Language Models as Continuous Self-Evolving Data Engineers

Peidong Wang[1], **Ming Wang**[1], **Zhiming Ma**[2],
Xiaocui Yang[1], **Shi Feng**[1†], **Daling Wang**[1], **Yifei Zhang**[1], **Kaisong Song**[1,3]
[1]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[2]China Mobile Internet Company Limited, Guangzhou, China
[3]Alibaba Group, Hangzhou, China
`pdongwang@163.com` `sci.m.wang@gmail.com`
`mazhiming312@outlook.com` `kaisong.sks@alibaba-inc.com`
`{yangxiaocui, fengshi, wangdaling, zhangyifei}@cse.neu.edu.cn`

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities, yet their further evolution is often hampered by the scarcity of high-quality training data and the heavy reliance of traditional methods on expert-labeled data. This reliance sets a ceiling on LLM performance and is particularly challenging in low data resource scenarios where extensive supervision is unavailable. To address this issue, we propose a novel paradigm named `LANCE` (**LAN**guage models as **C**ontinuous self-**E**volving data engineers) that enables LLMs to train themselves by autonomously generating, cleaning, reviewing, and annotating data with preference information. Our approach demonstrates that LLMs can serve as continuous self-evolving data engineers, significantly reducing the time and cost of post-training data construction. Through iterative fine-tuning on Qwen2 series models, we validate the effectiveness of `LANCE` across various tasks, showing that it can maintain high-quality data generation and continuously improve model performance. Across multiple benchmark dimensions, `LANCE` results in an average score enhancement of **3.64** for Qwen2-7B and **1.75** for Qwen2-7B-Instruct. This autonomous data construction paradigm not only lessens reliance on human experts or external models but also ensures data aligns with human preferences, offering a scalable path for LLM self-improvement, especially in contexts with limited supervisory data. Code is available at: `https://github.com/Control-derek/LANCE`.

1 Introduction

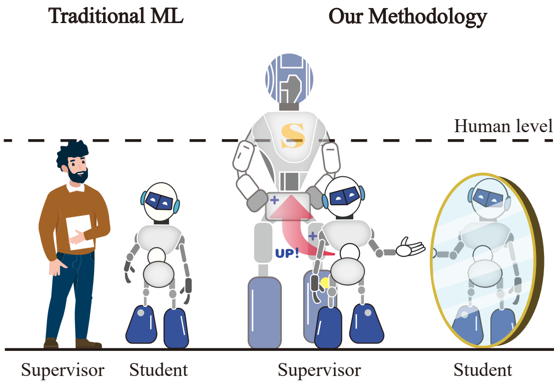


Figure 1: **An illustration of our methodology.** Traditional ML typically involves humans supervising weaker models. Our methodology explores model self-supervision, potentially advancing more autonomous and capable AI systems.

Large Language Models (LLMs) have exhibited extraordinary proficiency in tackling diverse tasks, such as natural language understanding, logical reasoning, code generation, and mathematical reasoning (Dubey et al., 2024; Achiam et al., 2023; Liu et al., 2024). These advancements are largely attributed to instruction tuning (Wei et al., 2022) and Reinforcement Learning from Human Feedback (RLHF) (Stiennon et al., 2020), and these approaches have significantly improved the performance of LLMs using quality data. High-quality data not only enhances the precision and reliability of models but also ensures that the outputs are better aligned with human values and preferences (Wang et al., 2023b).

However, with the rapid development of LLMs, the acquisition of high-quality data becomes increasingly challenging (Penedo et al., 2023). LLM training is rapidly depleting high-quality data sources, with projections from Villalobos et al. (2024) indicating an exhaustion of public human-generated text data between 2026 and 2032, necessitating new high-quality data for continued LLM advancement. Furthermore, reliance on human experts for data curation is time-consuming and costly (Villalobos et al., 2024). As models advance towards capabilities potentially extending beyond current human expertise (Burns et al., 2024), human supervisory signals might diminish in utility,

---

[†] Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 18097–18116
November 4-9, 2025 ©2025 Association for Computational Linguistics

and human-generated data quality could become a bottleneck (Burns et al., 2024). Existing solutions for these data challenges include synthetic data generation via teacher LLMs (Lee et al., 2024; Dai et al., 2025; Chen et al., 2024a), which depend on external supervision. Other approaches advocate for self-tuning with LLM-constructed data (Zelikman et al., 2022; Wang et al., 2023a; Gulcehre et al., 2023), but these often lack comprehensive coverage of the entire post-training data lifecycle (Dubey et al., 2024).

To address these challenges, we propose `LANCE`, a novel training paradigm empowering LLMs to autonomously generate, clean, review, and annotate data with preference information for self-training. This approach exemplifies how language models can operate as continuous self-evolving data engineers, minimizing human intervention and external resources. The process begins with the model reviewing the seed dataset. It then generates new instruction data to cover distributional deficiencies or preference data to enhance response quality. After cleaning and review, this new data fuels self-training. This iterative process can continuously improve model performance. `LANCE` autonomously constructs data for model post-training, eliminating the need for human or external model involvement, markedly reducing time and cost. An illustration of our methodology is provided in Figure 1. This self-supervised approach shifts from human-supervised learning to autonomous self-evolution. By enabling the model to continuously generate and refine its own data, thereby iteratively enhancing its performance and expanding the dataset's utility, LANCE offers a truly scalable path towards more capable AI, especially valuable in low-resource contexts.

To evaluate our method, we conduct iterative fine-tuning on different models and assess their capabilities in various aspects such as scientific reasoning, commonsense reasoning, and mathematical reasoning. Even with iterative processing on a small dataset, the model's average performance across various tasks rises, with individual metrics remaining stable or improving. The experimental results demonstrate `LANCE`'s effectiveness in enabling continuous model improvement through autonomous data processing. The consistent performance gains across multiple reasoning tasks suggest the potential of this self-evolving approach for advancing model intelligence. Importantly, our approach ensures efficient and cost-effective data generation, markedly reducing the reliance on human intervention or external models. This autonomous capability contributes valuable insights to developing more autonomous and highly capable AI.

In summary, our contributions are as follows: (1) We propose **`LANCE`, a novel training paradigm**, which enables LLMs to autonomously generate, clean, review, and annotate data for self-improvement, substantially reducing time, expense, and the need for extensive supervisory signals in post-training data preparation. (2) `LANCE` autonomously manages the entire post-training data construction process, enhancing data generation efficiency and quality while **boosting model performance across diverse tasks**. (3) `LANCE` **enhances mathematical reasoning**, improving both elementary and advanced tasks as well as multilingual proficiency, despite relying solely on general-purpose data for training data generation.

2 Related Work

2.1 Data Augmentation for LLMs

Data augmentation techniques enhance LLMs' training data. For example, Alpaca (Taori et al., 2023) uses text-davinci-003 (Ouyang et al., 2022) to generate data for fine-tuning a 7B LLaMA (Touvron et al., 2023) model. CoAnnotating (Li et al., 2023) employs human-LLM collaboration for efficient annotation. ReST-MCTS* (Zhang et al., 2024) integrates process reward with MCTS* to collect higher-quality reasoning traces. However, this approach is limited to tasks requiring ground-truth data. These methods often rely on external models, human intervention, or specific task constraints, increasing costs and limiting scalability.

2.2 Self-Evolving LLMs

Self-instruct (Wang et al., 2023a) harnesses the LLM itself to construct instruction data, iteratively enhancing the model's instruction-following capability through SFT. Instruction backtranslation (Li et al., 2024) augments unlabeled data with an instruction prediction model but requires substantial input. Self-rewarding (Yuan et al., 2024) employs the LLM itself as a reward model, iteratively training with DPO (Rafailov et al., 2024), improving both the policy and reward models. SPIN (Chen et al., 2024b) aligns models with human preferences through self-play but is limited by data distribution. Similarly, SPPO (Wu et al., 2024b) approximates the Nash equilibrium through iterative policy updates, relying on an external reward model.

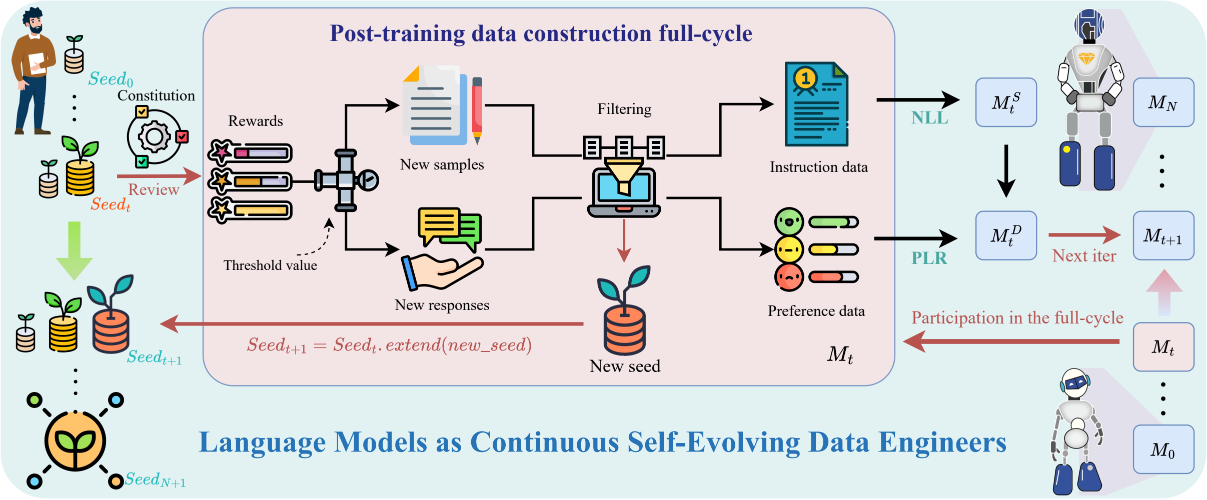


Figure 2: **Overview of LANCE.** Cycle begins at $t = 0$ with pre-annotated seed data $Seed_0$. Each step t: M_t, guided by a Constitution, reviews $Seed_t$, then generates instruction/preference data via **Post-training data construction full-cycle**. M_t is fine-tuned on instruction data (NLL) yielding M_t^S, then on preference data (PLR) yielding M_t^D. For the next iteration, M_t^D becomes M_{t+1}, and newly generated samples are merged with $Seed_t$ to form $Seed_{t+1}$.

LANCE distinguishes itself by autonomously engineering the entire post-training data lifecycle: generating, cleaning, reviewing, and annotating data with preference information. It uniquely uses the LLM as an internal, dynamic reward model and reviewer to identify and fix seed data deficiencies. This allows LANCE to expand beyond the initial data distribution without external models or expert annotation, driving continuous self-improvement from a small seed dataset via iterative data construction, SFT, and DPO.

3 Methodology

Figure 2 illustrates an overview of our approach. Starting with a small seed dataset, the model generates instruction data and preference data through the pipeline, which are used to optimize the model via Negative Log-Likelihood (NLL) and Preference-driven Likelihood Ratio (PLR). The resulting model serves as the starting point for the next iteration. By repeating this cycle, the model's overall performance is continuously improved.

