Boost, Disentangle, and Customize: A Robust System2-to-System1 Pipeline for Code Generation

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated remarkable capabilities in various domains, particularly in system 1 tasks, yet the intricacies of their problem-solving mechanisms in system 2 tasks are not sufficiently explored. Recent research on System2-to-System1 methods surge, exploring the System 2 reasoning knowledge via inference-time computation and compressing the explored knowledge into System 1 process. In this paper, we focus on code generation, which is a representative System 2 task, and identify two primary challenges: (1) the complex hidden reasoning processes and (2) the heterogeneous data distributions that complicate the exploration and training of robust LLM solvers. To tackle these issues, we propose a novel BDC framework that explores insightful System 2 knowledge of LLMs using a MC-Tree-Of-Agents algorithm with mutual **B**oosting, **D**isentangles the heterogeneous training data for composable LoRA-experts, and obtain **C**ustomized problem solver for each data instance with an input-aware hypernetwork to weight over the LoRA-experts, offering effectiveness, flexibility, and robustness. This framework leverages multiple LLMs through mutual verification and boosting, integrated into a Monte-Carlo Tree Search process enhanced by reflection-based pruning and refinement. Additionally, we introduce the DisenLora algorithm, which clusters heterogeneous data to fine-tune LLMs into composable Lora experts, enabling the adaptive generation of customized problem solvers through an input-aware hypernetwork. Our contributions include the identification of critical challenges in existing methodologies, the development of the MC-Tree-of-Agents algorithm for insightful data collection, and the creation of a robust and flexible solution for code generation. This work lays the groundwork for advancing LLM capabilities in complex reasoning tasks, offering a novel System2-to-System1 solution.

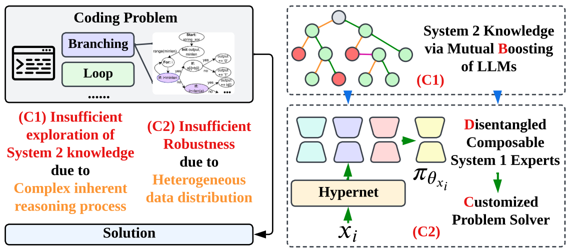

Figure 1: Illustration of the motivation.

1 Introduction

Large language models show significant intelligence in various domains, striking both the academic and industrial institutions. Despite their prominent problem-solving abilities in system 1 tasks, the mechanism behind the system 2 task solving procedure remain opaque. In this paper, we focus on the code generation task, which emerges as a captivating frontier (Zheng et al., 2023; Roziere et al., 2023; Shen et al., 2023), promising to revolutionize software development by enabling machines to write and optimize code with minimal human intervention. Recent research of llms for code focus on inference-time computation (System 2 methods) (Yang et al., 2024; Yao et al., 2024b; Zhang et al., 2023) and post-training. While during post-training, distilling system 2 knowledge into system 1 backbones is important and widely-used (Yu et al., 2024b).

However, the complex hidden reasoning process and the heterogeneous data distribution pose challenges to the existing System2-to-System1 pipeline. On one hand, the hidden reasoning process for code generation is complex and hard to explore **(C1)**. On the other hand, the heterogeneous data distribution, e.g., jumping structure like branching, recursion, etc., makes the existing train-once-for-all strategy hard to fit the complex latent patterns for robust and generalizable llm solvers **(C2)**.

For **(C1)**, we propose to disentangle the problem solving process into problem2thought and thought2solution stages, exploring the inherent reasoning clues via combining the strengths of multiple llms by mutually-verification and boosting. The exploration is integrated into a Monte-Carlo Tree Search process, where reflexion-based pruning and refinement are designed for more efficient and effective reasoning clues search.

For **(C2)**, we propose to disentangle the heterogeneous data into clusters, finetuning llms capable of different aspects of tasks to obtain the meta LoRA experts hub, and then adaptively generate customized problem solver for each code problem. Concretely, we design an input-aware hypernetwork to generate rank-wise weights over meta LoRA experts for customized problem solver, offering robustness and flexibility.

The main contributions of our work can be summarized below.

- **Identification of problems and novel BDC framework.** We identify the high-reasoning demand and heterogeneous latent patterns problems that hinders the performance of existing methods and propose a BDC framework that explores insightful inherent reasoning clues via multi-llms boosting, generates meta-LoRA experts via finetuning on disentangled data, and offer customized problem solver with an input-aware hypernet for rank-wise LoRA merging.

- **Novel MC-Tree-of-Agents algorithm for insightful data collection.** We disentangle the System 2 solving process into problem2thought and thought2solution stages, integrating the exploration process into a reflexion-based monte carlo tree search armed with pruning and refinement, enabling mutually verification and boosting of different agents for insightful data collection.

- **Novel DisenLoRA algorithm that offers customized problem solver for robust code generation.** We disentangle the heterogeneous data distribution into clusters on which meta-LoRA experts are trained, and design an input-aware hypernetwork to weight over the LoRA-experts for customized problem solver, offering robustness and flexibility.

