AGENT-NASCD: EFFICIENT AND RELIABLE AGENTIC ARCHITECTURE SEARCH FOR COGNITIVE DIAGNOSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Cognitive diagnosis (CD) aims to estimate students' proficiency in knowledge concepts from response logs. However, existing evolutionary and one-shot neural architecture search (NAS) methods for designing CD interaction functions still incur substantial candidate evaluation costs. Besides, directly prompting a language model can produce invalid programs or unstable search directions. To address these issues, this paper proposes Agent-NASCD, a single-agent framework that treats CD architecture search as sequential evidence-conditioned decision making. Specifically, CD-oriented tools support program construction and mutation, a model optimization graph (MOG) records candidate lineage and evaluation results, and constrained proxy/full evaluation screens candidates at different training fidelities. Experiments on three real-world datasets show competitive predictive performance of the selected architectures and support the contributions of the MOG and diagnostic feedback to candidate-training efficiency. The source code for Agent-NASCD is provided in the supplementary material.

1 INTRODUCTION

Cognitive diagnosis (CD) is a student assessment task in intelligent education (Wang et al., 2020; Yang et al., 2025a). Given students' historical exercise records and the exercise–concept relational matrix (Q-matrix), a CD model infers each student's proficiency in knowledge concepts. These diagnosis results support personalized exercise recommendation and targeted learning activities.

Existing CD approaches can be broadly grouped into two types. The first type relies on handcrafted psychometric functions, such as IRT (Lord, 1980), MIRT (Reckase, 2009), and DINA (de la Torre, 2009). The second type uses neural networks to model more flexible student–exercise interactions, including NCDM (Wang et al., 2020), KaNCD (Wang et al., 2023), KSCD (Ma et al., 2022), and graph-based models (Yang et al., 2024b; Qian et al., 2024). These models improve predictive quality and thus better support subsequent adaptive tasks. However, their interaction architectures are still designed by experts. Besides, manual architecture design requires considerable trial-and-error effort.

Neural architecture search (NAS) has therefore been introduced to design CD models automatically. EMO-NAS-CD searches operator programs by evolutionary multi-objective optimization (Yang et al., 2025a); OSCD further studies one-shot evaluation and robustness (Wang et al., 2026); and recent work incorporates language models into evolutionary NAS (Yang et al., 2026). These methods formulate diagnostic function design as an architecture search problem. However, candidate-wise search still trains many architectures independently; reusing earlier evaluation evidence to guide subsequent actions remains important, as illustrated in Figure 1(a). A ReAct-style controller can still generate illegal programs, choose search directions without supporting evaluations, or repeatedly explore the same structural family, as shown in Figure 1(b).

To this end, this paper proposes Agent-NASCD, a single-agent NAS framework in which an LLM controller π_ϕ adaptively decides what to observe, where to explore, which candidate to evaluate, at what fidelity, and when to stop. Specifically, Agent-NASCD consists of three core designs (Figure 2): CD search tools, a model optimization graph, and constrained proxy/full evaluation. These designs identify illegal or duplicate candidates and provide evidence for π_ϕ to choose further evaluation of the same program or exploration from another parent.

The main contributions of this paper are summarized as follows:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

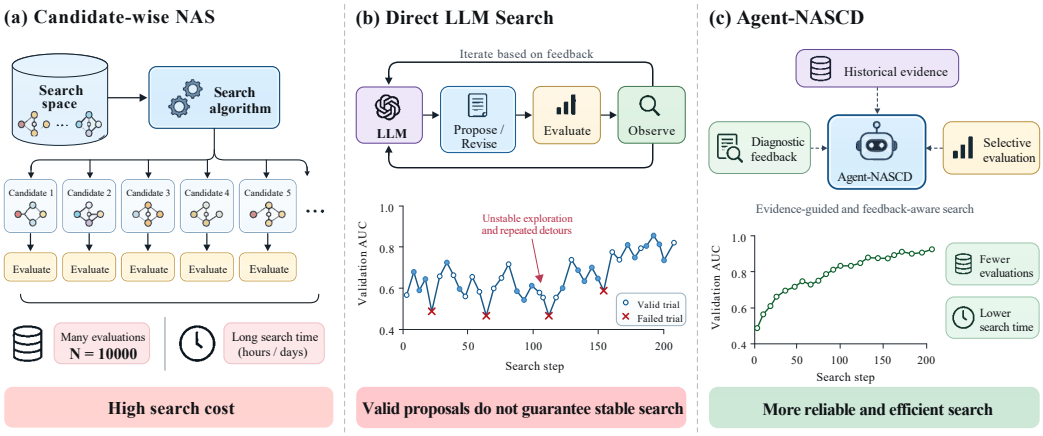

Figure 1: CD architecture search paradigms. (a) Candidate-wise NAS trains and evaluates many architectures independently. (b) Direct LLM search uses feedback but can remain unstable. (c) Agent-NASCD uses historical evidence and diagnostic feedback for selective evaluation.

- We formulate CD architecture search as sequential evidence-conditioned decision making, allowing one LLM controller to adapt observation, exploration, evaluation, and stopping decisions to the accumulated search evidence.
- We design three complementary components for this formulation: CD-oriented construction and mutation tools, a model optimization graph with structural summaries, and constrained multi-fidelity evaluation with diagnostic feedback.
- We conduct experiments on three real-world datasets. The results show best or near-best predictive performance and support the contributions of graph observations and diagnostic feedback to candidate-training efficiency.

2 Related Work

2.1 Cognitive Diagnosis

Existing CD approaches can be broadly classified into psychometric models (Lord, 1980; Reckase, 2009; de la Torre, 2009), neural interaction models (Wang et al., 2020; 2023; Ma et al., 2022), and graph- or structure-aware variants (Yang et al., 2024b; Qian et al., 2024; Yang et al., 2025b; Li et al., 2025). For psychometric models, IRT (Lord, 1980) and MIRT (Reckase, 2009) use logistic functions over latent abilities, while DINA (de la Torre, 2009) uses a conjunctive slipping–guessing form. For neural models, NCDM (Wang et al., 2020) and KSCD (Ma et al., 2022) use student, exercise, and concept embeddings as inputs to a neural diagnostic function, and KaNCD (Wang et al., 2023) further models implicit concept relations. For graph- or structure-aware models, DisenGCD (Yang et al., 2024b) and ORCDF (Qian et al., 2024) learn representations on student–exercise–concept graphs, while KAN2CD (Yang et al., 2025b) and GIRT (Li et al., 2025) modify the interaction function itself. Despite this progress, their interaction architectures remain handcrafted.

2.2 Neural Architecture Search

NAS has been used to design network architectures automatically (Yang et al., 2021). In intelligent education, ENAS-KT searches Transformer architectures and input features for knowledge tracing (Yang et al., 2023). EMO-NAS-CD searches operator programs for CD by evolutionary multi-objective optimization (Yang et al., 2025a). SymbolicCDM jointly optimizes symbolic trees and continuous parameters (Shen et al., 2024). Besides, multi-fidelity evaluation and one-shot search further reduce the cost of scoring candidates (Yang et al., 2021; Wang et al., 2026). This paper uses candidate lineage and previous results to guide construction and evaluation throughout search.

2.3 Language Models as Search Controllers

ReAct interleaves reasoning with actions and observations so that a language model can call tools and read the returned evidence (Yao et al., 2023). Optimization by prompting uses previously evaluated solutions to guide later proposals (Yang et al., 2024a). NADER studies multi-agent architecture design (Yang et al., 2025c). AFlow searches executable agentic workflows (Zhang et al., 2025). LLM-assisted evolutionary NAS incorporates a language model into CD architecture evolution (Yang et al., 2026). These studies suggest that a language model can participate throughout optimization rather than only generate an initial design. However, existing controllers lack CD-specific memory of candidate lineage and evaluation fidelity, or remain inside an evolutionary outer loop. Agent-NASCD coordinates these decisions through CD tools and a lineage-aware graph.

3 Problem Formulation

Formally, let $\mathcal{S}$, $\mathcal{E}$, and $\mathcal{K}$ denote the sets of students, exercises, and knowledge concepts. The response log is $\mathcal{D} = \{(s_i, e_i, y_i)\}_{i=1}^{N}$, where $y_i \in \{0, 1\}$ indicates whether student s_i answers exercise e_i correctly. A binary Q-matrix records the concepts required by each exercise, where q_e denotes the row corresponding to exercise e. Based on the response logs and the Q-matrix, an architecture $a \in \mathcal{A}$ defines a response function $f_a(s, e, q_e; \theta)$, where $\mathcal{A}$ is the space of admissible operator programs. The model parameters are learned by minimizing the binary cross-entropy loss:

$$\theta_a^\star = \arg\min_{\theta} \sum_{(s,e,y)\in\mathcal{D}_{\mathrm{tr}}} \left[-y \log f_a(s, e, q_e; \theta) - (1 - y) \log(1 - f_a(s, e, q_e; \theta))\right]. \tag{1}$$

The search selects the highest-utility completed architecture under a budget-triggered stopping rule,

$$a^\star = \arg\max_{a\in V_\tau^{\mathrm{eval}}} U(a; \mathcal{D}_{\mathrm{score}}), \qquad R_t \prec B \quad (t < \tau), \tag{2}$$

Here U is validation AUC, τ is the final search turn, and V_τ^{eval} contains the Top-K candidates successfully confirmed at Full fidelity when confirmation is enabled, or all successfully evaluated candidates otherwise. $R_t \prec B$ compares usage at turn boundaries with nominal limits componentwise; in-flight work and final confirmation may exceed these thresholds. Evaluation uses proxy or full fidelity on $\mathcal{D}_{\mathrm{score}} = \mathcal{D}_{\mathrm{val}}$; test responses are reserved for final assessment.

At decision step t, π_ϕ receives conversation history H_t, an observation $O(G_t)$ extracted from the current MOG G_t, and the remaining allowances $b_t = (B - R_t)_+$. One turn produces one controller response with an ordered tool-call batch $u_t = (u_{t,1}, \ldots, u_{t,m_t})$, executed sequentially:

$$u_t \sim \pi_\phi(\cdot \mid H_t, O(G_t), b_t), \qquad (o_{t+1}, G_{t+1}) = \mathrm{ExecuteBatch}(u_t, G_t). \tag{3}$$

The observation o_{t+1} collects the executed tools' records, summaries, evaluation results, and diagnostics in order. The controller adapts through context and search state without fine-tuning π_ϕ.