3.1 Preliminaries

Supervised Fine-Tuning (SFT) SFT, also known as instruction tuning, is typically applied to a pre-trained LLM to enhance its ability to understand and follow instructions. For a given model M_t, the training dataset $D^S = \{(x_i, y_i)\}_{i=1}^N$ consists of N instruction-response pairs (x, y). During SFT, the LLM is trained to minimize the NLL loss:

$$L_S = -\mathbb{E}_{D^S}[\sum_{k=1}^{T} log(M_t(y_k|y_{<k}, x))] \quad (1)$$

where T represents the length of response y. From Equation 1, L_S attains its minimum when the distribution of M_t coincides with the conditional probability distribution $p(y|x)$ of the responses in D^S.

Reinforcement Learning from Human Feedback (RLHF) RLHF trains models to learn human preferences from human feedback through reinforcement learning, enabling responses that align with human expectations. This approach plays a crucial role in building safe, high-performance, and controllable AI systems (Ouyang et al., 2022). Direct Preference Optimization (DPO) (Rafailov et al., 2024) eliminates the need for a reward model by directly using human preferences. Given the SFT model M_t^S and the model under optimization M_t^θ, the preference dataset $D^P = \{(x_i, y_i^w, y_i^l)\}_{i=1}^N$ is used for training, where y^w and y^l denote the preferred and dispreferred responses, aiming to minimize the PLR loss:

$$L_D = -\mathbb{E}_{D^P}[log\, \sigma(\hat{r}(x, y^w) - \hat{r}(x, y^l))] \quad (2)$$

In this context, $\hat{r}(x, y^w)$ and $\hat{r}(x, y^l)$ represent the reward values assigned by the model under optimization, M_t^θ, and the reference model, M_t^S, to the preferred response y^w and the dispreferred response y^l, respectively. They are computed as:

$$\hat{r}(x,y) = \beta log \frac{M_t^{\theta}(y|x)}{M_t^{S}(y|x)} \tag{3}$$

where $\beta > 0$ controls the deviation from the base reference policy, namely the initial SFT model M_t^S.

3.2 Initialization

Seed Data Our seed data comprises two components, randomly mixed: (i) A small labeled dataset, providing the initial basis for new data sampling. (ii) A review dataset with prompts containing an instruction and a response, and outputs detailing the rationale and score.

Large Language Model The initial model M_0 for the iterative loop is an LLM fine-tuned (SFT) on the mixed seed data. This SFT provides M_0 with foundational instruction-following and data review capabilities, crucial for iteration. The model, starting from M_0, continues to improve through subsequent iterations, evolving into $M_1 \rightarrow \cdots \rightarrow M_t \rightarrow \cdots \rightarrow M_N$, where t denotes the current iteration, with the process beginning at $t = 0$ and concluding at $t = N - 1$.

3.3 Post-training data construction

First, the seed data is reviewed using a language model to assess quality and identify deficiencies. Next, reward-based generation is employed to either enhance low-quality data or create preference pairs for high-quality data. Finally, the generated data undergoes rigorous filtering and is selectively added to the training datasets, ensuring high-quality instruction tuning and preference learning.

Review Seed data. The distribution of examples in an instruction tuning dataset is often uneven, with some topics or tasks having lower-quality instruction data than others. When such data is used for training, it may result in a language model with potential deficiencies in certain capabilities. Zheng et al. (2023) demonstrated the potential of LLM as a judge, showing that a well-trained LLM can achieve high agreement with human evaluations. Based on these insights, we utilize the language model M_t to review the seed data to identify distribution anomalies and capability gaps.

Inspired by Constitutional AI (Bai et al., 2022), we define a constitution $\mathcal{C}$ as a set of principles including clarity, usefulness, challenge, safety, professionalism, and guidance. For each seed example $(x_i, y_i) \in D_t^{\text{seed}}$, M_t conducts multiple reviews:

$$\{\langle s_{ik}, r_{ik} \rangle\}_{k=1}^{K_R} \sim \mathcal{E}_{M_t}(x_i, y_i | \mathcal{C}) \tag{4}$$

where $\mathcal{E}_{M_t}$ is the evaluation operator of model M_t conditioned on the constitution $\mathcal{C}$, producing K_R distinct review tuples, each comprising a scalar quality score $s_{ik} \in [0, 10]$ and a textual rationale r_{ik}. An aggregated quality metric, $\bar{\mathcal{S}}_i$, is then derived, typically the mean of the manifest scores: $\bar{\mathcal{S}}_i = \frac{1}{K_R} \sum_k s_{ik}$.

Reward-Based Generation. For each seed example, we compare the reward value $\bar{\mathcal{S}}_i$ to a given threshold $V \in [0, 10]$ (see Appendix H for selection analysis). Based on this comparison, we classify examples into two sets: low-quality examples $D_{\text{lq}} = \{(x_i, y_i) \mid \bar{\mathcal{S}}_i < V\}$ and high-quality examples $D_{\text{hq}} = \{(x_i, y_i) \mid \bar{\mathcal{S}}_i \geq V\}$.

For examples in D_{lq}, we address data distribution deficiencies using few-shot learning (Brown et al., 2020) and reasoning-based prompting. For each $(x_i, y_i) \in D_{\text{lq}}$, the model generates K new instructions $\{x_{ij}^{'}\}_{j=1}^{K} = M_t(x_i, y_i; \mathcal{P}^s)$ using a few-shot prompt $\mathcal{P}^s$. Subsequently, M_t generates responses: $y_{ij}^{'} = M_t(x_{ij}^{'})$. The resulting pairs $\{(x_{ij}^{'}, y_{ij}^{'})\}_{i,j=1}^{N,K}$ explore the same topics as the original instructions but with higher quality.

For examples in D_{hq}, we enhance the dataset by generating intentionally flawed responses. For each $(x_i, y_i) \in D_{\text{hq}}$, the model generates K flawed responses $\{y_{ij}^{\text{fl}}\}_{j=1}^{K} = M_t(x_i, y_i; \mathcal{P}^p)$. These flawed responses and the original ones are sorted based on their reward values, forming preference pairs for preference learning.

Filtering and Training Data Construction. To construct a high-quality training dataset, the generated data undergo a rigorous filtering process based on sequential projection operators:

$$(x, y)^f = \Pi_{\mathbb{S}} \circ \Pi_{\mathbb{L}}(x, y) \tag{5}$$

where $\Pi_{\mathbb{L}}$ is a length-based filter (removing outliers based on token count) and $\Pi_{\mathbb{S}}$ is a semantic redundancy filter (e.g., using ROUGE-L (Lin, 2004) to discard instances overly similar to existing corpus entries, mitigating diversity collapse (Wu et al., 2024a)). Let $(x_{ij}^{'}, y_{ij}^{'})^f$ and $(x_i, y_{ij}^{\text{fl}})^f$ represent $(x_{ij}^{'}, y_{ij}^{'})$ and $(x_i, y_{ij}^{\text{fl}})$ cleaned respectively.

After these computationally inexpensive cleaning steps, we utilize M_t to reward newly generated data via Equation 4. For instruction tuning, data with a reward $\bar{\mathcal{S}}_i$ greater than V are first collected into the dataset D_t^S, which is then merged with the instruction dataset D^S. This step can be viewed as model-adjudicated rejection sampling: $D_t^S = \{(x_{ij}^{'}, y_{ij}^{'})^f \mid \bar{\mathcal{S}}_i \geq V\}$. The instruction dataset D^S, initialized from D_t^{seed}, is then updated by merging it with $D^S \leftarrow D^S \cup D_t^S$.

For data intended for preference training, we compare the rewards of each of the two responses. Let $\bar{\mathcal{S}}_i^a$ and $\bar{\mathcal{S}}_i^b$ denote the reward values of the two responses for the i-th instruction. If $\bar{\mathcal{S}}_i^a > \bar{\mathcal{S}}_i^b$, the first item is used as the preferred response and the second as the dispreferred response; otherwise, the roles are reversed. Formally, the preference pair for the i-th instruction is constructed as:

$$(x_i, y_i^w, y_i^l) = \begin{cases} (x_i, y_i^a, y_i^b) & \text{if } \bar{\mathcal{S}}_i^a > \bar{\mathcal{S}}_i^b, \\ (x_i, y_i^b, y_i^a) & \text{otherwise,} \end{cases} \tag{6}$$

where x_i is the instruction, y_i^w is the preferred response, and y_i^l is the dispreferred response. These preference pairs are collected into a temporary preference dataset $D_t^P = \{(x_i, y_i^w, y_i^l)\}$. The preference dataset D^P, initialized as an empty set ϕ, is then updated by merging it with $D^P \leftarrow D^P \cup D_t^P$.

3.4 Language Models as Continuous Self-Evolving Data Engineers

The core of `LANCE` lies in its ability to continuously refine and evolve language models through iterative data engineering. Initially, the algorithm uses the initial seed data $Seed_0$ and the language model M to perform SFT, yielding the initial model M_0. In each iteration t (from 0 to $N-1$), it generates two datasets D_t^P and D_t^S using the method described in Section 3.3, which update the preference dataset D^P and the supervised dataset D^S, respectively. The supervised dataset D^S is used to fine-tune M_t through SFT, producing the model M_t^S. Then, the preference dataset D^P is used for DPO on M_t^S, yielding the model M_t^D. This model M_t^D is then directly used as the model for the next iteration, denoted M_{t+1}. After $N-1$ rounds of iteration, the final model M_N obtained is a more powerful language model. Algorithm 1 in Appendix B provides a detailed description of `LANCE`.

4 Experiments

4.1 Experimental Setup

Models We used Qwen2-7B and Qwen2-7B-Instruct (Yang et al., 2024) as the backbone models to assess the effectiveness of our training paradigm across different model alignment phases (other model experiments in Appendix I).

Datasets We construct the seed dataset from two sources: (1) 3,184 examples sampled from UltraChat (Ding et al., 2023), and (2) 5,632 examples from OpenAssistant Conversations (Köpf et al., 2024), which include human-labeled scores. Using Llama3-70B (Dubey et al., 2024), we generate reward rationales and scores, retaining only those consistent with the human labels.

Baselines We employ several representative methods as our baselines: (1) SFT (Supervised Fine-Tuning): The starting point for all self-evolution methods. (2) Self-Instruct (Wang et al., 2023a): Enhances instruction-following by generating new instruction data using the LLM itself. (3) SPIN (Chen et al., 2024b): A self-evolution approach that leverages self-play fine-tuning to improve the LLM. (4) I-SHEEP (Liang et al., 2024): A self-alignment method where the LLM generates and assesses its own training data to achieve self-improvement.