2 Related Work

2.1 System 2 Methods in LLMs

Recent research on large language models for System 2 tasks focus on inference-time computation optimization to stimulate the inherent reasoning ability of LLMs. Few-shot learning methods (Wang et al., 2022; Madaan et al., 2022) utilize the in-context-learning ability of LLMs for enhanced generation. Retrieval-augmented generation (RAG) approaches (Nashid et al., 2023; Du et al., 2024) further introduce domain knowledge into LLMs. Techniques such as Chain-of-Thought (CoT) (Yang et al., 2024; Jiang et al., 2024; Li et al., 2023), Tree-of-Thought (ToT) (Yao et al., 2024b; La Rosa et al., 2024), and Monte Carlo Tree Search (MCTS) (Li et al., 2024; Zhang et al., 2023; Hu et al., 2024; Hao et al., 2023; Feng et al., 2024b) are used to explore the inherent reasoning process, often based on the self-play mechanism to reflect on previously generated contents to learn from itself (Haluptzok et al., 2022; Chen et al., 2023a; Lu et al., 2023; Chen et al., 2023b; Madaan et al., 2024; Shinn et al., 2024). During inference, error position can be beneficial in improving the reliability and performance of the model. With identification and analysis of where and why errors occur, recent research (Yao et al., 2024a; Luo et al., 2024; Wu et al., 2025) has made significant strides in quantifying and mitigating errors during model inference. Refinement (Madaan et al., 2024; Gou et al., 2023) and reflexion (Shinn et al., 2024; Lee et al., 2025) are also powerful techniques for enhancing the inference capabilities of LLMs, usually by enabling iterative improvement and self-correction.

2.2 Model Composition

Model composition technique gains notable attention in cross-tasks generalization. Traditional methods for multiple tasks are to train models on a mixture of datasets of different skills (Caruana, 1997; Chen et al., 2018), with the high cost of data mixing and lack of scalability of the model though. Model merging is a possible solution to this. Linear merging is a classic merging method that consists of simply averaging the model weights (Izmailov et al., 2018; Smith and Gashler, 2017). Furthermore, Task Arithmetic (Ilharco et al., 2022) computes task vectors for each model, merges them linearly, and then adds back to the base, and SLERP (White, 2016) spherically interpolates the parameters of two models. Based on Task Arithmetic framework, TIES (Yadav et al., 2024) specifies the task vectors and applies a sign consensus algorithm to resolve interference between models, and DARE (Yu et al., 2024a) matches the performance of original models by random pruning.

Recently, LoRA merging methods are also widely applied to cross-task generalization. CAT (Prabhakar et al., 2024) introduces learnable linear concatenation of the LoRA layers, and Mixture of Experts(MoE) (Buehler and Buehler, 2024; Feng et al., 2024a) method has input-dependent merging coefficients. Other linear merging methods of LoRAs, such as LoRA Hub (Huang et al., 2023), involve additional cross-terms compared to simple concatenation.

3 Preliminaries

3.1 Monte-Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a decision-making algorithm widely used in environments with large state and action spaces, particularly in game AI and planning. It incrementally builds search trees to estimate optimal actions by simulating random plays from various nodes and gradually improving action-value estimates based on simulation outcomes. Over iterations, this approach gradually converges to near-optimal decision-making policies. Notably, its integration with reinforcement learning has driven breakthroughs in systems like AlphaGo and AlphaZero (Silver et al., 2017), achieving superhuman performance in games.

Classical MCTS consists of four stages: selection, expansion, simulation, and backpropagation. It typically employs Upper Confidence Bounds for Trees (UCT) (Kocsis and Szepesvári, 2006), which balances exploration and exploitation by guiding the search to promising nodes. After simulation, results propagate back through the tree, updating node values. However, MCTS struggles in domains with large action spaces, where excessive branching can degrade performance. Progressive Widening and Double Progressive Widening techniques have been proposed to mitigate this by dynamically limiting the number of actions considered at each decision node (Coulom, 2006).

3.2 LoRA Finetuning

LoRA (Low-Rank Adaptation) (Hu et al., 2021) fine-tuning is a technique used to adapt large pre-trained models, such as transformers, to specific tasks with minimal computational overhead. The key idea behind LoRA is to introduce low-rank matrices into the model's weight updates, which reduces the number of trainable parameters and makes fine-tuning more efficient.

LoRA starts with a model that has been trained on a large dataset. During finetuning, instead of updating the full weight matrix $W \in \mathbb{R}^{m\times n}$, LoRA introduces two low-rank matrices $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$, where $r \ll \min(m, n)$. The updated weight matrix W' is then given by:

$$W' = W + \Delta W = W + A \cdot B. \tag{1}$$

During fine-tuning, only the matrices A and B are updated, while the original weight matrix W remains frozen. This reduces the number of trainable parameters from $m \times n$ to $m \times r + r \times n$, which is much smaller when r is small. For a given task with loss function $\mathcal{L}$, the objective is to minimize:

$$\mathcal{L}(y, f(x; W + A \cdot B)), \tag{2}$$

where y is the target output, x is the input, and f is the model's forward function.