4 The Proposed Method

4.1 Overall Framework

Figure 2 summarizes five operations in Agent-NASCD, whose tool calls are selected by the controller rather than executed in a fixed sequence. First, randomly generated root architectures and their evaluation results form the initial graph G_0. During search, π_ϕ can retrieve earlier outcomes from $O(G_t)$. For candidate construction, π_ϕ chooses a parent to refine or an alternative family to explore. For candidate evaluation, π_ϕ decides whether an architecture a should be trained at proxy or full fidelity. Construction, mutation, and evaluation update G_t after the corresponding requests. Search continues until termination is accepted or a checked resource limit is reached; an optional full-confirmation stage reevaluates the top-K candidates before final selection. The following subsections present the three core designs. Algorithm 1 in **Appendix** B gives the complete procedure.

4.2 CD-Oriented Search Tools

The search space consists of CD diagnostic functions that combine three input representations. Formally, for student s and exercise e, the learned embeddings are $h_s, h_e \in \mathbb{R}^d$, and $h_c = W_c q_e + b_c \in$

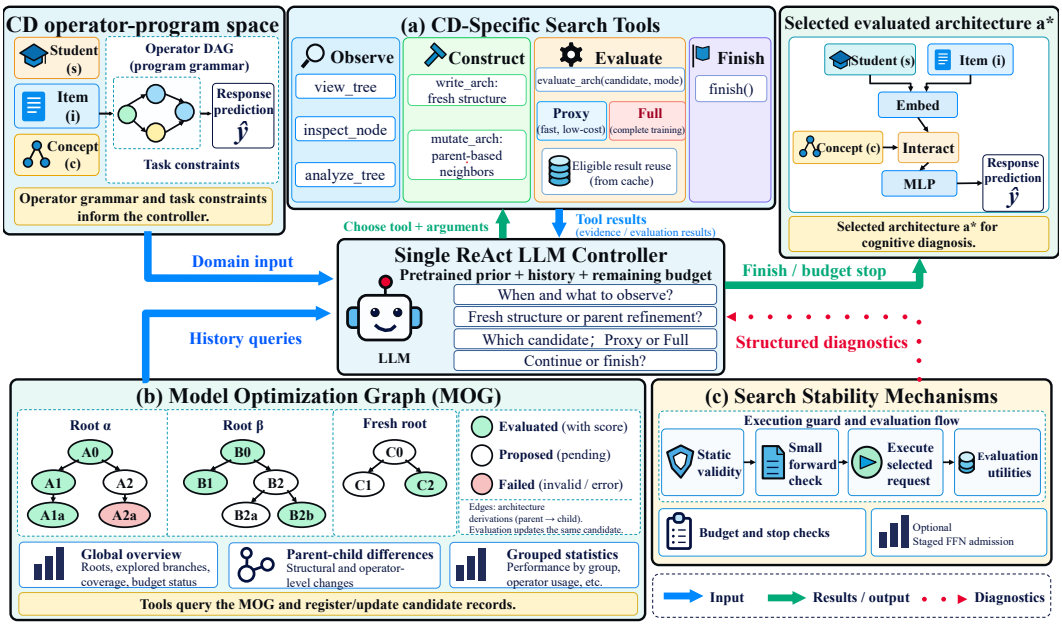

Figure 2: Overview of Agent-NASCD: one ReAct agent and three supporting components. The controller π_ϕ follows a Think–Act–Observe loop. Arrows connect the seven tools (a), the MOG G_t (b), and the controller. Panel (c) shows execution checks and diagnostic feedback.

$\mathbb{R}^d$ projects the exercise's concept indicators into the same hidden space. Following operator-based CD search (Yang et al., 2025a), we encode an architecture a as a sequence of integer triples,

$$\texttt{Dec}(a) = [(p_j, q_j, o_j)]_{j=1}^{m}, \qquad z_{j+2} = o_j(z_{p_j}, z_{q_j}), \tag{4}$$

where $z_0 = h_s$, $z_1 = h_e$, and $z_2 = h_c$. Each reference points to an input or a preceding node, while a unary operator uses only its designated operand. The 25 enabled operators cover elementwise transforms, arithmetic composition, scalar reductions, and CD-oriented interactions. Table 10 in **Appendix** C lists every operator, including arity, output dimension, learned parameters, and numerical guards. A vector terminal output is passed through a three-layer prediction head with nonnegative weights; a scalar output is directly clipped to the probability interval $[0, 1]$.

To operate on this representation, seven search tools are provided for π_ϕ. Specifically, `write_arch` creates a fresh operator program, while `mutate_arch` generates structural neighbors of a selected parent through programmatic edits. For observation, `view_tree` reads candidate records, `inspect_node` returns a candidate's structure and metrics, and `analyze_tree` summarizes evaluated structures by operator presence, output type, and program length. The `evaluate_arch` tool trains a at a requested fidelity, and `finish` requests termination. Creation alone does not launch training: writing or mutating only registers $\texttt{Dec}(a)$ in G_t.

4.3 Model Optimization Graph

The MOG maintains search history as evidence for subsequent action selection. Formally, $G_t = (V_t, E_t)$ contains architecture vertices and derivation edges (v_i, v_j). A written architecture a may form a root, whereas a mutated a retains its parent relation. Each vertex stores an identifier, the `Dec` encoding, derivation information, execution status, and current metrics. Parent–child comparisons relate structural differences to validation-score changes within each search branch. Keeping evaluated, pending, and failed nodes allows π_ϕ to compare siblings and to revisit a strong parent after an unsuccessful edit. A repeatedly refined parent marks an already explored structural neighborhood; an unevaluated sibling or new root permits further exploration.

To provide this history to the controller, `view_tree` returns scores and lineage, and `analyze_tree` groups evaluated candidates and reports score summaries. Each node stores its current metrics, with separate proxy and full records that distinguish additional training from a

change in architecture. These observations guide sibling evaluation and parent refinement while retaining alternative branches for further exploration.

4.4 Constrained Evaluation and Search Procedure

Before any training starts, the environment checks operator availability, reference legality, arity, structural bounds, and tensor execution. The returned diagnostics let the controller revise the program or choose another action. An illegal program is rejected with a diagnostic and is not trained; a training failure is recorded separately from a successful evaluation. The searchable FFN operator becomes available after a configured score threshold is reached, allowing early search to focus on simpler interactions while retaining the terminal prediction head. Canonical encodings identify duplicate architectures so that eligible completed evaluations can be reused. These checks enforce the execution requirements before any requested candidate training.

To allocate the training budget, the controller can request proxy or full evaluation. Specifically, the proxy recipe uses N_{fid}=55; full evaluation uses N_{fid}=100 with early stopping. Since a low proxy score may result from incomplete training, the controller can request full evaluation of the same candidate or explore from a new parent. When confirmation is enabled, the Top-K validation-ranked candidates are fully evaluated, and the candidate with the highest resulting validation AUC is selected from successful confirmations; otherwise, the highest-scoring completed candidate is returned. The total cost of candidate training is therefore defined as follows:

$$C_{\text{search}} = C_{\text{roots}} + C_{\text{loop}} + C_{\text{confirm}}, \tag{5}$$

The three terms account for root evaluation, in-loop candidate training, and final confirmation, with $C_{\text{confirm}} = 0$ when the last stage is not enabled. LLM waiting adds wall time outside C_{search}; B specifies loop stopping thresholds rather than hard bounds on final training cost.

5 Experiments

To evaluate the effectiveness of Agent-NASCD, we conduct experiments on three real-world datasets. We compare predictive performance and examine how the search components affect candidate-training cost. Table 3 compares DeepSeek-V4-Pro and Claude-Sonnet-5 as controllers; **Appendix** G specifies the shared conditions for these controller comparisons.

5.1 Experimental Settings

Datasets. We conducted experiments on three real-world datasets: ASSIST09 and the 2,000-student XES-2k and MOOC-2k subsets associated with DFCD preparation (Liu et al., 2025). Their statistics are summarized in Table 4. **Appendix** A provides dataset statistics, training configurations, seed counts, AUC statistics, and DOA-based diagnostic results.

Compared Approaches and Metrics. We compared the architectures selected by Agent-NASCD with psychometric models IRT (Lord, 1980), MIRT (Reckase, 2009), and DINA (de la Torre, 2009), neural CDMs NCDM (Wang et al., 2020), KaNCD (Wang et al., 2023), KSCD (Ma et al., 2022), and KAN2CD (Yang et al., 2025b), graph- or structure-aware models GIRT (Li et al., 2025), ORCDF (Qian et al., 2024), and SymbolicCDM (Shen et al., 2024), and the search methods EMO-NAS-CD (Yang et al., 2025a) and OSCD (Wang et al., 2026). All three NAS methods independently search each dataset–ratio–seed setting in the same CD operator-program space; EMO-A7 and OSCD-AA1 label the per-run outputs of EMO-NAS-CD and OSCD. Both baselines follow their original search procedures; their labels do not denote a fixed architecture reused across settings. The LLM-based search variants use ReAct under shared experimental settings, with individual components or all proposed search modules removed. Predictive performance is measured by AUC, ACC, and RMSE.

Parameter Settings. All methods use the same processed datasets and four training/validation/test ratios (40%/10%/50%, 50%/10%/40%, 60%/10%/30%, and 70%/10%/20%); the split seed follows the run seed. The controller π_ϕ is `glm-5.3-flash`; search starts from randomly generated roots and ranks architectures by U. All experiments use the same ten random seeds, giving 120 independent searches per NAS method across twelve dataset–ratio settings. All experiments use a single

NVIDIA GeForce RTX 3090 GPU and batch size 256; Agent-NASCD uses dimension 128, Adam (10^{-3}), 55/100 proxy/full epochs, and full-training patience 12; baselines follow their papers.

Evaluation Protocol. Both the main searches and component studies use validation feedback for architecture construction, candidate ranking, and final selection. The test set is held out throughout search and used only to evaluate the selected architecture. Table 1 reports the final test performance of the selected architectures, while the controlled study examines the contributions of individual search components. We report final predictive quality and search resources in separate comparisons (Li & Talwalkar, 2020; Lindauer & Hutter, 2020).

5.2 Comparison of the Selected Architectures

Table 1: Prediction performance on (a) ASSIST09, (b) XES-2k, and (c) MOOC-2k. ↑ (↓): higher (lower) is better. **Bold**: best; underline: runner-up, sharing displayed ties. –: missing; †: unranked numerical fallback. Scores are means of final test results over ten seeds under the protocol in Section 5.1.