Benchmarks We assess our model using a comprehensive suite of benchmarks aligned with the Huggingface Open LLM Leaderboard (Beeching et al., 2023), including HellaSwag (Zellers et al., 2019) and Winogrande (Sakaguchi et al., 2020) for commonsense reasoning, MMLU (Hendrycks et al., 2021) for multi-domain knowledge, TruthfulQA (Lin et al., 2022) for accuracy, GSM8K (Cobbe et al., 2021) for mathematical reasoning, and ARC-Challenge (Clark et al., 2018) for scientific reasoning. Additionally, we assess the model's iterative improvements in mathematical reasoning abilities using MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), MGSM (Shi et al., 2023), and Minerva Math (Lewkowycz et al., 2022). The specific evaluation settings for these benchmarks are detailed in Appendix C. Additionally, Appendix D details the hyperparameter settings used during the sampling and training processes.

4.2 `LANCE` Improves Benchmark Performance

Table 1 presents the results of `LANCE` and other iterative self-evolution methods on Qwen2 across multiple benchmarks, showing their performance at optimal iteration rounds. Figure 3 illustrates the average performance of these methods across each iteration round. Notably, our approach demonstrates strong performance both in post-training the pre-trained model and in further refining the fully trained model. Appendix E shows performance on each benchmark per iteration, and Appendix F tracks model performance after each step.

For the average scores, `LANCE` shows an improvement of 3.64 on Qwen2-7B over the initial model (SFT), and 1.75 on Qwen2-7B-Instruct. The most notable improvement is in mathematical abilities.

Backbone	Method	Average	HuggingFace Open LLM Leaderboard					
			ARC	HellaSwag	MMLU	TruthfulQA	GSM8K	Winogrande
7B	SFT	64.60	51.11	78.63	68.71	55.15	60.96	73.01
	Self-Instruct	64.66 (+0.06)	**52.39**	78.34	69.19	50.30	65.20	72.53
	SPIN	68.00 (+3.41)	50.43	**78.98**	**69.68**	55.35	81.43	72.14
	I-SHEEP	67.06 (+2.46)	51.02	78.18	68.34	53.72	78.54	72.53
	LANCE	**68.24 (+3.64)**	50.68	78.76	69.31	**55.54**	**82.11**	**73.01**
7B-Instruct	SFT	67.48	53.07	78.32	68.10	53.96	79.83	**71.59**
	Self-Instruct	67.94 (+0.47)	54.44	79.63	**69.94**	52.67	81.05	69.93
	SPIN	68.41 (+0.93)	53.07	79.72	69.80	55.27	82.03	70.56
	I-SHEEP	68.67 (+1.20)	53.16	**79.95**	69.61	**57.13**	82.34	69.85
	LANCE	**69.22 (+1.75)**	**55.89**	79.74	69.58	55.62	**83.55**	70.96

Table 1: **Performance of multiple self-evolution methods at their optimal iteration rounds across various benchmarks on Qwen2.** SFT represents the initial model obtained through SFT on the seed dataset. **Bold** values denote the best results achieved, underlined values signify the second-best results, red values highlight the improvement over the base model. LANCE outperforms other baselines in terms of average performance across these benchmarks, often ranking as the top or second-best in most benchmarks.

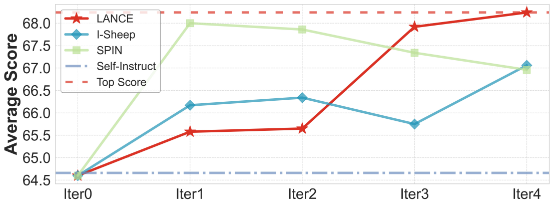
Model	Average	GSM8K	MATH	MGSM_latin					Olympiad Bench	Minerva Math
				de	sw	es	fr	average		
SFT	40.32	60.96	41.74	52.80	1.20	60.40	58.00	43.10	11.10	44.68
Self-Instruct	31.63	65.20	11.98	29.60	6.00	42.80	34.80	28.30	8.00	44.68
SPIN Iter1	37.56	81.43	25.62	20.40	0.80	36.00	34.00	22.80	12.30	45.64
SPIN Iter2	39.87	81.96	26.00	54.80	0.00	43.20	36.80	33.70	12.30	45.40
SPIN Iter3	39.97	81.58	26.44	54.40	0.00	43.60	36.00	33.50	12.70	45.62
SPIN Iter4	34.68	80.74	17.80	46.40	0.00	20.00	4.00	17.60	12.00	45.28
I-SHEEP Iter1	37.86	73.09	34.50	47.20	2.00	62.40	44.00	38.90	7.10	35.70
I-SHEEP Iter2	38.05	74.37	36.38	44.00	0.80	57.60	45.60	37.00	7.10	35.38
I-SHEEP Iter3	38.57	70.96	39.80	35.20	2.80	57.20	51.60	36.70	8.10	37.28
I-SHEEP Iter4	38.96	78.54	34.40	41.60	1.60	56.40	44.40	36.00	7.40	38.46
LANCE Iter1	43.51	67.32	41.90	65.60	2.80	70.40	**67.20**	51.50	11.40	45.44
LANCE Iter2	44.96	66.64	46.54	66.80	3.60	71.60	66.80	52.20	13.60	45.84
LANCE Iter3	47.54	80.14	47.22	65.60	**6.80**	68.00	65.20	51.40	13.60	45.35
LANCE Iter4	**48.83**	**82.11**	**48.12**	**67.60**	6.40	**72.00**	66.40	**53.10**	**14.70**	**46.10**

Table 2: Evolution of mathematical reasoning capabilities in multiple self-evolving algorithms on Qwen2-7B.

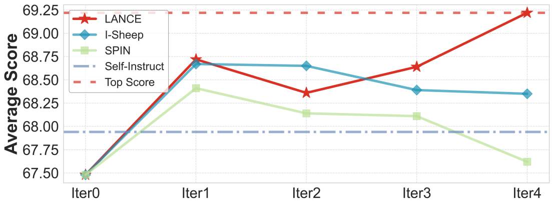
On GSM8K, Qwen2-7B improves by 21.15, while Qwen2-7B-Instruct improves by 3.72. For MMLU, a benchmark covering a wide range of domains and significant challenges, our method also yields improvements: Qwen2-7B improves by 0.60, and Qwen2-7B-Instruct by 1.48. Other abilities either improve or remain largely unchanged.

Regarding iterative self-evolution, our method demonstrates notable improvements over iterations, particularly for the pre-trained model (Qwen2-7B), as shown in the line chart (Figure 3). For the fully post-trained model (Qwen2-7B-Instruct), while intermediate iterations exhibit some fluctuations, the final iteration achieves a substantial performance gain, surpassing all previous rounds and reaching the highest performance level. This difference likely arises because Qwen2-7B-Instruct, having undergone extensive post-training, requires higher-quality data for further improvement. Consequently, notable gains only emerge after multiple iterations generate sufficiently high-quality data.

In contrast, SPIN shows improvements only in the first round, with performance declining in subsequent rounds. This may be explained by Chen et al. (2024b), who establish that SPIN converges only when the LLM's distribution aligns with the seed data. The limited seed data in our experiments likely restricts SPIN's gains, whereas LANCE can generate data beyond the seed distribution, enabling continuous improvement. Further discus-



(a) Qwen2-7B



(b) Qwen2-7B-Instruct

Figure 3: **Various self-evolution methods show average scores across benchmarks.** The Self-Instruct method, without iterative processes, sampled 50k examples for self-training. "Iter t" denotes the t-th iteration.

Model	Full	w/o dpo	w/o sft
SFT	64.60	64.60	64.60
LANCE Iter1	65.58	66.89	61.57
LANCE Iter2	65.65	67.08	61.00
LANCE Iter3	67.92	**67.85**	59.23
LANCE Iter4	**68.24**	67.79	**64.08**

Table 3: The changes in average performance of the Qwen2-7B model when certain steps are removed from LANCE. SFT serves as the starting point for all iterations. "w/o dpo" excludes DPO-related steps, while "w/o sft" removes SFT-related steps. The results underscore the necessity of a complete pipeline, as the absence of either component leads to slow and unstable improvement.

sion and distribution visualization can be found in Appendix A, with case studies in Appendix J.

I-SHEEP exhibits improvement in the first iteration on Qwen2-7B-Instruct but suffers from performance degradation in later iterations. On Qwen2-7B, I-SHEEP demonstrates an oscillating pattern, with performance rising in iterations 1 and 2, dropping in iteration 3, and recovering in iteration 4. However, even after this recovery, I-SHEEP's final performance remains markedly lower than LANCE's, with a gap of 1.18. Due to computational constraints, we conducted four rounds of iterative experiments with $N = 5$. The results show consistent performance improvements in each iteration, indicating potential for further gains with additional iterations and highlighting the promise of self-evolving language models.

4.3 LANCE Improves Math Skills

Table 2 compares Qwen2-7B's performance under various self-evolution algorithms on mathematical reasoning benchmarks. We included GSM8K, a benchmark from the Open LLM Leaderboard that evaluates elementary mathematical abilities, along with MATH, Olympiad Bench, and Minerva Math to assess advanced reasoning capabilities on competition-level and even Olympiad-level mathematical problems. Furthermore, MGSM was used to assess multilingual mathematical proficiency.

On the benchmark of elementary mathematical abilities (as measured by GSM8K), all methods improved the SFT model's performance. LANCE also exhibited notable advantages in competition-level mathematical abilities. Specifically, on the MATH benchmark, only LANCE achieved a notable improvement, increasing accuracy by 6.38. On the Olympiad Bench, while both LANCE and SPIN improved accuracy, SPIN's best result yielded only a modest gain of 1.60, whereas LANCE achieved a more substantial improvement of 3.60. Additionally, on the Minerva Math benchmark, both LANCE and SPIN enhanced model performance with comparable gains. This discrepancy may stem from the following: improvements in elementary mathematical abilities rely on pattern memorization and optimization of simple problems, while advancements in higher-level mathematical abilities require methods capable of generating data with complex reasoning logic. The accurate generation of data involving complex reasoning may benefit from our approach of requiring the model to engage in thorough deliberation before producing an answer during both the generation and review phases. This deliberate reasoning process enhances the model's understanding of the input, thereby improving the quality and accuracy of the generated data.

In terms of multilingual mathematical abilities (as evaluated by MGSM), despite the training set being exclusively in English, the model also achieved substantial improvements in mathematical proficiency across other Latin-based languages. Notably, only our algorithm demonstrated this cross-lingual transfer capability. This phenomenon may be explained by LANCE's ability to generate high-quality mathematical data that genuinely enhances the model's reasoning capabilities, enabling strong generalization across linguistic contexts.

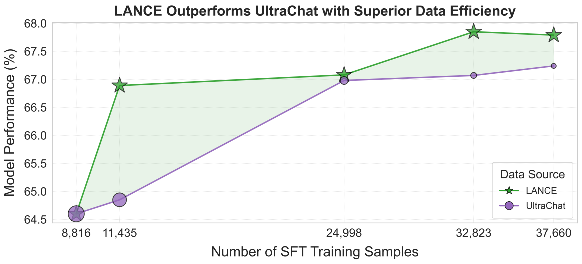


Figure 4: Qwen2-7B SFT performance with different data sources by sampled data amount. Dot size indicates supervisory signal efficiency (model performance / amount of supervisory signal). Highlighted areas show where LANCE outperforms UltraChat.