By introducing low-rank matrices, both the number of trainable parameters and memory footprint are reduced. This approach is particularly useful in scenarios where computational resources are limited or when fine-tuning needs to be done quickly.

4 Methodology

In this section, we introduce the overall methodology of BDC, addressing challenges in the System2-to-System1 pipeline for code generation, specifically the complexity of hidden reasoning processes and heterogeneous data distributions. The proposed BDC pipeline consists of three main stages: 1) explore the System 2 knowledge via mutual verification and boosting between LLMs; 2) disentangle the obtained data into clusters over which composible LoRA experts are tuned; 3) customize problem solver by weighting over the composable LoRA experts using an input-aware hypernetwork.

4.1 System 2 Knowledge Exploration

In this subsection, we introduce the mechanism design for the data collection process. Due to the complex reasoning nature embodied, code blocks are hard to evaluate and estimate before mature. Reliable reward signals of a reasoning path therefore mainly depend on the dynamic compilation and execution feedbacks, which are extremely sparse and require extensive simulations. To simplify the

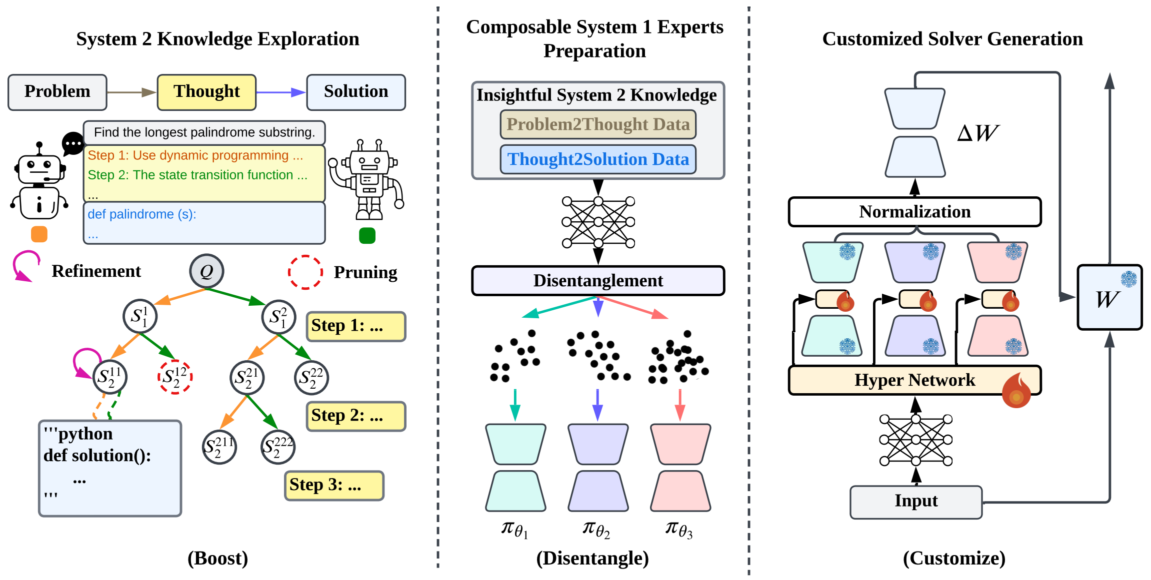

Figure 2: Illustration of the overall framework of BDC.

generation paradigm and exploit the mutual verification capabilities of the collective searching, we decompose the generation process into two distinct stages: problem-to-thought and thought-to-solution.

4.1.1 Problem-to-thought

Traditional Monto-Carlo Tree Searching comprises three key operations in each iteration: (a) Select, (b) Expand, (c) Backup. In the problem-to-thought stage, we further extend MCTS by two distinct operations (d) Prune, and (e) Refine to reduce the searching space. We elaborate on these operations as follows.

Select. Starting from the root, the reasoning path is prolonged by iteratively adding a specific child of the latest node. The operation is usually governed by certain policies, among which we adopt Probability-weighted Upper Confidence Bound(P-UCB) to balance the exploration and exploitation:

$$\mathrm{PUCB}(S_c) = Q(S) + c \cdot P(a|S_p) \cdot \frac{\sqrt{\log N(S)}}{1 + N(S_c)}, \tag{3}$$

where S_c is the state of the child node. S and $Q(S)$ denote the parent node's state and value. $P(a|S)$ is the conditional probability of sampling the sequence a. $N(S)$ is the total number of times the parent node S has been visited during simulations, while $N(S_c)$ tracks visits to the child node S_c. The selection process will stop if either a semantic or rule-based(e.g. length limits) terminal state encounters.

