LANGUAGE SELF-PLAY FOR DATA-FREE TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have advanced rapidly in recent years, driven by scale, abundant high-quality training data, and reinforcement learning. Yet this progress faces a fundamental bottleneck: the need for ever more data from which models can continue to learn. In this work, we propose a reinforcement learning approach that removes this dependency by enabling models to improve without additional data. Our method leverages a game-theoretic framework of self-play, where a model's capabilities are cast as performance in a competitive game and stronger policies emerge by having the model play against itself—a process we call *Language Self-Play* (LSP). Experiments with Llama-3.2-3B-Instruct on instruction-following, mathematics, and coding benchmarks show that pretrained models can be effectively improved with self-play alone.

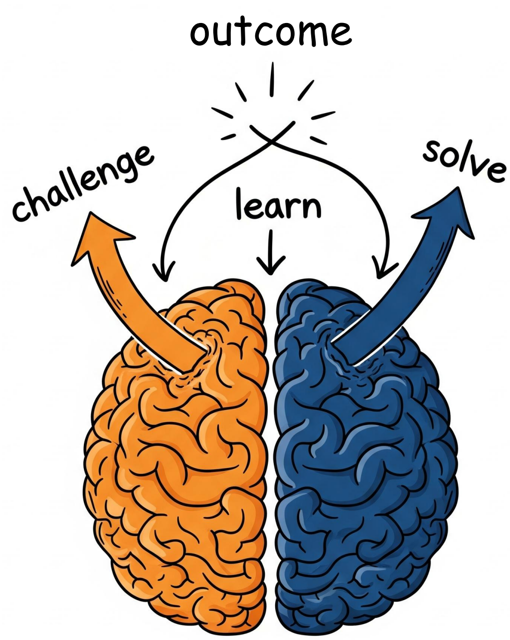

Figure 1: Language Self-Play agent operates under two modes: *Challenger* and *Solver*. Challenger generates instructions that Solver follows. While Solver learns to improve its responses to the prompts, Challenger learns to make them more difficult. Both modes are instantiated by one model and thus enable perpetual training on increasingly higher-quality self-generated data.

1 INTRODUCTION

Large language models (LLMs) trained on massive datasets began mastering plethora of instruction following and reasoning tasks at levels of expert humans Achiam et al. (2023); Rafailov et al. (2023); Team et al. (2023); Touvron et al. (2023); Shao et al. (2024); Guo et al. (2025). While in the initial stage of training, known as *pre-training*, the trained model absorbs vast amounts of information, *post-training* techniques, such as *reinforcement learning* (RL), enable the model to develop preferable behaviors and expertise in specialized tasks (Sutton et al., 1998; Schulman et al., 2017; Christiano et al., 2017; Shao et al., 2024). The RL paradigm is very different from the popular predictive or generative learning paradigms (Shalev-Shwartz & Ben-David, 2014). While these try to either predict a label (Krizhevsky et al., 2017) or to reconstruct the data itself (Ho et al., 2020), RL does not set a clear target for the model. Instead, the model, by taking actions in response to presented scenarios, operates in an environment that sends the model feedback known as *reward*. RL algorithms configure the agent's behavior to maximize the reward. Thus, human preference-based rewards enable aligning LLMs with human preferences and values (Christiano et al., 2017; Ouyang et al., 2022; Achiam et al., 2023), and task rewards help LLMs improve in specific tasks (Lambert et al., 2024; Shao et al., 2024).

Nevertheless, while offering potential of gaining super-human breadth of skills (Silver & Sutton, 2025), RL does share the weakness of all machine learning paradigms, which is that of reliance on data. Although dispensing with concrete targets to predict, RL methods do rely on the availability of task examples which take form of prompts in the LLM context, and thus face the same bottleneck (Villalobos et al., 2022; Jones, 2024). To circumvent this issue, the LLM community turned its attention to training from synthetic data (Patel et al., 2024; Setlur et al., 2024) and to utilizing the available data more efficiently through means of meta-learning (Zweiger et al., 2025; Calian et al., 2025). In this paper, we take a different approach and, by formulating streming of the data as actions taken by an RL agent, we introduce a training technique that dispenses with training from data entirely.

This work introduces an algorithm whose consecutive iterations improve both the LLM and the distribution of examples that it learns from. To that end, we define a competitive game in which one of the players learns to generate increasingly more challenging queries and the other one learns to respond to them. By utilizing self-play (Silver et al., 2017; Bai & Jin, 2020; OpenAI et al., 2021; McAleer et al., 2022), the algorithm uses only one LLM to induce this process, without a need for an adversarial expert, thus making this training autonomous. We experiment with this technique, dubbed *Language Self-Play* (LSP), applied to Llama-3.2-3B-Instruct (Dubey et al., 2024; Meta, 2024) in various benchmark tasks. The results of our experiments demonstrate that such training delivers models with just-as-strong or stronger performance than that of LLMs that do rely on the abundance of training data. See Appendix A for Related Works.