(a) ASSIST09												
Model	**40%**			**50%**			**60%**			**70%**		
	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓
IRT	0.7003	0.7178	0.4572	0.6777	0.6896	0.4675	0.6849	0.7032	0.4606	0.6783	0.6896	0.4608
MIRT	0.5967	0.6084	0.6025	0.6120	0.6218	0.5923	0.6253	0.6382	0.5828	0.6391	0.6543	0.5724
DINA	0.5921	0.6521	0.5141	0.6110	0.6703	0.5041	0.6257	0.6842	0.4965	0.6388	0.6959	0.4897
NCDM	0.7006	0.7293	0.4543	0.7070	0.7403	0.4476	0.7146	0.7480	0.4425	0.7188	0.7554	0.4381
KaNCD	0.7175	0.7433	0.4363	0.7246	0.7538	0.4308	0.7295	0.7611	0.4270	0.7334	0.7675	0.4243
KSCD	0.7021	0.7292	0.4443	0.7069	0.7410	0.4422	0.7153	0.7490	0.4372	0.7178	0.7568	0.4343
KAN2CD	0.7191	0.7528	0.4313	0.7256	0.7584	0.4285	0.7265	0.7638	0.4263	0.7305	0.7682	0.4238
GIRT	0.7110	0.7284	0.4391	0.7136	0.7383	0.4368	0.7168	0.7459	0.4348	0.7195	0.7522	0.4329
ORCDF	0.6671	0.6229	0.4656	0.6665	0.6206	0.4655	0.6661	0.6234	0.4652	0.6681	0.6235	0.4647
SymbolicCDM	0.6716	0.6839	–	0.6953	0.6981	–	0.6510	0.6848	–	0.6960	0.7073	–
EMO-A7	0.7278	0.7576	0.4342	0.7425	0.7810	0.4256	0.7566	0.8018	0.4165	0.7706	0.8211	0.4070
OSCD-AA1	0.7219	0.7584	0.4303	0.7283	0.7648	0.4266	0.7308	0.7695	0.4245	0.7340	0.7737	0.4218
Agent-NASCD	**0.7310**	**0.7621**	**0.4269**	**0.7465**	**0.7843**	**0.4183**	**0.7568**	**0.8020**	**0.4141**	**0.7779**	**0.8281**	**0.3939**

Table 1 compares selected architectures across twelve dataset–ratio settings:

(1) On ASSIST09, Agent-NASCD attains the best AUC, ACC, and RMSE among the compared architectures at all four ratios. At 70%, its AUC, ACC, and RMSE are 0.8281, 0.7779, and 0.3939, compared with 0.8211, 0.7706, and 0.4070 for EMO-A7. Compared with EMO-A7, the runner-up on ACC at every ratio and on RMSE at 50–70%, Agent-NASCD achieves consistent gains, including an AUC improvement of 0.0002 at 60%. The largest AUC gain occurs at 70% training.

(2) On XES-2k, the agent AUC increases from 0.7839 to 0.7925 as the training ratio grows, and it exceeds OSCD-AA1 by 0.0017–0.0028 on all four settings. It achieves the best ACC, AUC, and RMSE at every ratio; at 70%, its AUC and RMSE are 0.7925 and 0.3481, versus 0.7881 and 0.3498 for KAN2CD. OSCD-AA1 is the closest AUC competitor at every ratio.

(3) On MOOC-2k, the performance differences among the strongest models are smaller. Agent-NASCD attains the highest displayed AUC at all four ratios (0.9275, 0.9298, 0.9322, and 0.9345). At 70%, its ACC and RMSE are 0.9170 and 0.2452. MIRT reaches AUC 0.9288, 0.9313, and 0.9335 at 50%, 60%, and 70%. At 70%, the RMSE is close to the best value of 0.2451 achieved by EMO-A7, alongside the highest displayed AUC. The searched interaction functions therefore achieve best or near-best performance across the reported predictive metrics. The MIRT-40 numerical fallback is marked in the table and excluded from predictive ranking.

5.3 Search Efficiency

Table 2 reports ASSIST09 70% runtime for all thirteen main-table models. A single CD model usually finishes in minutes (0.37–70.49); complete agent search averages 9.73 hours, including roots, candidate evaluations, LLM interaction, and confirmation. All three search methods are rerun in the same experimental codebase with matched data partitions, hardware, and batch size. The EMO-A7 and OSCD-AA1 entries isolate post-search model-training time, separately from the architecture-

Table 1: Prediction performance (continued): (b) XES-2k and (c) MOOC-2k. Metric notation, displayed ties, and ranking exclusions follow the conventions in panel (a).

(b) XES-2k

Model	40%			50%			60%			70%		
	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓
IRT	0.8267	0.7656	0.3562	0.8281	0.7727	0.3545	0.8299	0.7784	0.3526	0.8306	0.7820	0.3519
MIRT	0.7242	0.6096	0.4999	0.7348	0.6207	0.4906	0.7436	0.6284	0.4831	0.7133	0.6092	0.5085
DINA	0.5461	0.6161	0.4842	0.5607	0.6236	0.4790	0.5747	0.6298	0.4739	0.5876	0.6351	0.4696
NCDM	0.8192	0.7217	0.3686	0.8192	0.7262	0.3679	0.8212	0.7299	0.3665	0.8203	0.7337	0.3663
KaNCD	0.8194	0.7358	0.3664	0.8203	0.7475	0.3646	0.8232	0.7557	0.3614	0.8247	0.7646	0.3585
KSCD	0.8210	0.7351	0.3646	0.8225	0.7468	0.3624	0.8260	0.7576	0.3596	0.8246	0.7648	0.3587
KAN2CD	0.8309	0.7805	0.3518	0.8317	0.7840	0.3509	0.8329	0.7867	0.3508	0.8331	0.7881	0.3498
GIRT	0.8219	0.7738	0.3619	0.8213	0.7769	0.3620	0.8212	0.7791	0.3617	0.8204	0.7806	0.3621
ORCDF	0.8178	0.6232	0.3827	0.8173	0.6239	0.3820	0.8172	0.6242	0.3823	0.8164	0.6265	0.3826
SymbolicCDM	0.8185	0.6947	–	0.8176	0.7020	–	0.8204	0.7134	–	0.8203	0.7159	–
EMO-A7	0.8241	0.7605	0.3584	0.8259	0.7693	0.3560	0.8283	0.7757	0.3538	0.8293	0.7815	0.3525
OSCD-AA1	0.8306	0.7811	0.3521	0.8324	0.7852	0.3506	0.8330	0.7879	0.3495	0.8323	0.7898	0.3502
Agent-NASCD	**0.8329**	**0.7839**	**0.3502**	**0.8331**	**0.7871**	**0.3496**	**0.8358**	**0.7896**	**0.3491**	**0.8351**	**0.7925**	**0.3481**

(c) MOOC-2k

Model	40%			50%			60%			70%		
	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓	ACC ↑	AUC ↑	RMSE ↓
IRT	0.9124	0.9239	0.2507	0.9142	0.9266	0.2489	0.9112	0.9231	0.2518	0.9105	0.9234	0.2520
MIRT	0.8785†	0.5000†	0.5000†	0.9149	0.9288	0.2485	0.9160	0.9313	0.2465	0.9161	0.9335	0.2459
DINA	0.4626	0.7100	0.4790	0.4912	0.7164	0.4682	0.5065	0.7211	0.4622	0.5351	0.7281	0.4513
NCDM	0.8905	0.8353	0.2863	0.8924	0.8418	0.2837	0.8940	0.8471	0.2816	0.8952	0.8519	0.2800
KaNCD	0.9110	0.9230	0.2533	0.9130	0.9258	0.2507	0.9138	0.9276	0.2498	0.9145	0.9299	0.2485
KSCD	0.9086	0.9197	0.2555	0.9113	0.9230	0.2535	0.9124	0.9260	0.2512	0.9143	0.9276	0.2489
KAN2CD	0.9114	0.9222	0.2533	0.9123	0.9238	0.2514	0.9122	0.9247	0.2511	0.9130	0.9255	0.2506
GIRT	0.8882	0.8915	0.2825	0.8884	0.8897	0.2823	0.8885	0.8887	0.2820	0.8887	0.8880	0.2816
ORCDF	0.8955	0.8647	0.2767	0.8959	0.8659	0.2761	0.8968	0.8688	0.2743	0.8972	0.8695	0.2742
SymbolicCDM	0.8847	0.8501	–	0.8841	0.8576	–	0.8839	0.8674	–	0.8826	0.8704	–
EMO-A7	0.9112	0.9244	0.2524	0.9135	0.9283	0.2492	0.9151	0.9311	0.2470	0.9164	0.9336	**0.2451**
OSCD-AA1	0.9133	0.9270	0.2500	0.9146	0.9297	0.2480	0.9158	0.9319	0.2462	0.9158	0.9336	0.2456
Agent-NASCD	**0.9136**	**0.9275**	**0.2499**	**0.9151**	**0.9298**	**0.2479**	**0.9169**	**0.9322**	**0.2456**	**0.9170**	**0.9345**	0.2452

Table 2: Runtime on ASSIST09 at 70% training. The first twelve model columns give single-model minutes; the final three give search cost (d: GPU days; h: hours). S-CDM: SymbolicCDM; EMO: EMO-NAS-CD; Ours: Agent-NASCD. All runtimes are measured locally under the same experimental setup. Entries are means over ten runs; **Appendix** F gives timing definitions, variability, and resource counts.

	Single-model (min)												Search		
	IRT	MIRT	DINA	NCDM	KaNCD	KSCD	KAN2CD	GIRT	ORCDF	S-CDM	EMO-A7	OSCD-AA1	EMO	OSCD	**Ours**
Time	4.69	4.52	4.95	1.56	1.06	1.44	13.48	0.37	3.96	70.49	6.40	1.30	15.2 d	26.1 h	**9.73 h**

search costs listed for EMO-NAS-CD and OSCD. The measured mean search costs are 15.2 GPU days for EMO-NAS-CD, 26.1 hours for OSCD, and 9.73 hours for Agent-NASCD.

Figure 3 reports evaluation efficiency on the same dataset. The seed-33 Agent-NASCD run reaches $U \geq 0.82$ at evaluation 57 and $U = 0.828$ at 83, with confirmed test AUC 0.8281 after 86 evaluations; our EMO-NAS-CD run reaches 0.82 at 2,885 and 0.8248 at 10,100 evaluations. At 120 evaluations, EMO-NAS-CD attains a best recorded AUC of only 0.7773, well below the 0.8280 reached by Agent-NASCD after 83 evaluations, a gap of 0.0507. Full evaluation uses the same candidate-evaluation procedure and training budget as EMO-NAS-CD, while Proxy uses fewer training epochs and is computationally cheaper. The comparison of training cost and elapsed time is reported with the component study in Section 5.4.