Expand. The Expand operation is triggered if a non-terminal leaf node of the tree is selected. A set of predefined LLM polices $\pi_0, \cdots, \pi_n$ generate subsequent thought sequences a_{in} given the state S of the current node, forming new leaf nodes:

$$\forall i \in [n], P(a_i|S) \sim \pi_i(|S). \tag{4}$$

Backup. For well-defined problems, a reasoning path S_t will eventually end at a terminal leaf node S_T by iterating the Select and Expand operations. The reward r_T is set according to the evaluation. We will skip the definition of reward r_t and passrate $PR(S_t)$, which will be detailed in the explanation of the Simulate operation. The reward value is back-propagated along the reasoning path to update the state values of corresponding ancestor nodes:

$$Q(S_{t-1}) = f(Q(S_t), r_t + \gamma PR(S_t)), \tag{5}$$

where f is the value function.

Additionally, the visit counts of ancestors are updated alongside the reasoning path:

$$N(S_t) = N(S_t) + 1. \tag{6}$$

We further extend and formalize reflective reason settings proposed in CoMCTS into Prune and Refine operations as shown in Figure 3.

Pruning. The pruning operation on a selected node will examine and compare its passrate with

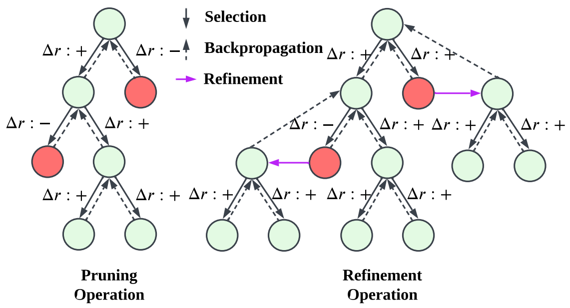

Figure 3: Pruning and refinement operations.

that of its parent. With powerful LLMs, we consider valid and reasonable thoughts to bring non-negative influence solution seeking, thus featuring monotonically non-decreasing values in the passrate $PR(S_t) <= PR(S_{t+1})$.

A child node alleviating this principle will be considered as an ill node that introduces wrong thoughts. The ill node will be removed and trigger an instant Backup operation with zero reward to downweight its ancestors.

Refine. The truncated error and state information left by ill nodes will be analyzed in the Refine operations. To mitigate the bootstrapping bias of LLMs, a distinct policy LLM will be adopted to infer and summarize the information in natural language, which will be later utilized to refine and replace the ill nodes:

$$\begin{aligned} &isIll(S^{\pi_i}) == 1, \\ &Summary(S^{\pi_i}) \sim \pi_j(Q(S^{\pi_i}), \\ &\qquad S^{\pi_i}, BlockAnalysis(S^{\pi_i})), \end{aligned} \tag{7}$$

where S^{π_i} denotes a ill node generated by π_i. A refined node is generated to replace the ill one:

$$a' \sim \pi_i(Q(S^{\pi_i}), Summary). \tag{8}$$

We enforce global and local constraints on possible times of calling Refine operation to avoid infinite loops and balance performance with compute budgets. A successful Refine operation will cause an in-place replacement of the ill-node, triggering another Backup operation to re-weight its ancestors.

4.1.2 Thought-to-solution

Simulate. For the thought-to-solution, we repurpose the Simulate operation for the collective solution generation process from the given state S. The operation will produce a set of possible solutions, each from a policy LLM:

$$Solut.(S)_i \sim \pi_i(S). \tag{9}$$

We define the passrate of a state as the average passrate of its corresponding solutions:

$$PR(S) = \frac{1}{n}\sum_i^n Passed(Solut.(S)_i), \tag{10}$$

where $Passed(\cdot)$ represents the supervising signal from dynamic compilation and execution feedback.

The node's value $Q(S_t)$ is determined by its $PR(S_t)$ and reward r_t. Sincere additional solution string will be appended to a non-terminal state S_t before evaluation, $PR(S_t)$ is an indirect supervising signal for the S_t, and the direct signal r_t is set to zero.

The terminal state S_T is treated as the unique solution itself since no string concatenation applies, therefore featuring a non-trivial reward r_T. Putting everything together, we have:

$$Q(S) = \begin{cases} r_T & \text{if terminal,} \\ \gamma PR(S_t) & \text{otherwise.} \end{cases} \tag{11}$$

4.2 System2-to-System1 Training

4.2.1 Heterogeneous Distribution Disentanglement

After the data collection, the resulting training data obtained from the MC-Tree-Of-Agents process consists of problem2thought data $D^{p2t} = \{\langle X_i^{p2t}, y_i^{p2t}\rangle | i \in \mathbf{P}\}$ and thought2solution data $D^{t2s} = \{\langle X_i^{t2s}, y_i^{t2s}\rangle | i \in \mathbf{P}\}$: $D_{train} = \{D^{p2t}, D^{t2s}\}$. As discussed in the introduction section, the latent patterns of coding problems are complex and tend to be heterogeneously distributed, e.g., the branching and recursion flow existing in the code blocks, different strategies of algorithm solutions, etc. Therefore, we disentangle the training data based on the latent semantics of the data into different clusters for fine-grained modeling.