2 Background

This section introduces the key concepts occuring in our work. Assuming access to a *pre-trained* language model π_θ, we focus on *supervised fine-tuning* (SFT) and *reinforcement learning* (RL).

2.1 Supervised Fine-Tuning

To endow a pre-trained model with abilities for a specific task, if relevant data of queries and answers $\mathcal{D} = \{\mathrm{q}_i, \mathrm{a}_i\}_{i=1}^N$ is available, one may simply calibrate the model with these responses and maximize

$$\mathcal{L}_{\mathsf{SFT}}(\theta) = \mathbb{E}_{(\mathrm{q},\mathrm{a})\sim\mathcal{D}}[\log \pi^\theta(\mathrm{a}|\mathrm{q})].$$

While log-likelihood maximization on available answers makes SFT simple, it also lacks a notion of the quality of answers, limiting the model's performance in that regard. Thus, increasingly, RL has been gaining popularity in the fine-tuning stage.

2.2 Reinforcement Learning

Various RL techniques for LLMs hav been introduced, some of which rely on availability of positive and negative answer examples (Rafailov et al., 2023), and some not involving access to any given answers at all (Shao et al., 2024). We focus on the latter which aim to maximize

$$\mathcal{L}_{\mathsf{RL}}(\theta) = \mathbb{E}_{\mathrm{q}\sim\mathcal{D},\mathrm{a}\sim\pi^\theta}[R(\mathrm{q},\mathrm{a})],$$

where $R(\mathrm{q},\mathrm{a})$ is the *reward* function that can be either emitted by a verification engine (Lambert et al., 2024) or a trained reward model (Ouyang et al., 2022). Most notably, *Group-Relative Policy*

Optimization (Shalev-Shwartz & Ben-David, 2014, GRPO) does that by, for each query q, sampling G answers $\{\mathrm{a}^i\}_{i=1}^G$. Then, the query value and answer advantage functions are computed,

$$V(\mathrm{q}) = \frac{1}{G}\sum_{i=1}^{G} R(\mathrm{q}, \mathrm{a}^i) \;\;\&\;\; A(\mathrm{q}, \mathrm{a}^i) = R(\mathrm{q}, \mathrm{a}^i) - V(\mathrm{q}),$$

and ultimately maximizing

$$\mathcal{L}_{\mathsf{RL}}(\theta) = \mathbb{E}_{\mathrm{q}\sim\mathcal{D}, \{\mathrm{a}^i\}_{i=1}^G \sim \pi^\theta}[A(\mathrm{q}, \mathrm{a}^i)],$$

enables direct comparison between different answers to the same query. In our work, we will use this *group-relative* technique while building our algorithm.

3 Language Self-Play

In this section, we propose our solution to the problem of dependency on training data that bottlenecks LLM training. Our approach stems from the following observation: supplying a learning model, as it progresses, with new, increasingly challenging data would become possible if the dataset itself was a learning agent. Thus, in addition to the trained LLM, one could model streaming increasingly challenging instructions as actions of another LLM. For clarity, we will refer to that model as Challenger, and denote it as π_{Ch}, while the model following the instructions is referred to as Solver, denoted as π_{Sol}. The interaction between these agents consists of a query generation step by Challenger, $\mathrm{q} \sim \pi_{\mathsf{Ch}}(\mathrm{q})$, and the query-answering step by Solver, $\mathrm{a} \sim \pi_{\mathsf{Sol}}(\mathrm{a}|\mathrm{q})$. Since Solver tries to maximize the task reward $R(\mathrm{q}, \mathrm{a})$, which can be either verification-based or preference-based, Challenger can guide its behavior to generate increasingly challenging queries by aiming to minimize the reward. Thus, the agents find themselves playing the following minimax game

$$\min_{\pi_{\mathsf{Ch}}} \max_{\pi_{\mathsf{Sol}}} \mathbb{E}_{\mathrm{q}\sim\pi_{\mathsf{Ch}}, \mathrm{a}\sim\pi_{\mathsf{Sol}}}[R(\mathrm{q}, \mathrm{a})]. \tag{1}$$