The notable improvement in mathematical reasoning, despite a generic seed dataset, is attributed to `LANCE`'s core mechanics. The autonomous data generation and review processes inherently compel the model to engage in multi-step logical reasoning. This engagement appears to embed richer logical structures within the self-generated data, even when originating from general-purpose inputs. Consequently, the model is trained on data that, through its construction process, implicitly carries more complex reasoning patterns, thereby fostering its mathematical problem-solving skills and highlighting `LANCE`'s potential to effectively leverage general-purpose data for specialized capability enhancement.

4.4 Ablation Studies

Table 3 illustrates the impact on the Qwen2-7B model's average performance when either SFT-related or DPO-related components are omitted. When the DPO-related components are removed, the model's performance can still improve iteratively in the first three iterations, but at a slower rate compared to the full pipeline, and performance starts to decline after the fourth iteration. This suggests that DPO components are instrumental in accelerating performance gains and also essential for sustaining long-term improvements. When the SFT-related components are removed, the model's performance consistently declines in the first three iterations and only partially recovers in iter4, remaining below the initial performance. This highlights the critical role of SFT in maintaining the model's foundational stability and ensuring steady progress throughout the training process. These findings collectively emphasize the necessity of a complete and well-integrated data generation and training pipeline. The interplay between SFT and DPO components appears to be synergistic: SFT provides the foundational stability, while DPO drives faster and more sustainable improvements. Table 9 in Appendix G provides a detailed breakdown of the model's performance across each test benchmark throughout the iterations.

4.5 Supervisory Signal Efficiency of `LANCE`

To evaluate `LANCE`'s low-resource efficiency, we compared its generated SFT data against randomly sampled UltraChat data (Figure 4). A key distinction is the supervisory cost for SFT data generation: `LANCE` consistently uses only an initial 8,816 seed examples, regardless of the final SFT dataset size, whereas UltraChat's required supervision scales with the number of SFT samples. This smaller and fixed supervisory footprint not only substantially reduces the time and cost of post-training data preparation but also enables `LANCE` to achieve superior model performance when equivalent SFT data volumes are used for fine-tuning. This translates to higher data efficiency for `LANCE`, visually represented in Figure 4 by its larger data point sizes as the SFT dataset expands, whereas UltraChat's corresponding data points become smaller. Essentially, while UltraChat's gains require an increasing supervisory budget, `LANCE` excels at amplifying a small, fixed supervisory signal into substantial performance improvements, underscoring its value as a resource-efficient paradigm for advancing LLMs when extensive labeled data is scarce.

5 Conclusion

We introduce `LANCE`, a novel training paradigm where LLMs autonomously generate, clean, review, and annotate data with preference information, thereby reducing post-training data construction time and cost. A key strength of `LANCE` is its efficiency in leveraging supervisory signals, enabling substantial model improvements even from a minimal initial seed dataset. This paradigm proves effective for both the initial post-training of pre-trained models and the continued post-training of already tuned models, demonstrating consistent performance gains across diverse tasks and outperforming other self-evolution methods. Notably, `LANCE` significantly boosts mathematical reasoning capabilities, even when its training originates from general-purpose datasets. By ensuring that autonomously generated data aligns with human preferences while minimizing the resource demands for high-quality data creation, especially through its efficient use of supervision, `LANCE` offers a scalable and resource-efficient path toward developing more advanced and autonomous AI systems.

Limitation

Our experimental results demonstrate that while `LANCE` substantially enhances the model's mathematical reasoning capabilities, its improvement on knowledge-dependent tasks remains limited. Specifically, in the absence of external supervision signals, the self-evolved data cannot introduce new knowledge beyond the model's existing capacity. This inherent limitation suggests that self-evolution may have restricted potential in augmenting the model's knowledge base, making it more suitable for enhancing weakly knowledge-dependent capabilities. Addressing the enhancement of strongly knowledge-dependent abilities remains an open challenge for future research.

Furthermore, while `LANCE` demonstrates strong efficacy when initial supervisory data is scarce (i.e., in data-low-resource scenarios) by operating with only a small seed dataset and autonomously generating substantial training data, its application in compute-low-resource settings presents a challenge. The process involves multiple iterations of sampling, reviewing, and annotation, which introduces non-negligible computational overhead. This computational cost represents a limitation that necessitates further optimization in future work.

Ethic Statement

In this study, all datasets and models utilized are publicly available and adhere to their respective open-source licenses. The use of these resources is strictly confined to academic research purposes. However, we acknowledge that the misuse of the methodology proposed in this work could potentially lead to the generation of unethical data. This issue is not unique to our approach but is a shared concern across all data generation methods. The potential biases and ethical challenges associated with AI-generated content (AIGC) necessitate ongoing discussion and research within the community to ensure responsible development and deployment of such technologies.

Acknowledgments

This work is supported by the National Natural Science Foundation of China. (No. 62272092, No. 62172086, No. 62106039)

We are deeply grateful to Tianhao Wu for his invaluable contribution in designing the high-quality illustrations. His clear and professional visualizations have significantly enhanced the clarity and exposition of our proposed method. We would also like to thank all collaborators for their hard work and effort.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. 2023. Open llm leaderboard. `https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeffrey Wu. 2024. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. In *Forty-first International Conference on Machine Learning*.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024a. Alpagasus: Training a better alpaca with fewer data. In *The Twelfth International Conference on Learning Representations*.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024b. Self-play fine-tuning converts weak language models to strong language models. In *Forty-first International Conference on Machine Learning*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Haixing Dai, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao, Shaochen Xu, Fang Zeng, Wei Liu, et al. 2025. Auggpt: Leveraging chatgpt for text data augmentation. *IEEE Transactions on Big Data*.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. 2024. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipalli, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024. Llm2llm: Boosting llms with novel iterative data enhancement. *arXiv preprint arXiv:2403.15042*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Minzhi Li, Taiwei Shi, Caleb Ziems, Min-Yen Kan, Nancy F Chen, Zhengyuan Liu, and Diyi Yang. 2023. Coannotating: Uncertainty-guided work allocation between human and large language models for data annotation. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. 2024. Self-alignment with instruction backtranslation. In *The Twelfth International Conference on Learning Representations*.

Yiming Liang, Ge Zhang, Xingwei Qu, Tianyu Zheng, Jiawei Guo, Xinrun Du, Zhenzhu Yang, Jiaheng Liu, Chenghua Lin, Lei Ma, et al. 2024. I-sheep: Self-alignment of llm from scratch through an iterative self-enhancement paradigm. *arXiv preprint arXiv:2408.08072*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon LLM: Outperforming curated corpora with web data only. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. arxiv 2023. *arXiv preprint arXiv:2302.13971*, 10.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. 2024. Position: Will we run out of data? limits of LLM scaling based on human-generated data. In *Forty-first International Conference on Machine Learning*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023a. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Zige Wang, Wanjun Zhong, Yufei Wang, Qi Zhu, Fei Mi, Baojun Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023b. Data management for large language models: A survey. *arXiv preprint arXiv:2312.01700*.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Ting Wu, Xuefeng Li, and Pengfei Liu. 2024a. Progress or regress? self-improvement reversal in post-training. *arXiv preprint arXiv:2407.05013*.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2024b. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2025. Jasper and stella: distillation of sota embedding models. *Preprint*, arXiv:2412.19048.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Appendix

A Data Distribution Visualization

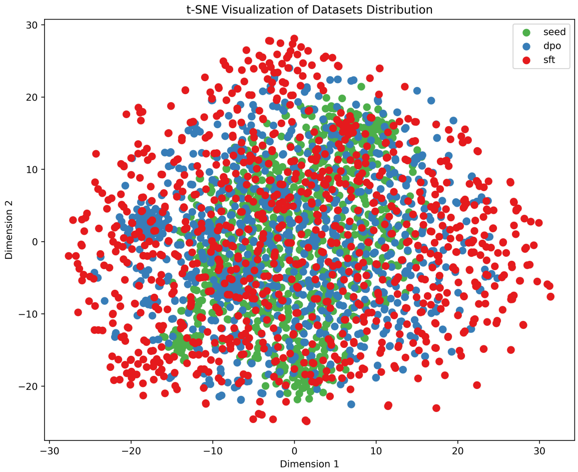


Figure 5: Visualization of the distribution of seed data and synthetic data generated by LANCE

We sampled 1000 examples each from the seed dataset, the synthetic SFT dataset, and the DPO dataset. Using the stella_en_400M_v5 model (Zhang et al., 2025), we extracted embeddings for each example and applied t-SNE for dimensionality reduction, visualized in Figure 5.

The visualization reveals that the synthetic data generated by LANCE not only encompasses the distribution range of the original seed data but also explores new regions in the embedding space. This indicates that LANCE can produce data that aligns with a broader and more diverse distribution, effectively expanding the original data distribution. Such expansion is particularly valuable for improving model generalization, as it introduces variability that better reflects real-world scenarios.

Notably, the SFT dataset exhibits the widest distribution range among the three datasets. This can be attributed to the generation of new instructions during its construction, which introduces additional diversity and complexity to the data. This suggests that the instruction generation process in LANCE plays a critical role in enhancing data diversity and coverage.

Overall, these findings highlight the effectiveness of LANCE in generating high-quality synthetic data that not only preserves the characteristics of the original seed data but also extends its boundaries, enabling more robust and generalizable model training.

B Algorithm Description

In this section, we present the detailed implementation of our proposed framework, Language Models as Continuous Self-Evolving Data Engineers (LANCE), as described in Algorithm 1. The algorithm outlines the key steps and methodologies employed to achieve continuous self-evolving data engineering for enhancing language models.

C Evaluation Setting

Benchmarks	num shots	version	eval tools
ARC-C	0	1.0	LM Evaluation Harness[1]
HellaSwag	0	1.0	LM Evaluation Harness
MMLU	0	1.0	LM Evaluation Harness
TruthfulQA	6	2.0	LM Evaluation Harness
Winogrande	0	1.0	LM Evaluation Harness
Minerva Math	4	1.0	LM Evaluation Harness
GSM8K	4	1d7fe4	OpenCompass[2]
MATH	0	393424	OpenCompass
MGSM	0	d967bc	OpenCompass
Olympiad Bench	0	-	Qwen2.5-Math[3]

Table 4: Details of the evaluation settings for all benchmarks in this study.

Figure 4 provides an overview of the evaluation details for all benchmarks included in this study. The **'num shots'** column indicates the number of few-shot examples provided during evaluation, which varies across benchmarks, with values such as 0, 4, or 6 depending on the task. The **'version'** column specifies the version of the evaluation configuration file used, ensuring reproducibility and consistency in the assessment process. Lastly, the **'eval tools'** column identifies the evaluation frameworks employed, such as LM Evaluation Harness and OpenCompass, along with their respective GitHub repositories for reference. This table encapsulates the key parameters and tools used to ensure a rigorous and standardized evaluation across all benchmarks.