The clustering objective can be summarized as below:

$$minimize_{\mathcal{C}} \sum_k \sum_{i \in C_k} cosine(e_i, \mu_k), \tag{12}$$

$$\begin{aligned} e_i =& \Phi_\theta(\langle X_i, y_i\rangle), \\ \mu_k =& mean\{e_i | i \in C_k\}, \end{aligned}$$

where Φ_{theta} is the encoder of a code llm and μ_k denotes the centroid of cluster C_k.

4.2.2 Composable LoRA Experts Preparation

Having obtained the disentangled data clusters, we then finetune on them to obtain the meta LoRA experts for specialized experts of different aspects.

$$\begin{aligned} &\forall C_k \in \mathcal{C}, \\ &\pi_{\theta_k} \leftarrow SFT(\pi_\theta, \{\langle X_i, y_i \rangle | i \in C_k\}), \end{aligned} \tag{13}$$

where π_θ denotes the base LLM and π_{θ_k} denotes the parameters of the LoRA adapter obtained by finetuning π_θ on C_k.

4.2.3 Input-Aware Hypernetwork for Customized Solver

Given specialized LoRA experts $\pi_{\theta_1}, \cdots, \pi_{\theta_K}$ trained on distinct data clusters, we design an input-aware Hypernetwork $f(\cdot)$ to dynamically compose these experts through rank-wise adaption for customized problem solver.

For each input instance, the hypernetwork generates customized expert weights digesting its encoding and semantic distances to the cluster centroids. we identify "rank" as the minimal unit for aggregation and generate rank-wise weights for different experts at each decoding layer:

$$G_i \leftarrow f(e_i, \langle cosine(e_i, \mu_1), \ldots, cosine(e_i, \mu_K) \rangle), \tag{14}$$

where e_i is the encoding of input X_i, $G_i \in R^{K \times r \times 1}$ is the output weight matrix, r is the rank of the LoRA matrices, and K is the number of LoRA experts.

The aggregated ΔW of the linear projection layer is then obtained by

$$\mathbf{A}^* = [A_1, \ldots, A_K] \odot G_i, \tag{15}$$

$$\mathbf{\Delta W}^* = [B_1 A_1^*, \ldots, B_K A_K^*], \tag{16}$$

$$\Delta W = ReduceSum(\mathbf{\Delta W}^*). \tag{17}$$

The projection output of ΔW is then merged during forwarding via:

$$y = W_0 x + \Delta W x. \tag{18}$$

We adopt a dedicated training phase for the Hypernetwork where all parameters are frozen except for the $f(\cdot)$. The training is supervised by the cross-entropy loss, with the randomly permuted input-output pairs from D_{train}.

5 Experiments

We conduct empirical studies starting from the following research questions.

RQ1 Does the proposed data collection algorithm explore insightful reasoning knowledge?

RQ2 Do the complex latent patterns of reasoning data impact the training performance?

RQ3 Can the disentangle-and-compose mechanism help to promote performance?

RQ4 Do the proposed input-aware hypernet work outperform other model composition techniques?

RQ5 How does DisenLoRA perform on untrained datasets?

5.1 Setup

In this section, we provide detailed setup information for the evaluation of the proposed BDC, including datasets, trajectory data collection, and competing methods.

The overall evaluation is conducted on two benchmark datasets: the competition-style APPS dataset and the CodeContest dataset. Both datasets categorize problems from easy to hard. We randomly sample problem subsets from each category of these two datasets. Each subset contains approximately 100 problems, except for the CodeContest-Hard category, which consists of around 50 problems due to inherent limitation in size.

We conduct isolated assessments of both stages of BDC to ensure a comprehensive comparison.

Data collection. For Python code generation, we compare the performance of MCTS over different methods: zeroshot, LDB (Zhong et al., 2024), RAP (Hao et al., 2023), Reflexion, LATS (Zhou et al., 2023), ToT and RethinkMCTS (Li et al., 2024). To mitigate the influence of factors such as context window limitations and instruction-following capabilities, we employ two advanced base models: GPT-4o-mini and Claude-3.5-Sonnet. Aligned with the purpose, we adopt a greedy decoding strategy for both models. Additionally, we provided peer comparisons between these two base models when driven by the MC-Tree-Of-Agents method in terms of their error position and refinement capability.

Fine-tuning. For fine-tuning, BDC is compared against several alternative methods, including SFT on clustered subsets, TIES, DARES, and TWINS (Liu et al., 2023).

Table 1: Main results on System 2 knowledge exploration.