5.4 Controlled Component Studies

Table 3 compares the search components using results averaged over ten runs on ASSIST09 70/10/20. Direct LLM Search uses the LLM alone in a ReAct loop without the proposed search modules. Flat-History retains the ReAct loop with flat candidate records, Generic-Error coarsens diagnostics, and Fixed-Policy replaces agentic action selection with fixed rules. The complete method leads all five search variants on AUC, ACC, and RMSE; against Fixed-Policy, AUC rises from

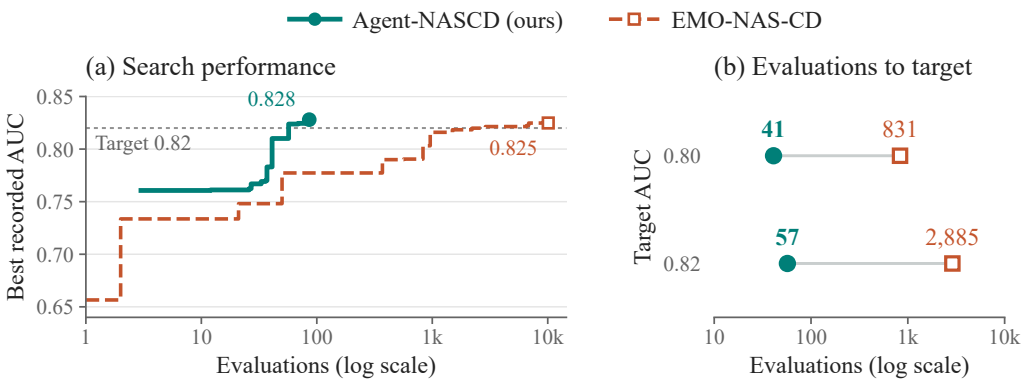

Figure 3: Anytime search performance versus candidate evaluations on ASSIST09 (70%). (a) Best validation AUC along representative local search trajectories of Agent-NASCD (seed 33) and EMO-NAS-CD. (b) Evaluations to first reach each target AUC; both panels use logarithmic count axes. Evaluation counts measure candidate use; elapsed search time is reported separately in Table 2.

0.7969 to 0.8281 with 86 versus 120 evaluations and 5.35 versus 8.06 training hours. Replacing π_ϕ with DeepSeek-V4-Pro or Claude-Sonnet-5 yields mean AUC values of 0.8228 and 0.8237, respectively. Flat-History and Generic-Error perform worse with the same LLM and evaluator, supporting structured observations and diagnostic feedback. Flat-History stops at 83 evaluations and uses 5.75 training hours on average, exceeding the complete method's 5.35 hours. Write-Only achieves mean test AUC, ACC, and RMSE of 0.8133, 0.7647, and 0.4049, respectively, with an average of 35 evaluations and 1.85 training hours. **Appendix** E gives intervention settings, evaluation stages, reliability measures, and paired repair results under the same evaluator and repair allowance.

Table 3: ASSIST09 70% component and controller comparisons. Values are means over ten runs; training time includes confirmation. All rows report confirmed test metrics; the complete method uses the same results as Table 1. Bold marks the best values within the controller group.

Variant	AUC↑	ACC↑	RMSE↓	Evals↓	Train (h)↓
Controller					
Agent-NASCD (GLM-5.3-Flash)	**0.8281**	**0.7779**	**0.3939**	86	5.35
DeepSeek-V4-Pro	0.8228	0.7731	0.3961	100	5.82
Claude-Sonnet-5	0.8237	0.7759	0.3962	**83**	**5.21**
Component					
Direct LLM Search	0.7602	0.7231	0.4260	80	5.21
Flat-History	0.7657	0.7318	0.4230	83	5.75
Generic-Error	0.7931	0.7480	0.4121	52	4.54
Fixed-Policy	0.7969	0.7466	0.4210	120	8.06
Write-Only	0.8133	0.7647	0.4049	35	1.85

Search Reliability. Table 11 reports mean repair performance over ten seeds, with 100 paired invalid-program cases evaluated under each seed. On average, detailed feedback repairs 96 of the 100 cases using 153 calls, compared with 89 cases and 174 calls under generic feedback. Each case allows up to three repair responses without candidate training, isolating the effect of feedback on program repair. These results support recovery from invalid proposals with fewer repair calls.

5.5 Trajectory Case Study

Figure 4(a) shows the seed-33 MOG on ASSIST09 70%. It contains 95 candidate records and 91 derivation edges, with four roots and nine parents that have descendants. Several parent nodes serve as centers of local exploration. Specifically, the selected `n82` descends from `n03` through `n35`, `n41`, `n45`, `n51`, and `n68`. With nine descendants, the selected `n82` is an internal graph node. Confirmation selects `n82` by validation AUC (0.8280); its test AUC is 0.8281. The graph thus preserves the selected architecture's lineage together with the alternatives evaluated during

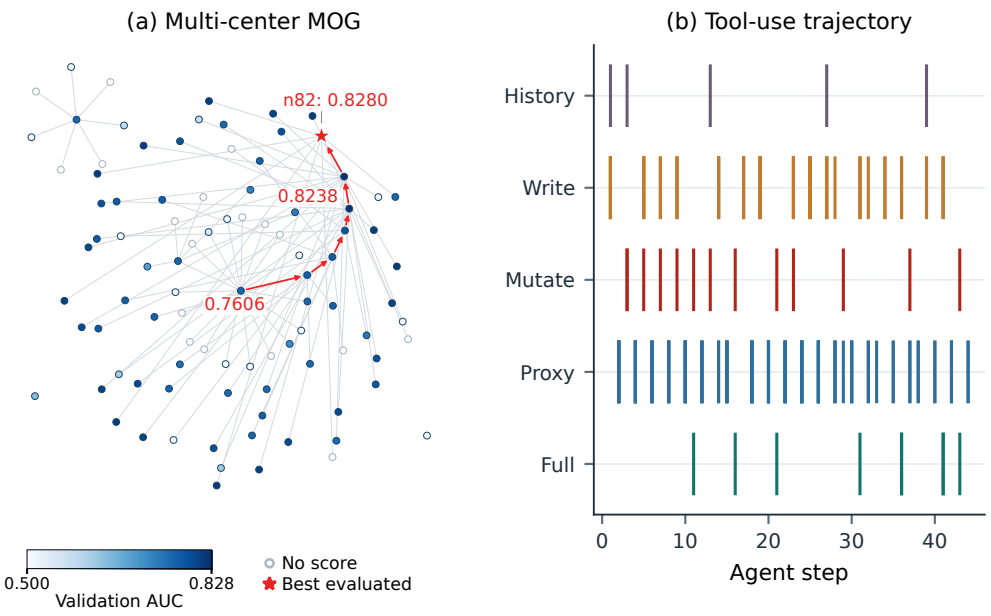

Figure 4: ASSIST09 search records. (a) Seed-33 MOG; red arrows trace the lineage of validation-selected `n82`, and positions are schematic. (b) Seed-7 tool actions. Inspection of `n48` precedes refinement toward the selected residual–cosine program `n129`.

search. Graph positions are schematic and encode neither architectural similarity nor monotonic AUC improvement.

Figure 4(b) shows the seed-7 tool-use trajectory on the same dataset. The run issues 101 evaluation requests (88 completed trainings, 12 nonconvergence outcomes, and one cached duplicate), 60 writes, 13 mutations, and five history inspections; after inspecting `n48` at step 13, the agent evaluates `n58` and `n60` at step 14 and writes residual–cosine variants on that branch. The selected `n129` computes a concept-modulated residual $r = (h_s - h_e) \odot h_c$ and its bounded transformation $\tilde{r} = \tanh(r)$, then forms two cosine differences:

$$d_{\text{sat}} = \cos(\tilde{r}, h_s) - \cos(\tilde{r}, h_e), \qquad d_{\text{raw}} = \cos(r, h_s) - \cos(r, h_e). \tag{6}$$

The terminal `mix_gs` averages and clips these contrasts to $[0, 1]$, bypassing the vector-output prediction head. Additional training improves this candidate's validation AUC from 0.8024 at proxy fidelity to 0.8170 at full fidelity. This sequence connects evidence retrieval with branch refinement and fidelity allocation: the controller uses tool outcomes to choose its next search action. **Appendix** D gives controller prompts, example tool-use trajectories, and executable encodings for both selected architectures.

6 Conclusion

In this paper, we proposed Agent-NASCD, which formulates CD architecture search as sequential evidence-conditioned decision making. Specifically, CD tools were provided for writing and mutating operator programs, a model optimization graph was maintained to store lineage and scores, and constrained proxy/full evaluation was used to train only requested, legal candidates. Experiments on three datasets support the selected architectures' predictive quality and the contributions of graph observations and diagnostic feedback. The complete method outperforms Fixed-Policy with fewer evaluations and lower training cost in the controlled comparison. Future work will examine larger search spaces and compare adaptive fidelity allocation under different wall-time budgets.

AI USE STATEMENT

AI tools were used to polish the writing of this paper and prepare figures. The authors remain responsible for the manuscript and accompanying materials.

ETHICS STATEMENT

Cognitive diagnosis can influence how learners are assessed and supported. Prediction accuracy alone does not justify consequential placement or intervention decisions, and learned proficiency representations should not be treated as complete measures of individual ability. The study uses existing benchmark response data and does not involve a new learner intervention. Any deployment would require appropriate data permissions, privacy safeguards, and evaluation of error disparities across relevant learner groups. The searched interaction functions are presented as predictive models, not validated causal explanations of individual learning processes.

REPRODUCIBILITY STATEMENT

The manuscript specifies the architecture representation, enabled operators, tool semantics, training recipes, partition conventions, and available run counts. Experimental results are averaged over ten random seeds, while the case studies illustrate individual search trajectories. Test responses are held out during search and used to evaluate the architectures after final selection, following the protocol in Section 5.1.

REFERENCES

Jimmy de la Torre. DINA model and parameter estimation: A didactic. *Journal of Educational and Behavioral Statistics*, 34(1):115–130, 2009. doi: 10.3102/1076998607309474.

Jiatong Li, Qi Liu, and Mengxiao Zhu. Generative cognitive diagnosis. *arXiv preprint arXiv:2507.09831*, 2025. URL https://arxiv.org/abs/2507.09831.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence*, volume 115 of *Proceedings of Machine Learning Research*, pp. 367–377, 2020.

Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020. URL https://jmlr.org/papers/v21/20-056.html.

Yuanhao Liu, Shuo Liu, Yimeng Liu, Chanjin Zheng, Wei Zhang, and Hong Qian. A dual-fusion cognitive diagnosis framework for open student learning environments. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025. doi: 10.1145/3711896.3736820.

Frederic M. Lord. *Applications of Item Response Theory to Practical Testing Problems*. Lawrence Erlbaum Associates, 1980.

Haiping Ma, Manwei Li, Le Wu, Haifeng Zhang, Yunbo Cao, Xingyi Zhang, and Xuemin Zhao. Knowledge-sensed cognitive diagnosis for intelligent education platforms. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, pp. 1451–1460, 2022. doi: 10.1145/3511808.3557372.

Hong Qian, Shuo Liu, Mingjia Li, Bingdong Li, Zhi Liu, and Aimin Zhou. ORCDF: An oversmoothing-resistant cognitive diagnosis framework for student learning in online education systems. *arXiv preprint arXiv:2407.17476*, 2024. URL https://arxiv.org/abs/2407.17476.

Mark D. Reckase. *Multidimensional Item Response Theory*. Springer, 2009.

Junhao Shen, Hong Qian, Wei Zhang, and Aimin Zhou. Symbolic cognitive diagnosis via hybrid optimization for intelligent education systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 14928–14936, 2024. doi: 10.1609/aaai.v38i13.29413.

Fei Wang, Qi Liu, Enhong Chen, Zhenya Huang, Yuying Chen, Yu Yin, Zai Huang, and Shijin Wang. Neural cognitive diagnosis for intelligent education systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6153–6161, 2020. doi: 10.1609/aaai.v34i04.6080.

Fei Wang, Qi Liu, Enhong Chen, Zhenya Huang, Yu Yin, Shijin Wang, and Yu Su. NeuralCD: A general framework for cognitive diagnosis. *IEEE Transactions on Knowledge and Data Engineering*, 35(8):8312–8327, 2023. doi: 10.1109/TKDE.2022.3201037.

Ziwen Wang, Shangshang Yang, Xiaoshan Yu, Haiping Ma, and Xingyi Zhang. Breaking robustness barriers in cognitive diagnosis: A one-shot neural architecture search perspective. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, 2026. doi: 10.1145/3770854.3780159. URL https://arxiv.org/abs/2601.04918.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024a.

Shangshang Yang, Ye Tian, Xiaoshu Xiang, Shichen Peng, and Xingyi Zhang. Accelerating evolutionary neural architecture search via multi-fidelity evaluation. *arXiv preprint arXiv:2108.04541*, 2021. URL https://arxiv.org/abs/2108.04541.

Shangshang Yang, Xiaoshan Yu, Ye Tian, Xueming Yan, Haiping Ma, and Xingyi Zhang. Evolutionary neural architecture search for transformer in knowledge tracing. In *Advances in Neural Information Processing Systems*, 2023.

Shangshang Yang, Mingyang Chen, Ziwen Wang, Xiaoshan Yu, Panpan Zhang, Haiping Ma, and Xingyi Zhang. DisenGCD: A meta multigraph-assisted disentangled graph learning framework for cognitive diagnosis. In *Advances in Neural Information Processing Systems*, 2024b.

Shangshang Yang, Haiping Ma, Ying Bi, Ye Tian, Limiao Zhang, Yaochu Jin, and Xingyi Zhang. An evolutionary multiobjective neural architecture search approach to advancing cognitive diagnosis in intelligent education. *IEEE Transactions on Evolutionary Computation*, 29(6):2431–2445, 2025a. doi: 10.1109/TEVC.2024.3429180.

Shangshang Yang, Linrui Qin, Xiaoshan Yu, Ziwen Wang, Xueming Yan, Haiping Ma, and Ye Tian. Endowing interpretability for neural cognitive diagnosis by efficient Kolmogorov-Arnold networks. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 7894–7903, 2025b. doi: 10.24963/ijcai.2025/878.

Shangshang Yang, Linrui Qin, Xiaoshan Yu, Ziwen Wang, Ye Tian, Xueming Yan, Haiping Ma, and Xingyi Zhang. Large language model-assisted evolutionary neural architecture search for cognitive diagnosis. *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–20, 2026. doi: 10.1109/TETCI.2026.3683785. Early access.

Zekang Yang, Wang Zeng, Sheng Jin, Chen Qian, Ping Luo, and Wentao Liu. NADER: Neural architecture design via multi-agent collaboration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4452–4461, 2025c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

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

A SUPPLEMENTARY EVALUATION RESULTS

Search and Training Configuration. Candidate training uses dimension 128, batch size 256, Adam with learning rate 10^{-3}, proxy/full limits of 55/100 epochs, and full-training patience 12. The soft budget vector B specifies thresholds of 120 evaluations, 400 controller turns, 800 LLM calls, and nine hours; one turn may request several tools, while retries also consume LLM calls. Resource checks occur at turn or tool boundaries; τ is the last, possibly partial turn, and V_τ^{eval} is restricted to the Top-K candidates successfully confirmed at Full fidelity when confirmation is enabled. Final confirmation evaluates up to $K{=}10$ candidates with the highest validation scores outside these loop thresholds and selects the best candidate among those successfully confirmed at Full fidelity. Baseline-specific parameters follow the original papers; data, partitions, batch size, and hardware are shared across methods, with architecture selection based only on validation and final assessment on held-out test responses.

Table 4: Processed benchmark statistics. ASSIST09 counts distinct IDs observed in the response records; its embedding index bounds are 4,128 students and 17,746 exercises. Each XES-2k and MOOC-2k subset contains responses from 2,000 students.

Dataset	Students	Exercises	Concepts	Responses
ASSIST09	2,493	17,671	123	267,415
XES-2k	2,000	1,624	241	207,204
MOOC-2k	2,000	915	696	385,323

Table 1 reports the available predictive metrics. Tables 5–7 give AUC means and sample standard deviations over ten seeds on the original 0–1 scale. Table 8 reports Wilcoxon signed-rank tests on these ten-seed AUC values as $+/-/\approx$ counts and results for each setting. Each metric is averaged only over records containing that metric; SymbolicCDM has AUC and ACC for all twelve settings but no reported RMSE. Displayed ties share a rank, while unavailable values and the flagged MIRT fallback outputs are excluded from the ranking of predictive performance.

Predictive results are averaged over the same ten seeds for each method and dataset–ratio setting. Repeated attempts under the same seed are not treated as independent runs. Each NAS method performs a fresh search in all twelve dataset–ratio settings for each of the ten shared seeds, giving 120 searches per method with matched response partitions. The main comparison includes thirteen model rows with recorded AUC and ACC across all twelve settings. Scores are averaged across runs for each dataset–ratio setting.

Baseline Configuration Details. The compared models use the parameter settings specified in their original papers. In particular, GIRT, ORCDF, and SymbolicCDM follow their original experimental configurations for training and evaluation. The IRT runs used bounded parameters. MIRT follows the original model's parameterization and training configuration. The MOOC-2k 40% entry for MIRT contains numerical-fallback outputs. The corresponding mean AUC and RMSE are both 0.5000, as marked in Table 1. These values are displayed with † but excluded from predictive ranking because they reflect a numerical fallback rather than successful model fitting.

Wilcoxon Signed-Rank Tests. Wilcoxon signed-rank tests (5% significance level) were conducted on paired AUC values from the same seeds. Table 8 shows the results, where '$+$', '$-$', and '$\approx$' indicate whether Agent-NASCD is significantly better than, significantly worse than, or not significantly different from the compared model. Each $+/-/\approx$ triple summarizes the test results across the twelve dataset–ratio settings. The grid reports a separate signed-rank test for each dataset–ratio setting. The MIRT count excludes the unranked fallback (†).

Random Seeds. All compared methods use the same ten random seeds in every dataset–ratio setting. The same pool is used for predictive means, sample standard deviations, and Wilcoxon comparisons, without counting failed attempts as independent seeds. Component studies and runtime summaries are also averaged over ten runs.

Cognitive Diagnostic Capability. To evaluate cognitive diagnostic capability, we report the degree of agreement (DOA) between inferred concept proficiency and observed response performance.

Table 5: AUC mean ± sample standard deviation over ten seeds on ASSIST09 (0–1).

Model	40%	50%	60%	70%
IRT	0.7178 ± 0.0009	0.6896 ± 0.0055	0.7032 ± 0.0089	0.6896 ± 0.0294
MIRT	0.6084 ± 0.0033	0.6218 ± 0.0017	0.6382 ± 0.0054	0.6543 ± 0.0053
DINA	0.6521 ± 0.0009	0.6703 ± 0.0010	0.6842 ± 0.0002	0.6959 ± 0.0014
NCDM	0.7293 ± 0.0008	0.7403 ± 0.0012	0.7480 ± 0.0010	0.7554 ± 0.0012
KaNCD	0.7433 ± 0.0015	0.7538 ± 0.0021	0.7611 ± 0.0016	0.7675 ± 0.0011
KSCD	0.7292 ± 0.0013	0.7410 ± 0.0007	0.7490 ± 0.0007	0.7568 ± 0.0008
KAN2CD	0.7528 ± 0.0010	0.7584 ± 0.0009	0.7638 ± 0.0005	0.7682 ± 0.0010
GIRT	0.7284 ± 0.0003	0.7383 ± 0.0004	0.7459 ± 0.0007	0.7522 ± 0.0013
ORCDF	0.6229 ± 0.0012	0.6206 ± 0.0011	0.6234 ± 0.0020	0.6235 ± 0.0036
SymbolicCDM	0.6839 ± 0.0126	0.6981 ± 0.0048	0.6848 ± 0.0379	0.7073 ± 0.0115
EMO-A7	0.7576 ± 0.0012	0.7810 ± 0.0014	0.8018 ± 0.0014	0.8211 ± 0.0014
OSCD-AA1	0.7584 ± 0.0002	0.7648 ± 0.0005	0.7695 ± 0.0003	0.7737 ± 0.0011
Agent-NASCD	0.7621 ± 0.0005	0.7843 ± 0.0119	0.8020 ± 0.0147	0.8281 ± 0.0051

Table 6: AUC mean ± sample standard deviation over ten seeds on XES-2k (0–1).