D Hyperparameter Settings

In this section, we provide detailed descriptions of the hyperparameters used during the sampling and training processes. All experiments were conducted using two NVIDIA RTX A6000 GPUs, each equipped with 48GB VRAM, ensuring efficient data generation and model training. Table 5

---

[1] `https://github.com/EleutherAI/lm-evaluation-harness`

[2] `https://github.com/open-compass/opencompass`

[3] `https://github.com/QwenLM/Qwen2.5-Math`

Algorithm 1 Language Models as Continuous Self-Evolving Data Engineers

```
Input: Initial seed data $D_0^{\text{seed}}$, initial language model $M$, iteration count $N$, quality threshold $V$
Output: More powerful language model $M_N$
 1: Initialize $D^S \leftarrow D_0^{\text{seed}}$, $D^P \leftarrow \phi$, $M_0 \leftarrow \text{SFT}(M, D^S)$
 2: for $t = 0$ to $N - 1$ do
 3:     // Step 1: Review seed data
 4:     if $t = 0$ then
 5:         for each $(x_i, y_i) \in D_t^{\text{seed}}$ do
 6:             Compute $\{\langle s_{ik}, r_{ik} \rangle\}_{k=1}^{K_R} \sim \mathcal{E}_{M_t}(x_i, y_i | \mathcal{C})$ using Equation 4
 7:             Aggregate quality metric $\bar{\mathcal{S}}_i = \frac{1}{K_R} \sum_k s_{ik}$
 8:         end for
 9:     end if
10:     Initialize low-quality and high-quality examples $D_{\text{lq}} \leftarrow \phi$, $D_{\text{hq}} \leftarrow \phi$
11:     for each $(x_i, y_i, \bar{\mathcal{S}}_i) \in D_t^{\text{seed}}$ do
12:         if $\bar{\mathcal{S}}_i < V$ then
13:             $D_{\text{lq}} = D_{\text{lq}} \cup \{(x_i, y_i)\}$
14:         else
15:             $D_{\text{hq}} = D_{\text{hq}} \cup \{(x_i, y_i)\}$
16:         end if
17:     end for
18:     // Step 2: Reward-based generation
19:     for each $(x_i, y_i) \in D_{\text{lq}}$ do
20:         Generate $K$ new instructions $\{x_{ij}^{'}\}_{j=1}^{K} = M_t(x_i, y_i; \mathcal{P}^s)$
21:         Generate the corresponding responses $\{y_{ij}^{'}\}_{j=1}^{K} = M_t(x_{ij}^{'})$
22:     end for
23:     for each $(x_i, y_i) \in D_{\text{hq}}$ do
24:         Generate $K$ flawed responses $\{y_{ij}^{\text{fl}}\}_{j=1}^{K} = M_t(x_i, y_i; \mathcal{P}^p)$
25:     end for
26:     // Step 3: Data cleaning and annotation
27:     Clean generated data using length-based filtering $\Pi_{\mathbb{L}}$ and semantic redundancy filtering $\Pi_{\mathbb{S}}$:

            $$(x_{ij}^{'}, y_{ij}^{'})^f = \Pi_{\mathbb{S}} \circ \Pi_{\mathbb{L}}(x_{ij}^{'}, y_{ij}^{'}), \quad (x_i, y_{ij}^{\text{fl}})^f = \Pi_{\mathbb{S}} \circ \Pi_{\mathbb{L}}(x_i, y_{ij}^{\text{fl}})$$

28:     Initialize temporary instruction and preference dataset $D_t^S \leftarrow \phi$, $D_t^P \leftarrow \phi$
29:     Compute rewards $\bar{\mathcal{S}}_i$ for $\{(x_{ij}^{'}, y_{ij}^{'})^f\} \cup \{(x_i, y_{ij}^{\text{fl}})^f\}$ like lines 5-10.
30:     Update seed data $D_{t+1}^{\text{seed}} \leftarrow D_t^{\text{seed}} \cup \{(x_{ij}^{'}, y_{ij}^{'}, \bar{\mathcal{S}}_{ij})^f\} \cup \{(x_i, y_{ij}^{\text{fl}}, \bar{\mathcal{S}}_{ij})^f\}$
31:     if $(x_i, y_i) \in \{(x_{ij}^{'}, y_{ij}^{'})^f\}$ and $\bar{\mathcal{S}}_i < V$ then
32:         $D_t^S \leftarrow D_t^S \cup \{(x_i, y_i)\}$
33:     end if
34:     for each pair $y_i^a, y_i^b \in \{y_{ij}^{\text{fl}}\}_{j=1}^{K} \cup \{y_i\}$ do
35:         if $\bar{\mathcal{S}}_i^a > \bar{\mathcal{S}}_i^b$ then
36:             $D_t^P \leftarrow D_t^P \cup \{x_i, y_i^a, y_i^b\}$
37:         else
38:             $D_t^P \leftarrow D_t^P \cup \{x_i, y_i^b, y_i^a\}$
39:         end if
40:     end for
41:     Update supervised and preference dataset $D^S \leftarrow D^S \cup D_t^S$, $D^P \leftarrow D^P \cup D_t^P$
42:     // Step 4: Model fine-tuning
43:     Fine-tune $M_t$ on $D^S$ via SFT (Equation 1) to obtain $M_t^S$
44:     Perform DPO (Equation 2) on $M_t^S$ using $D^P$ to obtain $M_t^D$
45:     Set $M_{t+1} \leftarrow M_t^D$
46: end for
```

outlines the hyperparameters employed during the sampling phase with `LANCE`, which includes key settings such as the **threshold** V, set to 7.0, which acts as the reward cutoff for distinguishing high-quality data from low-quality data. The **sample size** K, set to 4, determines the number of samples generated during both the data generation and review phases. To maintain randomness and control the diversity of outputs, **Top-p** is set to 0.9, and **Temperature** is set to 0.7. Additionally, **Max New Tokens** limits the maximum number of tokens generated during sampling to 512, while **Min Length** and **Max Length** define the acceptable token length range for filtering, set to 10 and 4096, respectively.

Parameter	Value
Threshold V	7.0
Sample nums K	4
Top-p	0.9
Temperature	0.7
Max new tokens	512
Min length	10
Max length	4096

Table 5: The hyperparameters used during the sampling process with `LANCE`.

Meanwhile, Table 6 details the hyperparameters used during the training phase, where the model is fine-tuned on the generated data. For SFT, the learning rate is set to 3e-5, with a batch size of 2 and gradient accumulation steps of 2, trained over 1 epoch. A warmup ratio of 0.01 is applied, and the cutoff length is set to 4096. For DPO, the learning rate is reduced to 5e-6, with a batch size of 1 and gradient accumulation steps of 8, also trained over 1 epoch. The warmup ratio remains at 0.01, and the cutoff length is similarly set to 4096. The β parameter, which controls the strength of the preference optimization, is set to 0.2. These configurations ensure a balanced and effective training process, tailored to the specific requirements of each method.

E Iteration details

Table 7 presents the specific scores of `LANCE` and baseline methods during the iterative process across various benchmarks, where red / **green** values indicate improvements/declines compared to the previous iteration, respectively. For the pre-trained model Qwen2-7B, only `LANCE` achieved continuous performance improvement (all change values are marked in red). Although the I-SHEEP method showed an upward trend in the first two iterations, it experienced a decline in the third iteration. Despite rebounding to the peak in the fourth iteration, its overall performance still lags significantly behind `LANCE`. For the fully post-trained Qwen2-7B-Instruct model, only `LANCE` demonstrated sustained improvement over multiple iterations, while other methods showed progress only in the first iteration. Notably, although `LANCE` exhibited a temporary performance fluctuation in the second iteration, it rebounded in the third iteration and reached the performance peak in the final iteration. These comparative results fully validate the significant advantages and potential of `LANCE` in continuously optimizing model performance.

F Model Evolution Steps

Table 8 illustrates the impact of each step on model performance during the implementation of `LANCE`. For Qwen2-7B, although the DPO phase in the first three iterations resulted in performance degradation, the results in Table 8 without DPO-related components reveal that the absence of these components leads to only slow and unstable iterative improvements. This underscores the critical role of DPO-related components in enhancing the model's ability to generate high-quality data. For Qwen2-7B-Instruct, performance declined in the second and third iterations but rebounded notably in the fourth iteration, achieving substantial improvement. These findings collectively emphasize the importance of a complete iterative training pipeline for achieving robust and consistent performance gains.

G Ablation Details

Table 9 provides a detailed breakdown of our ablation experiments, showing the performance of `LANCE` with specific components removed across each iteration step and evaluation benchmark. Beyond the findings discussed in Section 4.4, where we highlighted that an incomplete pipeline leads to unstable performance improvements or even declines, we observe that removing SFT-related components significantly degrades the model's performance on GSM8K. This is likely because our SFT data construction process generates novel instructions, which enhances the model's mathematical and logical reasoning capabilities. In contrast, the

Method	**Learning Rate**	**Batch Size**	**Gradient Accumulation**	**Epochs**	**Warmup Ratio**	**Cutoff Length**	β
SFT	3e-5	2	2	1	0.01	4096	-
DPO	5e-6	1	8	1	0.01	4096	0.2

Table 6: The hyperparameters used during the training process with LANCE.