Models	APPS						CodeContest			
	Intro.		Inter.		Comp.		Easy		Hard	
	PR	AC	PR	AC	PR	AC	PR	AC	PR	AC
ZeroShot	56.56	35.00	40.57	19.00	23.67	9.00	29.03	19.61	28.24	19.23
LDB	60.64	40.00	46.78	22.00	21.00	8.00	34.76	25.58	33.52	16.28
RAP	64.24	39.00	43.32	14.00	22.83	8.00	43.08	33.33	39.99	26.92
Reflexion	60.65	40.00	45.58	21.00	17.50	7.00	56.16	47.83	34.09	21.15
LATS	69.46	50.00	45.86	20.00	21.83	7.00	57.70	47.83	39.10	30.77
ToT	74.34	55.00	63.49	33.00	26.30	11.00	51.89	41.18	49.07	32.69
RethinkMCTS	76.60	59.00	74.35	49.00	42.50	28.00	60.84	51.53	55.79	48.04
Single (GPT4omini)	77.99	60.00	72.89	50.00	44.17	25.00	55.79	48.04	45.72	26.92
Single (Claude)	73.80	61.00	73.60	57.00	54.67	42.00	58.75	53.92	68.41	55.76
MC-Tree-Of-Agents	79.72	64.00	79.42	63.00	59.17	45.00	62.49	54.64	70.49	56.41
+ Pruning	85.18	76.00	81.95	67.00	54.00	38.00	64.62	59.80	73.12	59.62
+ Refine	81.29	68.00	78.85	62.00	60.33	44.00	63.23	56.86	73.80	63.46

Table 2: Main results on System2-to-System1 tuning.

Meta-llama-3.1-instruct-8b														
Finetune Method	Intro. (100)		Inter. (100)		Comp. (100)		Overall		Easy (102)		Hard (51)		Overall	
	PR	AC	PR	AC	PR	AC	PR	AC	PR	AC	PR	AC	PR	AC
w/o tuning	21.14	4.00	20.72	4.00	12.83	1.00	18.23	3.00	25.54	17.65	15.46	5.77	22.18	13.69
SFT on all	22.55	7.00	26.40	3.00	10.67	1.00	19.87	3.67	25.33	17.65	16.73	7.69	22.46	14.33
SFT on cluster 0	20.67	6.00	24.23	3.00	11.50	1.00	18.80	3.33	27.31	17.65	11.69	1.92	22.10	12.41
SFT on cluster 1	21.22	4.00	20.69	4.00	12.00	2.00	17.97	3.33	27.78	20.59	18.12	9.62	24.56	16.93
SFT on cluster 2	16.65	7.00	23.97	3.00	17.33	4.00	19.32	4.67	26.82	20.59	19.50	9.62	24.38	16.93
Ties	22.75	4.00	23.06	4.00	12.67	4.00	19.49	4.00	26.64	21.57	18.71	9.62	24.00	17.59
Dare	24.97	7.00	26.66	5.00	12.50	3.00	21.38	5.00	23.05	13.73	19.65	15.38	21.92	14.28
Twin	19.10	5.00	23.85	5.00	8.50	1.00	17.15	3.67	26.87	17.64	12.92	9.62	22.22	14.97
DisenLoRA	**27.11**	**9.00**	23.11	3.00	11.50	**4.00**	**20.57**	**5.33**	**32.24**	**22.55**	19.43	9.62	**27.97**	**18.24**

5.2 Empirical Analysis and Discussion

5.2.1 RQ1. MC-Tree-Of-Agents

We evaluate MC-Tree-Of-Agents against widely-used baseline methods, the results are summarized in Table 1. From the results, we can draw the following conclusions.

- The proposed MC-Tree-Of-Agents outperforms all the baseline methods, which effectively explores the insightful System 2 knowledge.

- Comparing with the single LLM as agents version, MC-Tree-Of-Agents allows for mutual verification and boosting between different LLMs, offering a superior performance over each distinct-LLM-as-agent method. This showcases the effectiveness of the interaction between LLMs of different wisdom.

- The pruning and refinement operations both contribute to the final performance, offering a notable accuracy gain. This validates that the designed pruning and refinement mechanism, based on the difference between rewards of parent-child nodes, can save the algorithm from erroneous exploration and lead to beneficial directions in limited rollouts.

5.2.2 RQ2. Impact of latent patterns

To study the distribution of the latent patterns of coding problems, we first conduct the T-SNE visualization on the encodings of reasoning data collected by MC-Tree-Of-Agents on APPS dataset. The visualization is displayed in Figure 4.

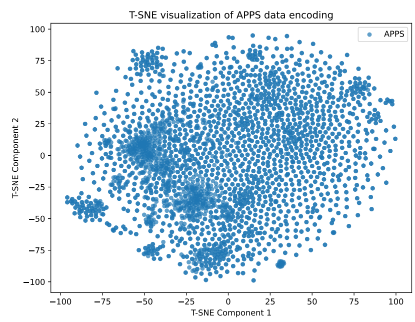

Figure 4: T-sne visualization of the APPS data encoding.

From the visualization, we can see that there different clusters of data distributions existing in the latent reasoning semantic space, which poses a potential challenge to robust and generalizable LLMs on code.