Model	40%	50%	60%	70%
IRT	0.7656 ± 0.0037	0.7727 ± 0.0031	0.7784 ± 0.0020	0.7820 ± 0.0005
MIRT	0.6096 ± 0.0042	0.6207 ± 0.0037	0.6284 ± 0.0015	0.6092 ± 0.0432
DINA	0.6161 ± 0.0007	0.6236 ± 0.0005	0.6298 ± 0.0009	0.6351 ± 0.0013
NCDM	0.7217 ± 0.0021	0.7262 ± 0.0019	0.7299 ± 0.0031	0.7337 ± 0.0041
KaNCD	0.7358 ± 0.0082	0.7475 ± 0.0053	0.7557 ± 0.0047	0.7646 ± 0.0057
KSCD	0.7351 ± 0.0086	0.7468 ± 0.0065	0.7576 ± 0.0032	0.7648 ± 0.0018
KAN2CD	0.7805 ± 0.0013	0.7840 ± 0.0019	0.7867 ± 0.0006	0.7881 ± 0.0024
GIRT	0.7738 ± 0.0016	0.7769 ± 0.0014	0.7791 ± 0.0005	0.7806 ± 0.0003
ORCDF	0.6232 ± 0.0020	0.6239 ± 0.0026	0.6242 ± 0.0020	0.6265 ± 0.0044
SymbolicCDM	0.6947 ± 0.0015	0.7020 ± 0.0029	0.7134 ± 0.0018	0.7159 ± 0.0047
EMO-A7	0.7605 ± 0.0018	0.7693 ± 0.0015	0.7757 ± 0.0015	0.7815 ± 0.0026
OSCD-AA1	0.7811 ± 0.0024	0.7852 ± 0.0019	0.7879 ± 0.0008	0.7898 ± 0.0019
Agent-NASCD	0.7839 ± 0.0020	0.7871 ± 0.0019	0.7896 ± 0.0020	0.7925 ± 0.0023

Table 7: AUC mean ± sample standard deviation over ten seeds on MOOC-2k (0–1).

Model	40%	50%	60%	70%
IRT	0.9239 ± 0.0007	0.9266 ± 0.0013	0.9231 ± 0.0045	0.9234 ± 0.0025
MIRT	$0.5000 \pm 0.0000^{\dagger}$	0.9288 ± 0.0014	0.9313 ± 0.0017	0.9335 ± 0.0024
DINA	0.7100 ± 0.0017	0.7164 ± 0.0004	0.7211 ± 0.0020	0.7281 ± 0.0010
NCDM	0.8353 ± 0.0012	0.8418 ± 0.0015	0.8471 ± 0.0018	0.8519 ± 0.0025
KaNCD	0.9230 ± 0.0009	0.9258 ± 0.0008	0.9276 ± 0.0010	0.9299 ± 0.0010
KSCD	0.9197 ± 0.0003	0.9230 ± 0.0019	0.9260 ± 0.0018	0.9276 ± 0.0022
KAN2CD	0.9222 ± 0.0010	0.9238 ± 0.0013	0.9247 ± 0.0019	0.9255 ± 0.0023
GIRT	0.8915 ± 0.0007	0.8897 ± 0.0015	0.8887 ± 0.0021	0.8880 ± 0.0027
ORCDF	0.8647 ± 0.0849	0.8659 ± 0.0869	0.8688 ± 0.0867	0.8695 ± 0.0880
SymbolicCDM	0.8501 ± 0.0127	0.8576 ± 0.0113	0.8674 ± 0.0073	0.8704 ± 0.0104
EMO-A7	0.9244 ± 0.0007	0.9283 ± 0.0007	0.9311 ± 0.0009	0.9336 ± 0.0013
OSCD-AA1	0.9270 ± 0.0006	0.9297 ± 0.0013	0.9319 ± 0.0016	0.9336 ± 0.0021
Agent-NASCD	0.9275 ± 0.0034	0.9298 ± 0.0021	0.9322 ± 0.0017	0.9345 ± 0.0044

Table 9 gives the mean DOA values over ten seeds on XES-2k. Agent-NASCD achieves higher DOA than EMO-A7 at all four training ratios, with absolute gains of 0.0108–0.0520. These results show that the proficiency estimates produced by Agent-NASCD are more consistent with students' observed responses, supporting its cognitive diagnostic capability.

Table 8: Wilcoxon signed-rank test results with a 5% significance level. '+', '−', and '≈' indicate whether Agent-NASCD is significantly better than, worse than, or not significantly different from the compared model. Each $+/-/\approx$ triple summarizes the 12 dataset–ratio settings (11 for MIRT). The grid reports paired tests over the ten matched seeds in each setting. The Agent-NASCD row is the reference. †: unranked MIRT fallback, excluded from the count.

$+/-/\approx$	IRT	MIRT	DINA	NCDM	KaNCD	KSCD
Agent-NASCD	10/0/2	8/0/3†	12/0/0	12/0/0	10/0/2	11/0/1
$+/-/\approx$	KAN2CD	GIRT	ORCDF	SymCDM	EMO-A7	OSCD
Agent-NASCD	10/0/2	12/0/0	12/0/0	12/0/0	5/0/7	4/0/8

	ASSIST09				**XES-2k**				**MOOC-2k**			
Model	40%	50%	60%	70%	40%	50%	60%	70%	40%	50%	60%	70%
IRT	+	+	+	+	+	+	+	+	≈	≈	+	+
MIRT	+	+	+	+	+	+	+	+	–†	≈	≈	≈
DINA	+	+	+	+	+	+	+	+	+	+	+	+
NCDM	+	+	+	+	+	+	+	+	+	+	+	+
KaNCD	+	+	+	+	+	+	+	+	≈	+	+	≈
KSCD	+	+	+	+	+	+	+	+	+	+	+	≈
KAN2CD	+	+	+	+	+	+	+	≈	≈	+	+	+
GIRT	+	+	+	+	+	+	+	+	+	+	+	+
ORCDF	+	+	+	+	+	+	+	+	+	+	+	+
SymbolicCDM	+	+	+	+	+	+	+	+	+	+	+	+
EMO-A7	+	≈	≈	≈	+	+	+	+	≈	≈	≈	≈
OSCD-AA1	+	+	+	+	≈	≈	≈	≈	≈	≈	≈	≈
Agent-NASCD	–	–	–	–	–	–	–	–	–	–	–	–

Table 9: DOA on XES-2k at four training ratios (higher is better). Values are means over ten seeds; bold marks the best result at each training ratio.

Model	**40%**	**50%**	**60%**	**70%**
EMO-A7	0.8356	0.8401	0.8445	0.8469
Agent-NASCD	**0.8876**	**0.8509**	**0.8694**	**0.8814**

B Complete Search Procedure

Algorithm 1 presents the loop of Eq. (3) and final selection with optional full confirmation, following the numbered-assignment notation of EMO-NAS-CD (Yang et al., 2025a). Here t indexes controller responses, u_t is an ordered batch of tool requests, and o_{t+1} collects their returned observations. ExecuteBatch processes requests sequentially with the runner's resource checks, stopping the batch if a check or accepted termination ends search. Each tool validates its inputs; an uncached `evaluate_arch` request trains once at its requested 55/100-epoch fidelity, whereas a cache hit returns the stored result without launching a second candidate-training job.

C Architecture Registry and Execution Semantics

Table 10 lists every implemented operator. The notes below clarify execution details beyond the operator names, including output handling and the FFN admission threshold.

The definitions in Table 10 specify the implemented functions rather than relying on operator names alone. Specifically, `q_mask` thresholds its supplied latent tensor, not a newly loaded binary Q-matrix; `irt_2pl` is a sigmoid difference, not a complete separately parameterized psychometric model; and `mix_gs` is an arithmetic mean, not a learned gate. Sum, mean, and FFN reject scalar inputs; cosine and inner product require equal operand shapes. However, numerical guards do not ensure finite values in all compositions, so runtime validation remains necessary.

IDs 5 and 14 remain disabled in the reported configuration, so the listed IDs refer to implementation identifiers rather than consecutive table positions. The `concat` operator concatenates its operands and applies a learned projection back to dimension d. A vector terminal output is processed by a $d \rightarrow$

Algorithm 1 Main steps of Agent-NASCD

Input: $\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{score}}, Q, \mathcal{R}, \pi_\phi, B, K$, confirmation flag c
Output: $a^\star \in \mathcal{A}, \theta^\star_{a^\star}$

$G_0 \leftarrow \text{Initialization}(\mathcal{R})$
Evaluate each $a \in V_0$ at proxy fidelity ($N_{\text{fid}} = 55$) and record its validation score
$C_{\text{roots}} \leftarrow \text{cost}(V_0)$; $H_0 \leftarrow \emptyset$; $t \leftarrow 0$
$R_0 \leftarrow \text{ResourceUsage}()$; $b_0 \leftarrow (B - R_0)_+$
while $R_t \prec B$ and `finish` is not accepted **do**
    $O(G_t) \leftarrow \text{Observation}(G_t)$
    $u_t \sim \pi_\phi(\cdot \mid H_t, O(G_t), b_t)$
    $(o_{t+1}, G_{t+1}) \leftarrow \text{ExecuteBatch}(u_t, G_t)$
    $H_{t+1} \leftarrow H_t \cup \{(u_t, o_{t+1})\}$; $R_{t+1} \leftarrow \text{ResourceUsage}()$
    $b_{t+1} \leftarrow (B - R_{t+1})_+$; $t \leftarrow t + 1$
end while
$V_t^{\text{eval}} \leftarrow \{a \in V_t : \text{evaluation completed successfully}\}$
if c is enabled **then**
    $\mathcal{A}_K \leftarrow \text{TopK}(V_t^{\text{eval}}, K)$
    Fully evaluate $\mathcal{A}_K$ and update validation scores using Full results
    $V_t^{\text{eval}} \leftarrow \{a \in \mathcal{A}_K : \text{Full confirmation completed successfully}\}$
end if
$a^\star \leftarrow \arg\max_{a \in V_t^{\text{eval}}} U(a; \mathcal{D}_{\text{score}})$
return $a^\star, \theta^\star_{a^\star}$

$512 \to 256 \to 1$ prediction head with sigmoid activations and dropout 0.5 after the first two layers; its effective linear weights are elementwise absolute values of learned weights. FFN admission is governed by a configured threshold on the best completed validation result. The threshold uses completed search scores at either fidelity and does not access the held-out test set. It applies to the searchable FFN operator, not to all learned linear maps or the terminal head.

The OSCD baseline follows the original method's architecture-search procedure and parameter settings. OSCD-AA1 labels the architectures selected independently for each dataset–ratio–seed setting, with predictive performance measured on the corresponding held-out test partition.