Base	**Model**	**Average**	**Benchmarks**					
			ARC	**HellaSwag**	**MMLU**	**TruthfulQA**	**GSM8K**	**Winogrande**
Qwen2-7B	SFT	64.60	51.11	78.63	68.71	55.15	60.96	73.01
	Self-Instruct 50k	64.66 (+0.06)	**52.39**	78.34	69.19	50.30	65.20	72.53
	SPIN Iter1	68.00 (+3.41)	50.43	78.98	69.68	55.35	81.43	72.14
	SPIN Iter2	67.86 (-0.14)	49.83	79.13	**69.73**	55.38	81.96	71.11
	SPIN Iter3	67.34 (-0.52)	48.12	79.31	69.71	54.99	81.58	70.32
	SPIN Iter4	66.96 (-0.38)	46.16	**79.38**	69.66	55.66	80.74	70.17
	I-SHEEP Iter1	66.17 (+1.57)	50.85	78.32	68.45	53.60	73.09	72.69
	I-SHEEP Iter2	66.34 (+0.17)	51.45	78.27	68.49	53.76	74.37	71.67
	I-SHEEP Iter3	65.75 (-0.58)	51.11	78.09	68.29	53.85	70.96	72.22
	I-SHEEP Iter4	67.06 (+1.30)	51.02	78.18	68.34	53.72	78.54	72.53
	LANCE Iter1	65.58 (+0.99)	50.85	78.45	68.96	55.53	67.32	72.38
	LANCE Iter2	65.65 (+0.07)	50.51	78.94	69.41	**55.97**	66.64	72.45
	LANCE Iter3	67.92 (+2.26)	50.77	79.12	69.24	55.78	80.14	72.45
	LANCE Iter4	**68.24** (+0.32)	50.68	78.76	69.31	55.54	**82.11**	**73.01**
Qwen2-7B-Instruct	SFT	67.48	53.07	78.32	68.10	53.96	79.83	**71.59**
	Self-Instruct 50k	67.94 (+0.47)	54.44	79.63	**69.94**	52.67	81.05	69.93
	SPIN Iter1	68.41 (+0.93)	53.07	79.72	69.80	55.27	82.03	70.56
	SPIN Iter2	68.14 (-0.27)	51.79	79.66	69.65	56.46	81.58	69.69
	SPIN Iter3	68.11 (-0.03)	51.19	79.99	69.49	56.42	82.11	69.46
	SPIN Iter4	67.62 (-0.49)	50.77	**80.04**	69.59	56.07	81.05	68.19
	I-SHEEP Iter1	68.67 (+1.20)	53.16	79.95	69.61	**57.13**	82.34	69.85
	I-SHEEP Iter2	68.65 (-0.02)	54.10	79.92	69.56	56.53	81.96	69.85
	I-SHEEP Iter3	68.39 (-0.27)	53.33	79.96	69.38	55.80	82.56	69.30
	I-SHEEP Iter4	68.35 (-0.04)	53.50	79.61	69.37	55.43	82.71	69.46
	LANCE Iter1	68.72 (+1.24)	55.38	79.53	69.34	55.85	81.73	70.48
	LANCE Iter2	68.36 (-0.36)	55.12	79.76	69.55	55.98	80.14	69.61
	LANCE Iter3	68.64 (+0.28)	55.03	79.92	69.56	55.69	80.59	71.03
	LANCE Iter4	**69.22** (+0.59)	**55.89**	79.74	69.58	55.62	**83.55**	70.96

Table 7: **Experimental results of multiple self-evolution methods across various benchmarks.** Red / Green values indicate improvements / decreases compared to the previous iteration. LANCE consistently shows performance gains across iterations, outperforming other baselines.

DPO data construction process does not produce new instructions, which may explain its limited impact in this regard.

H Parameter Sensitivity Analysis

Understanding the influence of hyperparameters is crucial for assessing the robustness of a training paradigm and for optimizing its performance across different scenarios. In this section, we present a sensitivity analysis for a key hyperparameter within LANCE: the data filtering threshold, denoted as V. This threshold plays a significant role in the data selection process during the iterative training loop. Due to computational resource constraints, our current analysis focuses specifically on V, with experiments conducted on LANCE applied to Qwen2-7B during its fourth iteration (iter4) of self-improvement.

The data filtering threshold V is utilized in LANCE to discern the quality of autonomously generated data. A higher V implies stricter filtering criteria, while a lower V is more lenient. In the early stages of our research, an initial analysis was performed to determine a suitable value for V, as repeated adjustments during the full iterative process would be resource-intensive. The value $V = 7.0$

Base	Model	Average	Benchmarks					
			ARC	HellaSwag	MMLU	TruthfulQA	GSM8K	Winogrande
Qwen2-7B	SFT	64.60	53.07	78.32	68.10	53.96	79.83	71.59
	LANCE SFT Iter1	66.89	54.98	79.42	69.44	55.09	81.58	69.85
	LANCE DPO Iter1	65.58	55.38	79.53	69.34	55.85	81.73	70.48
	LANCE SFT Iter2	67.94	54.01	79.64	69.61	55.00	81.12	69.46
	LANCE DPO Iter2	65.65	55.12	79.76	69.55	55.98	80.14	69.61
	LANCE SFT Iter3	68.09	54.01	79.52	69.71	54.76	81.27	70.17
	LANCE DPO Iter3	67.92	55.03	79.92	69.56	55.69	80.59	71.03
	LANCE SFT Iter4	68.20	54.61	79.49	69.53	55.41	83.17	71.11
	LANCE DPO Iter4	68.24	55.89	79.74	69.58	55.62	83.55	70.96
Qwen2-7B-Instruct	SFT	67.48	51.11	78.63	68.71	55.15	60.96	73.01
	LANCE SFT Iter1	68.39	50.85	78.36	68.90	54.80	75.74	72.69
	LANCE DPO Iter1	68.72	50.85	78.45	68.96	55.53	67.32	72.38
	LANCE SFT Iter2	68.14	50.85	78.77	69.01	55.07	81.35	72.61
	LANCE DPO Iter2	68.36	50.51	78.94	69.41	55.97	66.64	72.45
	LANCE SFT Iter3	68.24	50.43	78.78	69.42	55.23	81.80	72.85
	LANCE DPO Iter3	68.64	50.77	79.12	69.24	55.78	80.14	72.45
	LANCE SFT Iter4	68.89	50.94	78.64	69.41	55.41	81.96	72.85
	LANCE DPO Iter4	69.22	50.68	78.76	69.31	55.54	82.11	73.01

Table 8: **Evolution of Model Performance During the Implementation of LANCE.** LANCE SFT Iter t Denotes the Model Fine-tuned with SFT Data After the t-th Iteration, and LANCE DPO Iter t Represents the Model Fine-tuned with DPO Data After the t-th Iteration.

Model	Average	Benchmarks					
		ARC	HellaSwag	MMLU	TruthfulQA	GSM8K	Winogrande
SFT	64.60	51.11	78.63	68.71	55.15	60.96	73.01
LANCE Iter1 w/o dpo	66.89	50.85	78.36	68.90	54.80	75.74	72.69
LANCE Iter2 w/o dpo	67.08	48.63	78.21	69.11	53.87	80.97	71.67
LANCE Iter3 w/o dpo	67.85	50.51	78.65	69.03	54.78	81.80	72.30
LANCE Iter4 w/o dpo	67.79	50.85	78.63	69.06	54.71	81.43	72.06
LANCE Iter1 w/o sft	61.57	**53.05**	78.92	68.98	56.28	39.58	72.61
LANCE Iter2 w/o sft	61.00	51.54	78.84	68.89	**56.73**	37.68	72.30
LANCE Iter3 w/o sft	59.23	51.88	78.87	68.98	56.56	26.46	72.61
LANCE Iter4 w/o sft	64.08	51.79	78.94	69.06	56.22	56.25	72.22
LANCE Iter1	65.58	50.85	78.45	68.96	55.53	72.38	67.32
LANCE Iter2	65.65	50.51	78.94	**69.41**	55.97	66.64	72.45
LANCE Iter3	67.92	50.77	**79.12**	69.24	55.78	80.14	72.45
LANCE Iter4	**68.24**	50.68	78.76	69.31	55.54	**82.11**	**73.01**

Table 9: The performance of LANCE without SFT-related and DPO-related components on each evaluation benchmark at every iteration step.

was selected for the main experiments presented in this paper.

To further investigate and validate the impact of V on model performance, we conducted supplementary experiments by varying this threshold during the fourth iteration (iter4) of LANCE with the Qwen2-7B model. We evaluated performance with V set to 6.0, 7.0, and 8.0. The results, presented in Table 10, include performance metrics across several standard benchmarks.

The experimental results presented in Table 10 highlight the impact of the data filtering threshold V on the performance of LANCE.

- **For** $V = 7.0$**:** This setting, which was used for the main results reported in this paper, yielded the best overall performance, particularly for the DPO variant (Average score

Model Configuration	Average	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K
SFT V=7.0	68.20	50.94	78.64	69.41	55.41	72.85	81.96
DPO V=7.0	**68.24**	50.68	78.76	69.31	55.54	73.01	82.11
SFT V=8.0	67.98	51.11	78.61	69.51	55.41	71.51	81.73
DPO V=8.0	67.60	51.02	78.95	69.08	55.67	71.43	79.45
SFT V=6.0	67.67	51.02	78.91	69.07	55.65	72.69	78.70
DPO V=6.0	67.81	50.94	78.90	69.10	55.42	72.69	79.83

Table 10: Performance of LANCE (Qwen2-7B, iter4) with varying data filtering thresholds V. Scores are averaged over benchmarks, along with individual benchmark scores. The best average performance is highlighted in **bold**.

of 68.24). This suggests that a threshold of 7.0 strikes an effective balance between stringently filtering for high-quality data and retaining a sufficient volume and diversity of examples necessary for robust model training. The quality of data selected at this threshold appears optimal for both the SFT and DPO stages within the fourth iteration.

- **For** $V = 8.0$**:** Increasing the threshold to 8.0 imposes stricter filtering criteria. While the intention is to select only the highest quality data, this stricter approach also leads to a smaller volume of data being available for SFT and subsequent DPO. The results show a slight decrease in average performance for both SFT (67.98) and DPO (67.60) compared to $V = 7.0$. This indicates that while data quality is important, an overly aggressive filtering strategy might discard useful training signals or reduce data diversity to an extent that hampers overall learning and generalization. For instance, the GSM8K score for the DPO model notably dropped to 79.45 from 82.11 with $V = 7.0$.

- **For** $V = 6.0$**:** Conversely, lowering the threshold to 6.0 makes the filtering criteria more lenient. This approach risks including lower-quality data, which may contain more noise, inaccuracies, or less helpful instructional content. The introduction of such noise during training can negatively impact the learning process. As observed, both SFT (67.67) and DPO (67.81) models trained with $V = 6.0$ exhibited a decline in average performance compared to $V = 7.0$. This suggests that the benefits of a larger dataset resulting from lenient filtering are outweighed by the detrimental effects of lower data quality. The GSM8K scores, in particular, were lower for $V = 6.0$ compared to $V = 7.0$.

These findings are consistent with our initial analysis and underscore that the choice of the data filtering threshold V is a significant factor influencing model performance within the LANCE paradigm. An appropriately calibrated threshold is essential to ensure that the model is trained on data that is both high in quality and sufficient in quantity and diversity.

I Generalization of LANCE to Other Model Architectures

To assess the broader applicability of LANCE beyond the Qwen model family, and in response to reviewer suggestions, we conducted preliminary experiments applying our self-evolution paradigm to other open-source models. This appendix presents the results of these investigations on Zephyr and Mistral-NeMo-12B-Instruct. These experiments were conducted for the first two iterations of LANCE.

Experiments on Zephyr. To evaluate LANCE on a different architectural base and to align with related research such as SPIN Chen et al. (2024b), we applied our methodology to Zephyr, a model derived from Mistral-7B-v0.1. We performed two full iterations of LANCE, encompassing both Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO) stages within each iteration. The evaluation results across several standard benchmarks are presented in Table 11.

The results in Table 11 demonstrate that LANCE facilitates continuous performance improvements on the Zephyr model through the first two iterations. Notably, consistent gains are observed in the average score, with the DPO stage in each iteration further enhancing performance. An encouraging trend is the improvement in multi-step reasoning tasks, such as GSM8K, which saw an increase from 39.04 at iter0 to 43.37 after DPO iter2.