Furthermore, we perform finetuning on different clusters of data obtained in Section 4.2.1 and evaluate the corresponding models on the test data. From the results in Table 2, we can see the following conclusions. 1) LLMs finetuning on all the clusters can offer better performance than that of the non-tuning version, validating the quality of the collected System2 knowledge data. 2) Llm experts obtained from different clusters show different performances on different levels of tasks. One expert can demonstrate outstanding capability on one level of tasks, even outperforming the LLM finetuning on all the data, while performing weakly on a different level of task. This phenomenon further justifies the heterogeneous latent patterns of data distribution and serves as supportive evidence for disentangling LLM experts.

5.2.3 RQ3. Effectiveness of the Experts Composition

During the empirical study, we test different model merging methods that combine wisdom from different LoRA experts. We evaluate the well-known Ties, Dare, and the recently proposed TWIN merging methods. All of them yield a static composed model that takes in the strength of the candidate experts to be merged via solving parameter interference. From the results, we can see that merging over decomposed LoRA-experts can offer more robust problem solvers, outperforming the simple train-once-for-all mechanism. The experiments justify our major rationale that disentanglement-and-compose pipeline can offer more robust System2-to-System1 performance.

5.2.4 RQ4. Superiority of DisenLoRA over other composition methods

Although the static-composed expert model can promote robustness to some extent, its static nature lacks flexibility to different styles of inputs. As discussed in the previous contents, the data distribution of coding problems is complex, making the one-fits-all mechanism easy to fail. Therefore, we design DisenLoRA algorithm to yield a customized problem solver with input-awareness. From the results, we can see that DisenLoRA outperforms the competing merging methods, validating the effectiveness of the proposed input-aware hypernetwork that dynamically aggregates the candidate composable LoRA experts at a rank-wise level.

5.2.5 RQ5. Discussion of the Cross-Dataset Generalization of DisenLoRA

Despite the flexibility offered by the input-aware hypernetwork, its performance may degrade on new datasets where the hypernetwork is not trained. To study this scenario, we use the model trained on APPS to generate solutions for CodeContest and use the model trained on CodeContest to generate solutions for APPS. The results are displayed in Table 3.

Table 3: Cross-dataset generalization test.

OOD Dataset	APPS		CodeContest	
Method	PR	AC	PR	AC
w/o tuning	18.23	3.00	22.18	13.69
w/ SFT	17.44	4.33	20.99	14.29
DisenLoRA	18.25	4.33	25.09	14.34

From the results, we can see that the proposed DisenLoRA has the generalization ability to the untrained dataset, outperforming the train-once-for-all mechanism still. This demonstrates that the parameters of the trained hypernetwork have the awareness of semantic similarities across datasets.

6 Conclusion

We identify the complexity of inherent reasoning exploration and the heterogeneous data distribution problems that hinder the performance of System2-to-System1 methods. Correspondingly, we propose the BDC pipeline that explores insightful System2 knowledge via mutually **B**oosting between llm agents, **D**isentangle heterogeneous data distribution for composable LoRA experts, and **C**ustomize problem solver for each instance, offering flexibility and robustness. Correspondingly, we propose the MC-Tree-Of-Agents algorithm to efficiently and effectively explore the insightful System2 knowledge via mutual verification and boosting of different LLM agents, armed with reward-guided pruning and refinement to explore more beneficial states in limited rollouts for better performance. Additionally, we design an input-aware hypernetwork to aggregate over the disentangled composable LoRA experts trained on different clusters of data collected from MC-Tree-Of-Agents. This mechanism offers a customized problem solver for each data instance. Various experiments and discussions validate the effectiveness of different model components.

Limitations

While our work presents an efficient pipeline for transferring specialized knowledge from collective system-2-like LLMs to locally deployed language models through multiple LoRA adapters—enabling rapid, precise, system-1-like reasoning—three limitations merit discussion. First, despite code generation serving as an effective proxy for complex reasoning, our evaluation is restricted to this domain, leaving open questions about generalizability to broader textual reasoning tasks (e.g., commonsense reasoning and semantic parsing). Second, while we focus on their performance on the specific benchmarks, the safety alignment of derived models remains unaddressed. Systematic evaluation is required to assess whether our distilled experts preserve human values and mitigate harmful outputs. Finally, our ensemble methodology for LoRA experts, while input-aware, does not fully exploit potential sparsity optimizations in parameter activation, leaving room for computational efficiency improvements through advanced routing mechanisms.

References

Eric L Buehler and Markus J Buehler. 2024. X-lora: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design. *APL Machine Learning*, 2(2).

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28:41–75.

Kai Chen, Chunwei Wang, Kuo Yang, Jianhua Han, Lanqing Hong, Fei Mi, Hang Xu, Zhengying Liu, Wenyong Huang, Zhenguo Li, et al. 2023a. Gaining wisdom from setbacks: Aligning large language models via mistake analysis. *arXiv preprint arXiv:2310.10477*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023b. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR.

Rémi Coulom. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer.