D Prompts and Example Trajectories

This appendix records the prompts given to π_ϕ and two abbreviated ReAct traces. Following ReAct (Yao et al., 2023), each trace is written as numbered Thought / Action / Observation turns. The prompt and tool-result examples are condensed templates using the validation-AUC objective of the reported experiments. Unrecorded metric values are written as .... Recorded trajectory scores and their interpretation for both examples are given in Section 5.5.

D.1 Prompts

The following presents the system prompt and the compact user observation $O(G_t)$.

System prompt

```
You are an expert agent for designing Cognitive Diagnosis (CD) neural
architectures.
[GOAL] Find a valid CD architecture with high validation AUC.
Primary objective:  MAXIMIZE validation AUC. Keep the test set held
out until the final architecture has been selected.
```

Inputs (three leaves, all of shape (B, dim))

```
- Stu(n0) = student ability
- Exer(n1) = exercise/item embedding
- Concept(n2) = knowledge-concept vector
```

Operators (only these are allowed; names are case-sensitive)

Table 10: Complete operator registry. Definitions are for one example with $x, y \in \mathbb{R}^d$ and $\epsilon = 10^{-6}$. Unary operators use one input; binary operators use two. Elementwise arithmetic follows supported broadcasting rules. Output dimensions below assume vector inputs. W, w, b, γ are learned parameters. IDs 5 and 14 are disabled, leaving 25 enabled operators; FFN also requires the configured score threshold to be reached before operator admission.

ID	Operator	Definition	Arity	Output
0	`neg`	$-x$	1	d
1	`abs`	$\lvert x \rvert$	1	d
2	`square`	$x \odot x$	1	d
3	`sqrt`	$\mathrm{sign}(x) \odot \sqrt{\lvert x \rvert + \epsilon}$	1	d
4	`tanh`	$\tanh(x)$	1	d
5	`exp` (disabled)	$\exp(x)$	1	d
6	`inv`	$1/(x + \epsilon)$	1	d
7	`sigmoid`	$\sigma(x)$	1	d
8	`softplus`	$\log(1 + \exp(x))$	1	d
9	`sum`	$\sum_{i=1}^{d} x_i$	1	1
10	`ffn`	$w^\top x + b$	1	1
11	`add`	$x + y$	2	d
12	`mul`	$x \odot y$	2	d
13	`concat`	$W[x; y] + b$	2	d
14	`log` (disabled)	$\mathrm{sign}(x) \odot \log(\lvert x \rvert + \epsilon)$	1	d
15	`mean`	$d^{-1} \sum_{i=1}^{d} x_i$	1	1
16	`sub`	$x - y$	2	d
17	`relu`	$\max(x, 0)$	1	d
18	`cosine`	$\frac{x^\top y}{\lVert x \rVert_2 \lVert y \rVert_2 + \epsilon}$	2	1
19	`maximum`	$\max(x, y)$	2	d
20	`rmsnorm`	$\frac{x}{\sqrt{d^{-1} \sum_i x_i^2 + \epsilon}} \odot \gamma$	1	d
21	`minimum`	$\min(x, y)$	2	d
22	`inner`	$\sum_{i=1}^{d} x_i y_i$	2	1
23	`q_mask`	$x \odot \mathbf{1}[y > 0]$	2	d
24	`irt_2pl`	$\sigma(x - y)$	2	d
25	`mix_gs`	$(x + y)/2$	2	d
26	`softmin`	$\mathrm{softmax}(-x) \odot x$	1	d

```
- unary (B,dim)→(B,dim):  neg, abs, square, sqrt, tanh, inv,
sigmoid, softplus, relu, rmsnorm, softmin
- aggregate (B,dim)→(B,1):  sum, ffn, mean
- binary:  add, mul, concat, sub, cosine, maximum, minimum, inner,
q_mask, irt_2pl, mix_gs
```

Architecture validity rules

```
- Nodes numbered from n3; last line out = nX. At most 12 computation
nodes.
- A violation is rejected and is not trained.
```

Tools

```
- view_tree:  compact id / status / AUC / depth / operator-sketch +
lineage
- inspect_node:  exact structure and full metrics for a candidate
- analyze_tree:  operators / output-types / depths vs AUC, plus
Recommended
- write_arch:  write a complete architecture from scratch
- mutate_arch:  generate k validated structural neighbors of a parent
- evaluate_arch:  train at mode=proxy or mode=full
- finish:  stop only after a real plateau
```

Architecture format example (syntax only)

Model-input example

```
Leaves: Stu(n0), Exer(n1), Concept(n2)
n3 = concat(n0, n1)
n4 = ffn(n3)
n5 = mul(n0, n2)
n6 = ffn(n5)
n7 = add(n4, n6)
n8 = sigmoid(n7)
out = n8
```

At each turn, briefly state your thinking, then request the required tool calls.

User message: $O(G_t)$ and b_t

```
Search tree:  ...nodes (...completed, ...failed, ...duplicate).
  BEST = n48 auc=...(sub>mul>tanh>cosine>...)  --- beat it.
  n3 [completed] auc=...d= 4 add>mul>mean>sigmoid
  ...
  n48 [completed] auc=...d=10 sub>mul>tanh>cosine>...<- BEST
Lineage (parent->child):  ...
Remaining evals:  ...(n_evals=.../ max_evals=...)
Use inspect_node(node_id) for a node's exact structure and full
metrics.
```

D.2 Example 1: inspect, mutate, evaluate

The following reconstructs the tool-result format for the seed-7 inspect–mutate–evaluate sequence in Section 5.5, preserving its action order and leaving unrecorded metrics unspecified.

Example 1

```
Thought 1:  inspect the current leader before mutating.
Action 1:  analyze_tree()
Observation 1:  Recommended:  n48 auc=...
Meta-analysis of your evaluated architectures (overall mean AUC=...):
  By operator present:  cosine mean=...**; ...
  By output type:  scalar mean=...
  By depth:  7+ mean=...
  Best so far:  n48 auc=...ops=sub>mul>tanh>cosine>...
Use the validation-AUC ranking and structural evidence for the next
mutate/write.
Rank candidates by validation AUC and select among completed
evaluations.
Thought 2:  refine that parent rather than start an unrelated family.
Action 2:  mutate_arch(parent_id=n48)
Observation 2:  Generated k structural neighbors of n48 (local
search) ---
evaluate the promising ones with evaluate_arch:
  n58:  ...
  n60:  ...
Thought 3:  screen the mutants at proxy fidelity.
Action 3:  evaluate_arch(node_id=n58, mode=proxy)
Observation 3:  Training done n58 (proxy):  auc=...acc=...rmse=...
Action 4:  evaluate_arch(node_id=n60, mode=proxy)
Observation 4:  Training done n60 (proxy):  auc=...acc=...rmse=...
```

D.3 Example 2: illegal write and repair

The following error message is the parse diagnostic returned by the runtime.

Example 2

```
Thought 1:  write a residual--concept interaction from scratch.
Action 1:  write_arch
```

Model-input example (invalid)

```
n3 = sub(n0, n1)
n4 = mul(n3, n2)
out = n4
```

Observation 1: Architecture parse failed: ... Rewrite in the exact format (first line 'Leaves: Stu(n0), Exer(n1), Concept(n2)', then one 'nX = op(...)' per line from n3, last line 'out = nX').

Thought 2: repair the missing Leaves header and keep consecutive node ids.

Action 2: write_arch

Model-input example (repaired)

```
Leaves: Stu(n0), Exer(n1), Concept(n2)
n3 = sub(n0, n1)
n4 = mul(n3, n2)
out = n4
```

Observation 2: Created architecture nX from the repaired program. Next, evaluate_arch(node_id=nX) to train & score it.

D.4 Searched Architectures

Figure 5 presents two representative programs selected during architecture search. Their executable encodings below show how the searched operators form the corresponding diagnostic functions.

Executable program n129

```
Dec: [(0,1,16), (3,2,12), (4,0,4), (5,0,18), (5,1,18),
      (6,7,16), (4,0,18), (4,1,18), (9,10,16), (8,11,25)]
Leaves: Stu(n0), Exer(n1), Concept(n2)
n3 = sub(n0, n1);  n4 = mul(n3, n2);  n5 = tanh(n4)
n6 = cosine(n5, n0);  n7 = cosine(n5, n1);  n8 = sub(n6, n7)
n9 = cosine(n4, n0);  n10 = cosine(n4, n1);  n11 = sub(n9, n10)
n12 = mix_gs(n8, n11);  out = n12
```

Executable program n82

```
Dec: [(0,1,16), (3,0,7), (4,2,12), (5,0,15), (4,0,15), (0,0,7),
      (8,2,12), (9,0,15), (0,1,18), (6,7,11), (12,10,11), (13,11,11)]
Leaves: Stu(n0), Exer(n1), Concept(n2)
n3 = sub(n0, n1);  n4 = sigmoid(n3);  n5 = mul(n4, n2);  n6 = mean(n5)
n7 = mean(n4);  n8 = sigmoid(n0);  n9 = mul(n8, n2);  n10 = mean(n9)
n11 = cosine(n0, n1);  n12 = add(n6, n7);  n13 = add(n12, n10)
n14 = add(n13, n11);  out = n14
```

E Controlled Study Protocol and Reporting

E.1 Interventions and Shared Conditions

The component comparisons report means over ten runs, using the same seed set as the main experiments. The LLM-based component variants share the complete method's GLM controller, budget thresholds, evaluator, root initialization, and final-confirmation rule; training time includes all evaluation stages. Data-loader settings and batch size are held fixed across the component variants. Individual ablations remove or replace one component, whereas Direct LLM Search removes all three proposed search modules.

Direct LLM Search lets the LLM propose and revise architectures through a plain ReAct loop using a basic architecture-evaluation interface. It does not use the CD-oriented construction and mutation tools, the MOG, or the constrained multi-fidelity search module.

Flat-History is a ReAct search with candidate scores and architectures in a flat history, without lineage-aware observations; the remaining tools are unchanged. Its mean of 83 evaluations results from the controller actively ending search through the termination action. Its mean training time is 5.75 hours, so earlier termination reduces evaluation count without necessarily reducing training time under the shared stopping thresholds across variants.