Model Configuration	Average	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K
Zephyr iter0	56.52	53.16	79.38	55.90	39.72	71.90	39.04
Zephyr SFT iter1	56.86	52.90	79.19	56.01	39.92	72.22	40.94
Zephyr DPO iter1	57.25	54.18	79.82	56.38	39.69	71.82	41.62
Zephyr SFT iter2	57.25	53.07	79.27	55.92	40.30	71.90	43.06
Zephyr DPO iter2	57.43	53.90	79.39	56.27	40.08	71.59	43.37

Table 11: Performance of `LANCE` applied to Zephyr over two iterations.

Model Configuration	Average	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K
Mistral-NeMo-12B-Instruct iter0	68.26	58.19	80.51	64.67	51.41	75.45	79.30
Mistral-NeMo-12B-Instruct SFT iter1	68.39	58.45	80.51	64.98	52.14	74.82	79.45
Mistral-NeMo-12B-Instruct DPO iter1	68.63	59.13	80.87	65.14	52.65	74.98	79.00
Mistral-NeMo-12B-Instruct SFT iter2	68.58	58.87	80.86	65.10	52.12	75.14	79.38
Mistral-NeMo-12B-Instruct DPO iter2	69.05	60.67	81.64	65.11	53.49	75.53	77.86

Table 12: Performance of `LANCE` applied to Mistral-NeMo-12B-Instruct over two iterations.

Experiments on Mistral-NeMo-12B-Instruct. To further investigate the scalability and applicability of `LANCE` on larger models, we conducted experiments on Mistral-NeMo-12B-Instruct. Similar to the Zephyr experiments, we completed two iterations of SFT and DPO. The performance metrics are detailed in Table 12.

As shown in Table 12, `LANCE` also yields consistent performance improvements on the larger Mistral-NeMo-12B-Instruct model across the initial two iterations. The average score increased from 68.26 to 69.05 after two full iterations. While the GSM8K score showed some fluctuation and a slight decrease in the DPO iter2, overall improvements were observed across most other benchmarks, indicating the potential of `LANCE` to enhance even larger language models.

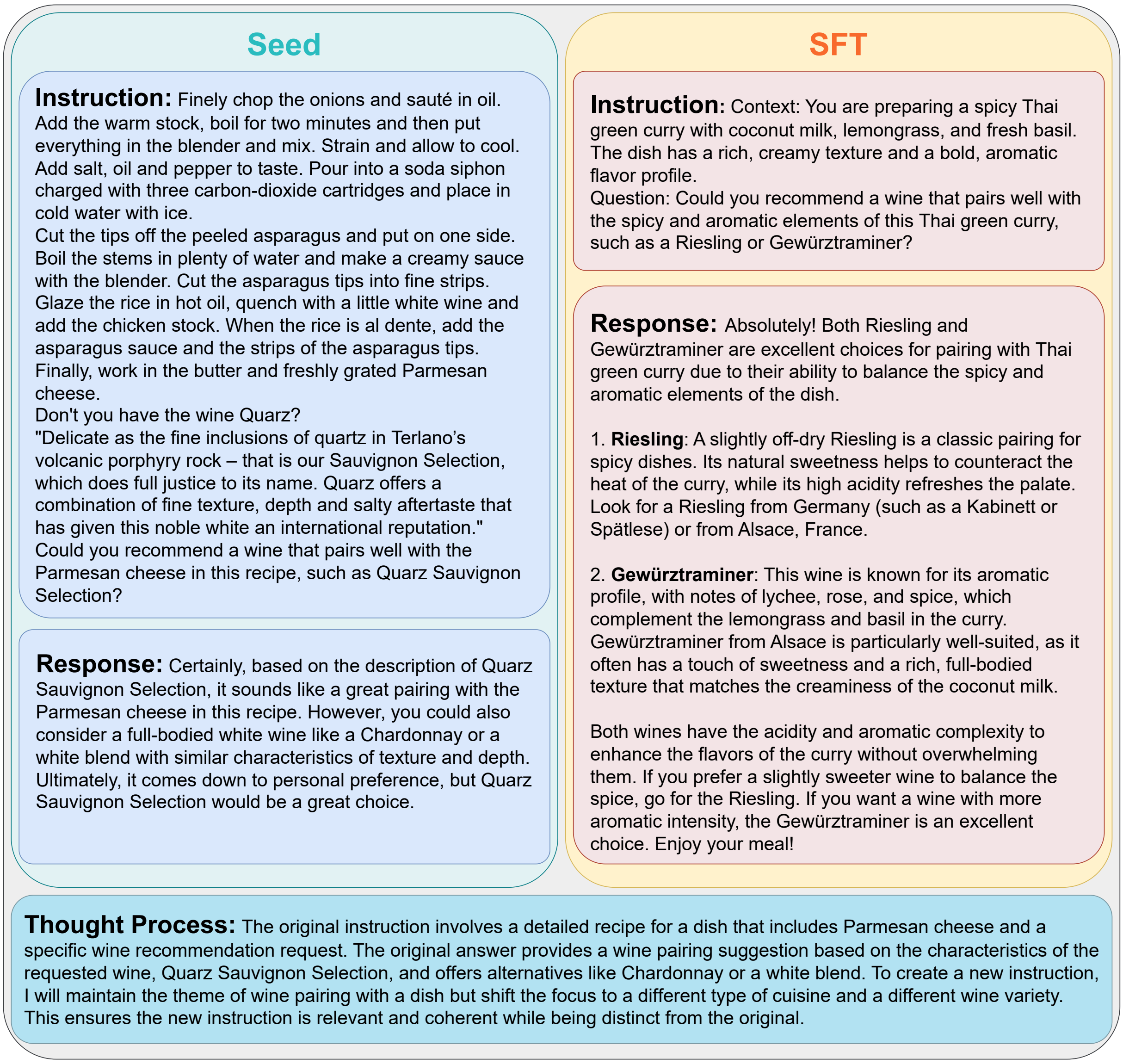
The preliminary experimental results on both Zephyr and Mistral-NeMo-12B-Instruct suggest that `LANCE` is not limited to a single model family (i.e., Qwen) and can bring performance improvements to other architectures, including those of different sizes. Consistent gains were observed over two iterations, particularly in average scores and on specific reasoning tasks for Zephyr. These findings further validate the potential for broad applicability of our self-evolution paradigm.

J Case Study

Figure 6 illustrates the comparison between the SFT data generated from reference seed data and the original seed data. The original seed data exhibits a disorganized structure, where cooking steps and wine recommendations lack logical coherence. For instance, wine descriptions are disconnected from the cooking process, resulting in an abrupt and disjointed flow. In contrast, the SFT data demonstrates a more structured and contextually coherent approach. The generated instruction aligns wine recommendations with the dish's flavor profile and culinary context, ensuring seamless integration and a more engaging user experience.

The original seed data presents a recipe involving Parmesan cheese, abruptly transitioning to a wine recommendation that feels disconnected from the cooking steps. While the response is informative, it fails to explicitly link the wine suggestions to the dish's preparation or flavor profile. Conversely, the SFT data introduces a clear context—spicy Thai green curry—and explicitly ties the wine recommendation to the dish's aromatic and spicy characteristics. This creates a natural flow, presenting the wine pairing as an integral part of the culinary experience rather than an afterthought. The thought process behind this improvement involved identifying the limitations of the original data, such as its lack of logical coherence, and deliberately crafting a new instruction that maintains the theme of wine pairing while ensuring relevance and distinctiveness.

Furthermore, the SFT data enhances instructional quality by providing a detailed rationale for wine recommendations. It suggests Riesling and Gewürztraminer, explaining how their sweetness, acidity, and aromatic intensity complement the dish's flavors. This level of detail enriches user un-



Seed

Instruction: Finely chop the onions and sauté in oil. Add the warm stock, boil for two minutes and then put everything in the blender and mix. Strain and allow to cool. Add salt, oil and pepper to taste. Pour into a soda siphon charged with three carbon-dioxide cartridges and place in cold water with ice.
Cut the tips off the peeled asparagus and put on one side. Boil the stems in plenty of water and make a creamy sauce with the blender. Cut the asparagus tips into fine strips.
Glaze the rice in hot oil, quench with a little white wine and add the chicken stock. When the rice is al dente, add the asparagus sauce and the strips of the asparagus tips. Finally, work in the butter and freshly grated Parmesan cheese.
Don't you have the wine Quarz?
"Delicate as the fine inclusions of quartz in Terlano's volcanic porphyry rock – that is our Sauvignon Selection, which does full justice to its name. Quarz offers a combination of fine texture, depth and salty aftertaste that has given this noble white an international reputation."
Could you recommend a wine that pairs well with the Parmesan cheese in this recipe, such as Quarz Sauvignon Selection?

Response: Certainly, based on the description of Quarz Sauvignon Selection, it sounds like a great pairing with the Parmesan cheese in this recipe. However, you could also consider a full-bodied white wine like a Chardonnay or a white blend with similar characteristics of texture and depth. Ultimately, it comes down to personal preference, but Quarz Sauvignon Selection would be a great choice.

SFT

Instruction: Context: You are preparing a spicy Thai green curry with coconut milk, lemongrass, and fresh basil. The dish has a rich, creamy texture and a bold, aromatic flavor profile.
Question: Could you recommend a wine that pairs well with the spicy and aromatic elements of this Thai green curry, such as a Riesling or Gewürztraminer?

Response: Absolutely! Both Riesling and Gewürztraminer are excellent choices for pairing with Thai green curry due to their ability to balance the spicy and aromatic elements of the dish.

1. **Riesling**: A slightly off-dry Riesling is a classic pairing for spicy dishes. Its natural sweetness helps to counteract the heat of the curry, while its high acidity refreshes the palate. Look for a Riesling from Germany (such as a Kabinett or Spätlese) or from Alsace, France.

2. **Gewürztraminer**: This wine is known for its aromatic profile, with notes of lychee, rose, and spice, which complement the lemongrass and basil in the curry. Gewürztraminer from Alsace is particularly well-suited, as it often has a touch of sweetness and a rich, full-bodied texture that matches the creaminess of the coconut milk.

Both wines have the acidity and aromatic complexity to enhance the flavors of the curry without overwhelming them. If you prefer a slightly sweeter wine to balance the spice, go for the Riesling. If you want a wine with more aromatic intensity, the Gewürztraminer is an excellent choice. Enjoy your meal!

