk control, compliance constraints, or other safety objectives. In these scenarios, business-level Outer Rewards (such as the overall risk event rate) are dominant signals that cannot be ignored.

In these strategic positions, prioritizing the Outer Reward serves a twofold purpose: first, it ensures that coordination and allocation tasks are more tightly aligned with higher-level objectives, acting as a vital bridge between local capabilities and global goals. Secondly, it minimizes the potential discrepancy between a state where every sub-agent is locally optimized for its inner reward and the state of true system-wide global optimality.

7. Future Direction

Moving forward, we will leverage these foundational principles to transition traditional, static recommendation pipelines into fully realized AgenticRS. Our roadmap focuses on transforming modular components into autonomous agents that can navigate complex multi objective environments with minimal manual intervention.

To support this transition, we plan to release a series of studies addressing the core technical challenges of AgenticRS, such as:

- **Advanced self evolution**: Developing hybrid optimization frameworks that combine the rapid, data-driven learning of RL with the high-level reasoning and architectural adaptability of LLMs.
- **Inter-Agent Communication**: Establishing robust protocols for information exchange and credit assignment, ensuring that decentralized agents can coordinate effectively without systemic instability.
- **Scalable Orchestration**: Researching efficient ways to manage the compositional evolution of hundreds of agents within live, high-concurrency production environments.

By tackling these hurdles, we aim to provide the community with both the theoretical depth and the practical tools necessary to build recommendation systems that are not just models, but living, evolving entities.

8. Conclusion

This paper establishes a conceptual framework and design principles for the Agentic Recommender System (AgenticRS), moving beyond static, manually iterated architectures toward a self-evolving ecosystem. By defining model-centric Agents through the criteria of functional closure, independent evaluability, and evolutionary potential, we provide a structured approach to decentralizing model intelligence. Our framework systematically addresses agent evolution through two dimensions:

- Methodological: Integrating RL-based and LLM-based mechanisms for autonomous optimization.
- Structural: Balancing individual evolution of single agents with the compositional evolution of multi-agent networks.

Central to this system is a hierarchical reward structure. We advocate that while most sub-agents should focus on Inner Rewards to maintain training stability and local proficiency, critical coordination

 doi:10.20944/preprints202603.2003.v1

agents must prioritize Outer Rewards to ensure global alignment with business objectives. Ultimately, this work shifts the focus from building monolithic models to designing adaptive environments where agents interact, compete, and cooperate. These principles offer a clear roadmap for developing sustainable, industrial-scale recommendation systems capable of autonomous adaptation in complex, dynamic environments.

References

1. Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.
2. Hao Deng, Haibo Xing, Kanefumi Matsuyama, Moyu Zhang, Jinxin Hu, Hong Wen, Yu Zhang, Xiaoyi Zeng, and Jing Zhang. Csmf: Cascaded selective mask fine-tuning for multi-objective embedding-based retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2122–2131, 2025.
3. Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.
4. Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1725–1731, 2017.
5. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.
6. Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, and Julian McAuley. Generating long semantic ids in parallel for recommendation. *arXiv preprint arXiv:2506.05781*, 2025.
7. Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *Proceedings of the 2018 IEEE International Conference on Data Mining*, pages 197–206. IEEE, 2018.
8. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
9. Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11523–11532, 2022.
10. Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1754–1763. ACM, 2018.
11. Juexin Lin, Sachin Yadav, Feng Liu, Nicholas Rossi, Praveen R Suram, Satya Chembolu, Prijith Chandran, Hrushikesh Mohapatra, Tony Lee, Alessandro Magnani, et al. Enhancing relevance of embedding-based retrieval at walmart. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 4694–4701, 2024.
12. Lingyu Mu, Hao Deng, Haibo Xing, Jinxin Hu, Yu Zhang, Xiaoyi Zeng, and Jing Zhang. Masked diffusion generative recommendation, 2026.
13. Lingyu Mu, Hao Deng, Haibo Xing, Kaican Lin, Zhitong Zhu, Yu Zhang, Xiaoyi Zeng, Zhengxiao Liu, Zheng Lin, and Jinxin Hu. Synergistic integration and discrepancy resolution of contextualized knowledge for personalized recommendation. *arXiv preprint arXiv:2510.14257*, 2025.
14. Lingyu Mu, Zhengxiao Liu, Zhitong Zhu, and Zheng Lin. Trust-grs: A trustworthy training framework for graph neural network based recommender systems against shilling attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 12408–12416, 2025.
15. Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.
16. Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.
17. Noah Shinn, Federico Cassano, Edward Labash, Ben Mildenhall, and Iddo Drori. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

 doi:10.20944/preprints202603.2003.v1

18. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
19. Hanbing Wang, Xiaorui Liu, Wenqi Fan, Xiangyu Zhao, Venkataramana Kini, Devendra Yadav, Fei Wang, Zhen Wen, Jiliang Tang, and Hui Liu. Rethinking large language model architectures for sequential recommendations. *arXiv preprint arXiv:2402.09543*, 2024.
20. Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)*, 54(7):1–38, 2021.
21. Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Learnable item tokenization for generative recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 2400–2409, 2024.
22. Wenjie Wang, Xinyu Lin, Fuli Feng, Xiangnan He, and Tat-Seng Chua. Generative recommendation: Towards next-generation recommender paradigm. *arXiv preprint arXiv:2304.03516*, 2023.
23. Xu Wang, Jiangxia Cao, Zhiyi Fu, Kun Gai, and Guorui Zhou. Home: Hierarchy of multi-gate experts for multi-task learning at kuaishou. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 2638–2647, 2025.
24. Haibo Xing, Hao Deng, Yucheng Mao, Jinxin Hu, Yi Xu, Hao Zhang, Jiahao Wang, Shizhun Wang, Yu Zhang, Xiaoyi Zeng, et al. Reg4rec: Reasoning-enhanced generative model for large-scale recommendation systems. *arXiv preprint arXiv:2508.15308*, 2025.
25. Yuhao Yang, Zhi Ji, Zhaopeng Li, Yi Li, Zhonglin Mo, Yue Ding, Kai Chen, Zijian Zhang, Jie Li, Shuanglong Li, et al. Sparse meets dense: Unified generative recommendations with cascaded sparse-dense representations. *arXiv preprint arXiv:2503.02453*, 2025.
26. Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Tom Griffiths, Graham Neubig, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
27. Guorui Zhou, Jiaxin Deng, Jinghao Zhang, Kuo Cai, Lejian Ren, Qiang Luo, Qianqian Wang, Qigen Hu, Rui Huang, Shiyao Wang, et al. Onerec technical report. *arXiv preprint arXiv:2506.13695*, 2025.
28. Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1059–1068. ACM, 2018.