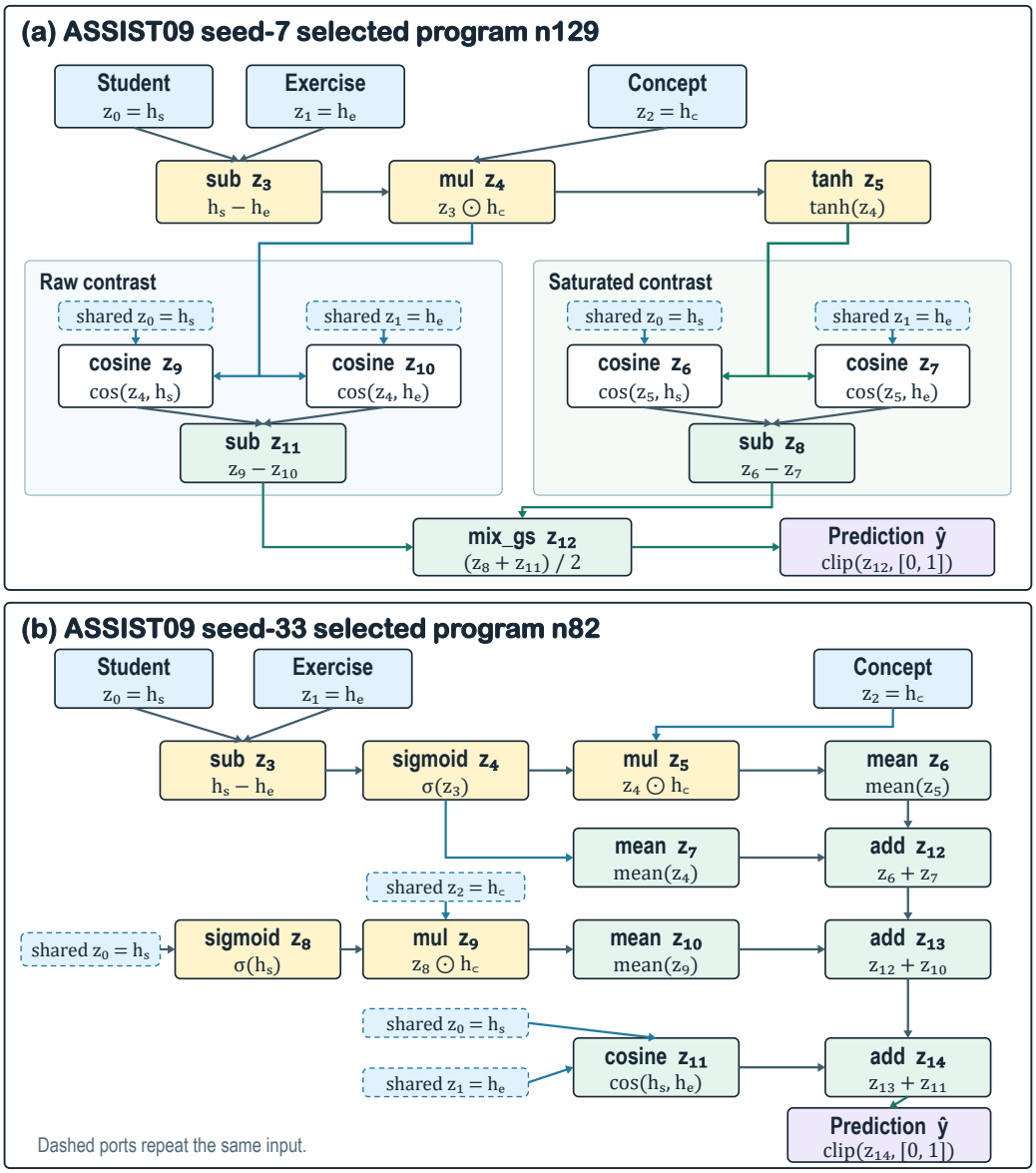

Figure 5: Examples of searched architectures. (a) ASSIST09 seed-7 case-study program `n129`. (b) ASSIST09 seed-33 controlled-study program `n82`. Cosine similarity uses the numerically stabilized definition in Table 10.

Write-Only removes `mutate_arch` while retaining `write_arch`. Its mean test scores appear in Table 3.

Fixed-Policy first evaluates a fresh proposed candidate in seeded random order. When no such candidate is available, it selects the best validation-scored parent and requests six neighbors, except at the periodic restart condition (step index divisible by ten), where it samples a legal architecture with two to eight operations. The controller configuration uses temperature 0.7, a 4,096-token response limit, a 100,000-character history allowance, and at most 12 architecture operations. Full evaluation follows EMO-NAS-CD's candidate-evaluation procedure and training budget; the 55-epoch Proxy recipe requires less computation than the 100-epoch Full recipe under the shared hardware and candidate-training configuration.

E.2 Execution Reliability Measures

For the execution reliability case study, we define the invalid-proposal rate and training-failure rate as follows:

$$R_{\text{invalid}} = \frac{\#\text{rejected candidate proposals}}{\#\text{all candidate proposals}}, \tag{7}$$

$$R_{\text{trainfail}} = \frac{\#\text{failed actual training attempts}}{\#\text{actual training attempts}}. \tag{8}$$

A zero denominator is treated as not applicable. Provider and infrastructure failures are distinguished from invalid proposals and candidate-training failures. In the seed-33 case study of four search variants, $R_{\text{invalid}} = 16/346 = 0.0462$ and $R_{\text{trainfail}} = 21/281 = 0.0747$.

The paired repair experiment uses the same 100 initially invalid cases under each of the ten seeds, with 20 cases each for malformed architecture syntax, unknown or disabled operators, invalid references, arity mismatch, and incompatible tensor operations. Each case is evaluated under detailed and generic feedback with the same starting context and at most three repair responses, using the same GLM controller across ten seeds. Repair success is the fraction of initial cases that become legal within this allowance. The case, not the individual retry, is the unit of comparison. The study reports paired outcomes and required calls, and does not launch candidate training merely to measure syntactic or execution-level repair.

Table 11 reports the mean paired diagnostic results over ten seeds. Across the ten seeds, detailed feedback repairs an average of 96/100 programs using 153 calls (1.53 per case), compared with 89/100 programs and 174 calls (1.74 per case) under generic feedback. The mean numbers of exact recoveries of the original encoding are 0 and 1 per seed under detailed and generic feedback, respectively. Both feedback conditions repair an average of 18/20 incompatible tensor cases; detailed feedback has higher mean success counts in the other four categories. The paired diagnostic measures recovery to legal programs, while the component comparison measures the selected model's predictive performance on held-out test data.

Table 11: Paired repair diagnostic without candidate training. Each cell gives the mean number of cases repaired within three `write_arch` responses over ten seeds, divided by the number of cases evaluated per seed (20 per category and 100 overall).

Initial failure type	Detailed feedback	Generic feedback
Malformed syntax	19/20	18/20
Unknown/disabled operator	19/20	18/20
Invalid reference	20/20	18/20
Arity mismatch	20/20	17/20
Incompatible tensor operations	18/20	18/20
All cases	96/100	89/100

F Runtime Accounting and Measurement Protocol

Runtime Coverage. Table 2 uses ten runs for each of the thirteen methods on ASSIST09 70/10/20, following Table 1, for a total of 130 runs. Runtime values average per-run measurements, with single-model training reported in minutes and search costs in GPU days or hours as indicated in Table 2. All methods are timed on a single NVIDIA GeForce RTX 3090 GPU, and the reported runtimes are averaged over ten seeds.

Timing Scope and Variability. Single-model runtimes include training, evaluation, and setup. GIRT, ORCDF, and SymbolicCDM are timed using the experimental configurations specified in their original papers. EMO-A7 and OSCD-AA1 report selected-model training time; the agent's 9.73-hour mean covers initialization, search, LLM interaction, and final confirmation. Table 3 reports candidate-training time including confirmation; end-to-end time additionally includes LLM interaction and other search overhead. The search methods use the same hardware, with single-model training and search costs reported separately. The reported means account for runtime variation across seeds under the same hardware and method-specific training settings.

Table 12 summarizes search resources on the three datasets across all four training ratios. One controller turn t returns a tool-call batch u_t, which can contain multiple evaluations; LLM calls also include retries, and cached evaluations do not launch training. The 120-evaluation and nine-hour limits are stopping thresholds rather than hard bounds on completed work, consistent with Eq. (2). Checks occur between tool actions or response batches, so evaluations and LLM calls already in progress can finish beyond a threshold; the next check stops further loop work. Final confirmation is outside these loop thresholds and remains included in total training and elapsed time.

Table 12: Search resources over all four training ratios. Coverage is runs with complete cost records / runs with predictive scores. Evaluation and LLM-call columns show ranges; elapsed time is the mean. Monetary costs are not recorded in the available artifacts.

Dataset	Coverage	Logged evals	LLM calls	Elapsed (h)
ASSIST09	12/40	79–122	40–66	9.62
XES-2k	12/40	97–122	39–71	8.37
MOOC-2k	11/40	43–111	14–55	9.34

In the seed-33 case study, the GLM controller uses 60 calls, 850,927 input tokens, and 37,787 output tokens. The provider reports these counts; recorded charges are unavailable, so no monetary cost is estimated. Matched token logs are unavailable for DeepSeek-V4-Pro and Claude-Sonnet-5.

Table 2 reports locally measured CDM training durations and architecture search costs. On ASSIST09, EMO-NAS-CD and OSCD have mean search costs of 15.2 GPU days and 26.1 hours, respectively, compared with 9.73 hours for Agent-NASCD. All three search costs are measured by rerunning the methods in the same experimental codebase with matched hardware and data partitions. The reported comparisons use the same processed ASSIST09 pool of 267,415 interactions, hardware, and batch size 256. The predictive comparisons use independently searched architectures in each dataset–ratio–seed setting, while the runtime comparison reports ten-run means on ASSIST09 at 70% training.

G Comparison of LLM Controllers

G.1 Experimental Settings and Results

The ASSIST09 70% controller comparison evaluates `deepseek-v4-pro` and `claude sonnet 5` with the same ten seeds and randomly generated roots as the default controller. The controller variants share the search space, initialization, budget limits, evaluation tools, and training configuration. Table 3 reports their mean results, with AUC values of 0.8228 and 0.8237 for DeepSeek-V4-Pro and Claude-Sonnet-5, respectively. The comparison changes the language model while retaining the experimental conditions used by the complete method.

Results are averaged over all ten seeds for each controller. Reported means and sample standard deviations use ten completed seeds per setting, with failed attempts excluded from the seed count. All controller variants share dimension 128, batch size 256, learning rate 10^{-3}, proxy/full epoch limits 55/100, and patience 12. The search loop has the same nine-hour stopping threshold and final-confirmation setting $K{=}10$ for each controller. Split seeds follow run seeds, and each controller uses the same candidate registry, validation partition, and final test protocol. Evaluation counts and elapsed time may differ because each controller chooses its own search actions and termination point within the shared limits on the same dataset.