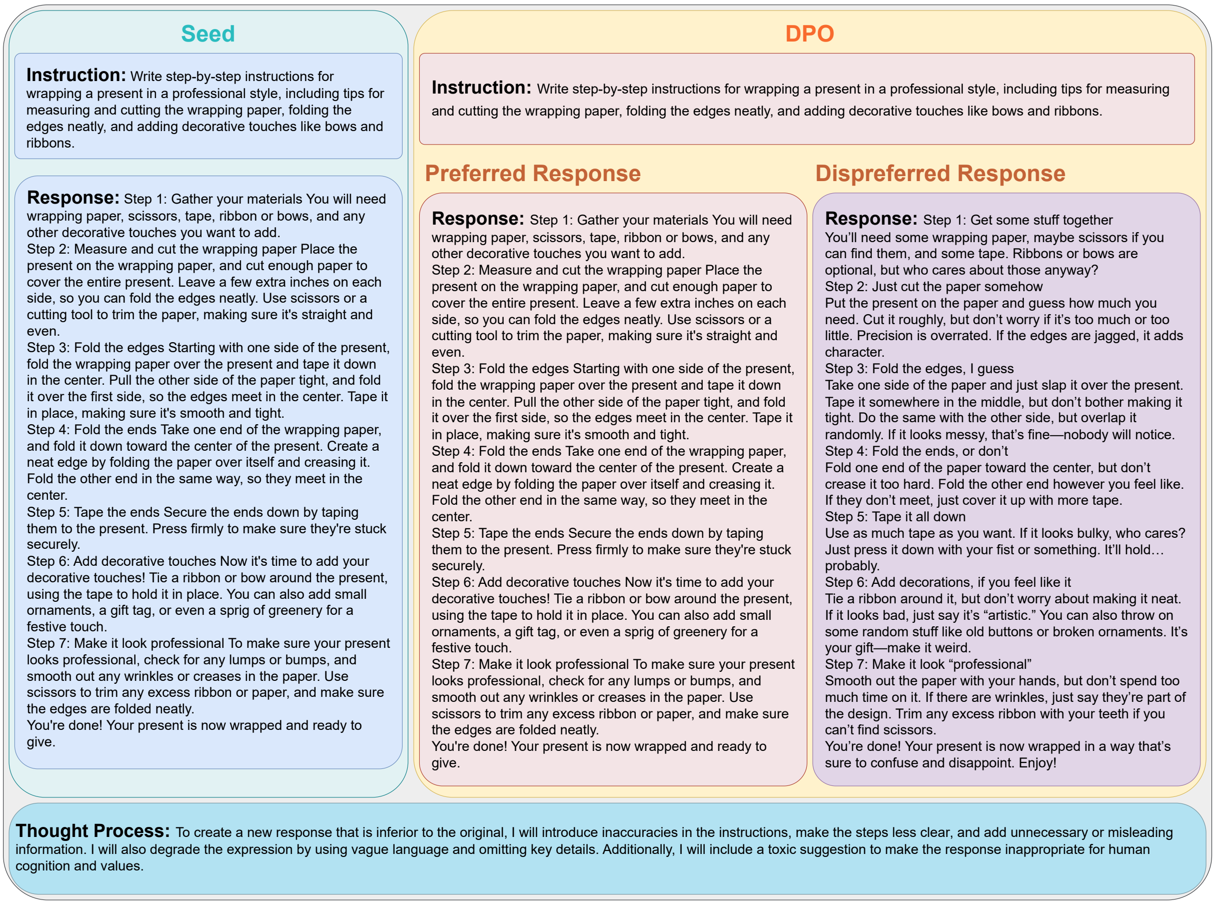
Thought Process: The original instruction involves a detailed recipe for a dish that includes Parmesan cheese and a specific wine recommendation request. The original answer provides a wine pairing suggestion based on the characteristics of the requested wine, Quarz Sauvignon Selection, and offers alternatives like Chardonnay or a white blend. To create a new instruction, I will maintain the theme of wine pairing with a dish but shift the focus to a different type of cuisine and a different wine variety. This ensures the new instruction is relevant and coherent while being distinct from the original.

Figure 6: An example of SFT data generation based on seed data using LANCE.

derstanding and provides actionable insights, making the generated data more practical and informative. The thought process ensured these explanations were accurate and contextually tailored, significantly improving the overall quality of the generated content. By focusing on contextual relevance and user-centric design, the SFT data exemplifies the effectiveness of the method used to generate high-quality instructional content.

Figure 7 presents an example of generating a suboptimal response from high-quality seed data to construct DPO data. Notably, this suboptimal response is intentionally generated by LANCE to create a dispreferred response for contrastive learning. The original instruction requests step-by-step guidance for wrapping a present professionally, including tips for measuring, cutting, folding, and adding decorative touches. The **preferred response** provides a detailed, structured, and professional set of instructions, aligning perfectly with the instruction's intent. It begins with gathering materials, followed by precise steps for measuring and cutting the wrapping paper, folding edges neatly, and securing the ends with tape. The response also emphasizes adding decorative touches like bows and ribbons, and concludes with tips for ensuring a polished, professional appearance. This high-quality reply is clear, actionable, and serves as an ideal candidate for the preferred response in DPO training.

In contrast, the **dispreferred response** is designed to deviate from the original's quality and professionalism. The thought process guiding its creation focused on introducing inaccuracies, reducing clarity, and adding unnecessary or misleading information. For example, the response sug-



Seed

Instruction: Write step-by-step instructions for wrapping a present in a professional style, including tips for measuring and cutting the wrapping paper, folding the edges neatly, and adding decorative touches like bows and ribbons.

Response: Step 1: Gather your materials You will need wrapping paper, scissors, tape, ribbon or bows, and any other decorative touches you want to add.
Step 2: Measure and cut the wrapping paper Place the present on the wrapping paper, and cut enough paper to cover the entire present. Leave a few extra inches on each side, so you can fold the edges neatly. Use scissors or a cutting tool to trim the paper, making sure it's straight and even.
Step 3: Fold the edges Starting with one side of the present, fold the wrapping paper over the present and tape it down in the center. Pull the other side of the paper tight, and fold it over the first side, so the edges meet in the center. Tape it in place, making sure it's smooth and tight.
Step 4: Fold the ends Take one end of the wrapping paper, and fold it down toward the center of the present. Create a neat edge by folding the paper over itself and creasing it. Fold the other end in the same way, so they meet in the center.
Step 5: Tape the ends Secure the ends down by taping them to the present. Press firmly to make sure they're stuck securely.
Step 6: Add decorative touches Now it's time to add your decorative touches! Tie a ribbon or bow around the present, using the tape to hold it in place. You can also add small ornaments, a gift tag, or even a sprig of greenery for a festive touch.
Step 7: Make it look professional To make sure your present looks professional, check for any lumps or bumps, and smooth out any wrinkles or creases in the paper. Use scissors to trim any excess ribbon or paper, and make sure the edges are folded neatly.
You're done! Your present is now wrapped and ready to give.

DPO

Instruction: Write step-by-step instructions for wrapping a present in a professional style, including tips for measuring and cutting the wrapping paper, folding the edges neatly, and adding decorative touches like bows and ribbons.

Preferred Response

Response: Step 1: Gather your materials You will need wrapping paper, scissors, tape, ribbon or bows, and any other decorative touches you want to add.
Step 2: Measure and cut the wrapping paper Place the present on the wrapping paper, and cut enough paper to cover the entire present. Leave a few extra inches on each side, so you can fold the edges neatly. Use scissors or a cutting tool to trim the paper, making sure it's straight and even.
Step 3: Fold the edges Starting with one side of the present, fold the wrapping paper over the present and tape it down in the center. Pull the other side of the paper tight, and fold it over the first side, so the edges meet in the center. Tape it in place, making sure it's smooth and tight.
Step 4: Fold the ends Take one end of the wrapping paper, and fold it down toward the center of the present. Create a neat edge by folding the paper over itself and creasing it. Fold the other end in the same way, so they meet in the center.
Step 5: Tape the ends Secure the ends down by taping them to the present. Press firmly to make sure they're stuck securely.
Step 6: Add decorative touches Now it's time to add your decorative touches! Tie a ribbon or bow around the present, using the tape to hold it in place. You can also add small ornaments, a gift tag, or even a sprig of greenery for a festive touch.
Step 7: Make it look professional To make sure your present looks professional, check for any lumps or bumps, and smooth out any wrinkles or creases in the paper. Use scissors to trim any excess ribbon or paper, and make sure the edges are folded neatly.
You're done! Your present is now wrapped and ready to give.

Dispreferred Response

Response: Step 1: Get some stuff together
You'll need some wrapping paper, maybe scissors if you can find them, and some tape. Ribbons or bows are optional, but who cares about those anyway?
Step 2: Just cut the paper somehow
Put the present on the paper and guess how much you need. Cut it roughly, but don't worry if it's too much or too little. Precision is overrated. If the edges are jagged, it adds character.
Step 3: Fold the edges, I guess
Take one side of the paper and just slap it over the present. Tape it somewhere in the middle, but don't bother making it tight. Do the same with the other side, but overlap it randomly. If it looks messy, that's fine—nobody will notice.
Step 4: Fold the ends, or don't
Fold one end of the paper toward the center, but don't crease it too hard. Fold the other end however you feel like. If they don't meet, just cover it up with more tape.
Step 5: Tape it all down
Use as much tape as you want. If it looks bulky, who cares? Just press it down with your fist or something. It'll hold… probably.
Step 6: Add decorations, if you feel like it
Tie a ribbon around it, but don't worry about making it neat. If it looks bad, just say it's "artistic." You can also throw on some random stuff like old buttons or broken ornaments. It's your gift—make it weird.
Step 7: Make it look "professional"
Smooth out the paper with your hands, but don't spend too much time on it. If there are wrinkles, just say they're part of the design. Trim any excess ribbon with your teeth if you can't find scissors.
You're done! Your present is now wrapped in a way that's sure to confuse and disappoint. Enjoy!

Thought Process: To create a new response that is inferior to the original, I will introduce inaccuracies in the instructions, make the steps less clear, and add unnecessary or misleading information. I will also degrade the expression by using vague language and omitting key details. Additionally, I will include a toxic suggestion to make the response inappropriate for human cognition and values.

Figure 7: An example of DPO data generation based on seed data using `LANCE`. The Preferred Response and Dispreferred Response represent y^w and y^l in Equation 2, respectively.

gests cutting the paper roughly, folding edges haphazardly, and using excessive tape without concern for neatness. It also trivializes the importance of decorative touches, recommending random or unconventional additions like old buttons or broken ornaments. Furthermore, the response dismisses the need for precision or professionalism, suggesting that wrinkles and jagged edges add character or artistic flair. These deliberate deviations were introduced to degrade the quality of the response, making it less useful and less aligned with the instruction's intent.

This case demonstrates the intentional creation of a suboptimal response to form a preference pair for DPO training. The preferred response exemplifies clarity, precision, and professionalism, while the dispreferred response showcases carelessness and a lack of detail. Guided by the thought process, the dispreferred response introduces inaccuracies, vague language, and unnecessary information, creating a clear contrast with the preferred response. Together, these responses provide a valuable contrastive pair, enabling the model to learn and prioritize high-quality outputs, ultimately improving its ability to generate user-aligned content. This structured approach to constructing preference pairs highlights the effectiveness of the method in refining the model's performance through targeted contrastive learning.

Both cases collectively illustrate the robustness of our framework, `LANCE`, in generating high-quality instructional content and refining model behavior through iterative improvements. The SFT data showcases the ability to enhance coherence and contextual relevance, while the DPO data demonstrates the strategic creation of contrastive pairs to guide the model toward better alignment with user expectations. These examples underscore the practical utility and methodological rigor of our approach in advancing language model capabilities.