Kounianhua Du, Jizheng Chen, Renting Rui, Huacan Chai, Lingyue Fu, Wei Xia, Yasheng Wang, Ruiming Tang, Yong Yu, and Weinan Zhang. 2024. Codegrag: Bridging the gap between natural language and programming language via graphical retrieval augmented generation. *Preprint*, arXiv:2405.02355.

Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. 2024a. Mixture-of-loras: An efficient multitask tuning for large language models. *arXiv preprint arXiv:2403.03432*.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024b. Alphazero-like tree-search can guide large language model decoding and training. *Preprint*, arXiv:2309.17179.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. 2022. Language models can teach themselves to program better. *arXiv preprint arXiv:2207.14502*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhiyuan Hu, Chumin Liu, Xidong Feng, Yilun Zhao, See-Kiong Ng, Anh Tuan Luu, Junxian He, Pang Wei Koh, and Bryan Hooi. 2024. Uncertainty of thoughts: Uncertainty-aware planning enhances information seeking in large language models. *arXiv preprint arXiv:2402.03271*.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2023. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293. Springer.

Ricardo La Rosa, Corey Hulse, and Bangdi Liu. 2024. Can github issues be solved with tree of thoughts? *arXiv preprint arXiv:2405.13057*.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. 2025. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*.

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2023. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology*.

Qingyao Li, Wei Xia, Kounianhua Du, Xinyi Dai, Ruiming Tang, Yasheng Wang, Yong Yu, and Weinan Zhang. 2024. Rethinkmcts: Refining erroneous thoughts in monte carlo tree search for code generation. *arXiv preprint arXiv:2409.09584*.

Ziquan Liu, Yi Xu, Xiangyang Ji, and Antoni B Chan. 2023. Twins: A fine-tuning framework for improved transferability of adversarial robustness and generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16436–16446.

Jianqiao Lu, Wanjun Zhong, Wenyong Huang, Yufei Wang, Fei Mi, Baojun Wang, Weichao Wang, Lifeng Shang, and Qun Liu. 2023. Self: Language-driven self-evolution for large language model. *arXiv preprint arXiv:2310.00533*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*.

Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-based prompt selection for code-related few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2450–2462. IEEE.

Akshara Prabhakar, Yuanzhi Li, Karthik Narasimhan, Sham Kakade, Eran Malach, and Samy Jelassi. 2024. Lora soups: Merging loras for practical skill composition tasks. *arXiv preprint arXiv:2410.13025*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Bo Shen, Jiaxin Zhang, Taihong Chen, Daoguang Zan, Bing Geng, An Fu, Muhan Zeng, Ailun Yu, Jichuan Ji, Jingyang Zhao, et al. 2023. Pangu-coder2: Boosting large language models for code with ranking feedback. *arXiv preprint arXiv:2307.14936*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

David Silver, Julian Schrittwieser, Karen Simonyan, et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550:354–359.

Joshua Smith and Michael Gashler. 2017. An investigation of how neural networks learn from the experiences of peers through periodic weight averaging. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 731–736. IEEE.

Xingyao Wang, Sha Li, and Heng Ji. 2022. Code4struct: Code generation for few-shot event structure prediction. *arXiv preprint arXiv:2210.12810*.

Tom White. 2016. Sampling generative networks. *arXiv preprint arXiv:1609.04468*.

Junxi Wu, Dongjian Hu, Yajie Bao, Shu-Tao Xia, and Changliang Zou. 2025. Error-quantified conformal inference for time series. *arXiv preprint arXiv:2502.00818*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2024. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36.

Guang Yang, Yu Zhou, Xiang Chen, Xiangyu Zhang, Terry Yue Zhuo, and Taolue Chen. 2024. Chain-of-thought in neural code generation: From and for lightweight language models. *IEEE Transactions on Software Engineering*.

Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, et al. 2024a. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search. *arXiv preprint arXiv:2412.18319*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024b. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024a. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*.

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. 2024b. Distilling system 2 into system 1. *Preprint*, arXiv:2407.06023.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*.

Li Zhong, Zilong Wang, and Jingbo Shang. 2024. Ldb: A large language model debugger via verifying runtime execution step-by-step. *arXiv preprint arXiv:2402.16906*.

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

A Implementation Details

For the size of retrieval pool, we use 11,913 C++ code snippets and 2,359 python code snippets. Due to the limited access, we do not use a large retrieval corpus for our experiment, which can be enlarged by other people for better performance. We also attach the graph extraction codes for both languages and all other expeirment codes here: https://anonymous.4open.science/r/Code-5970/

For the fintuning details, the learning rate and weight decay for the expert GNN training is 0.001 and 1e-5, repectively. We apply 8-bit quantization and use LoRA for parameter-efficient fine-tuning. The rank of the low-rank matrices in LoRA is uniformly set to 8, alpha set to 16, and dropout is set to 0.05. The LoRA modules are uniformly applied to the Q and V parameter matrices of the attention modules in each layer of the LLM. All the three models are optimized using the AdamW optimizer. For the CodeContest dataset, totally 10609 datapoints are used, and for APPS dataset, 8691 data samples are used to train the model.