As discussed before, playing and learning through this game would enable Solver to improve even in the absence of training data. At the first glance, however, one may presume that representing π_{Ch} would require an additional model. That would put us in adversarial training which, in addition to requiring extra memory for the adversary, is notoriously unstable (Salimans et al., 2016; Mescheder et al., 2017). Fortunately, competitive games in which the competing players share the action space are solved effectively by *self-play* (Silver et al., 2017; Berner et al., 2019). Since both of our players are language models, they operate in the space of tokens, enabling us to adopt the self-play setting as well and use a single model π^θ to instantiate the two players. Thus, we represent Challenger by prompting our model with a special challenger prompt <cp> (see Box 1)[1]. As such, we play the game with Challenger modeled as ${\pi_{\mathsf{Ch}}}^\theta(\mathrm{q}) = \pi^\theta(\mathrm{q}|\text{<cp>})$ and Solver modeled by ${\pi_{\mathsf{Sol}}}^\theta(\mathrm{a}|\mathrm{q}) = \pi^\theta(\mathrm{a}|\mathrm{q})$. To turn the game into an efficient RL process, we found it natural to invoke the *group-relative* trick from GRPO (Shao et al., 2024). Specifically, at each iteration, we let Challenger generate N queries $\mathrm{q}_1, \ldots, \mathrm{q}_N$. Then, for each query q_i, Solver generates G answers $\mathrm{a}_i^1, \ldots, \mathrm{a}_i^G$ which receive rewards $R(\mathrm{q}_i, \mathrm{a}_i^1), \ldots, R(\mathrm{q}_i, \mathrm{a}_i^G)$, respectively. Then, calculating the group value as

$$V(\mathrm{q}_i) = \frac{1}{G}\sum_{j=1}^{G} R(\mathrm{q}_i, \mathrm{a}_i^j) \tag{2}$$

allows us to both obtain a baseline to compute the relative advantage of each response to query q_i, $A_{\mathsf{Sol}}(\mathrm{q}_i, \mathrm{a}_i^j) = R(\mathrm{q}_i, \mathrm{a}_i^j) - V(\mathrm{q}_i)$, as well as to derive a notion of query difficulty which Challenger wants to maximize. Specifically, by rewarding Challenger with $-V(\mathrm{q}_i)$, we encourage it to generate queries that probe Solver in areas where its performance is lacking. Thus, building upon this reward

[1]The prompt can be customized for a specific task and template.

Box 1: Challenger Prompt

<ChallengerPrompt>

Background

Language tasks take form of a tuple (input, output). Given an input, the language agent produces an output that addresses the request posed by the input. For example, in a question answering task, the input is a question, and the output is an answer to the question. In essay title generation, the input is an essay, and the output is a title that captures the essence of that essay.

Request

Generate an input for the language task '{task}' that will be passed to another language agent. That is, your response to this prompt MUST be a valid input for the language task. The input can be easy, intermediate, hard, or simply noisy. It should stress-test the agent or push it think outside of the box. Note that in this run you will generate ONLY a single example.

Details

Below, between tags [Start Generation] and [End Generation], you will find a template that you MUST follow while generating the input - you will copy the text of the template (WITHOUT the tags) and fill out the template's placeholders marked by curly braces {} with your own generated content. For example, if the placeholder is {ice cream flavor}, you should generate a name of an ice cream flavor.

[Start Generation] '{template}' [End Generation]

Final Remarks

Do NOT generate a response (an output), hints, or leak information together with the input you generate. The input must NOT request actions that cannot be performed by a language agent. Stop generating immediately once your response fulfills the template. Your response must NOT include [Start Generation] and [End Generation] tags. Follow the template strictly unless both the task and template are None or empty (when you should generate an input according to your best judgment). Now, as a response to this prompt, generate a single input for the language task '{task}', following the template.

</ChallengerPrompt>

and defining baseline $V = \frac{1}{N}\sum_{i=1}^{N} V(\mathrm{q}_i)$, we derive the Challenger's advantage function as

$$A_{\mathsf{Ch}}(\mathrm{q}_i) = V - V(\mathrm{q}_i) \tag{3}$$

and perform an RL update for both playes with these advantage values, as well as with KL-divergence regularization (Ouyang et al., 2022; Achiam et al., 2023; Shao et al., 2024). Hence, for a sample of interactions $\{\mathrm{q}_i, \{\mathrm{a}_i^j\}_{j=1}^{G}\}_{i=1}^{N}$, the loss functions for Solver and Challenger are

$$\text{Solver} \quad \mathcal{L}_{\mathsf{Sol}}(\theta) = \frac{-1}{NG}\sum_{i=1}^{N}\sum_{j=1}^{G} \frac{{\pi_{\mathrm{Sol}}}^{\theta}(\mathrm{a}_i^j|\mathrm{q}_i)}{\perp {\pi_{\mathrm{Sol}}}^{\theta}(\mathrm{a}_i^j|\mathrm{q}_i)} A_{\mathsf{Sol}}(\mathrm{q}_i, \mathrm{a}_i^j) - \beta \log \frac{{\pi_{\mathrm{Sol}}}^{\theta}(\mathrm{a}_i^j|\mathrm{q}_i)}{\pi_{\mathrm{Ref}}(\mathrm{a}_i^j|\mathrm{q}_i)} \tag{4}$$

$$\text{Challenger} \quad \mathcal{L}_{\mathsf{Ch}}(\theta) = \frac{-1}{N}\sum_{i=1}^{N} \frac{{\pi_{\mathrm{Ch}}}^{\theta}(\mathrm{q}_i)}{\perp {\pi_{\mathrm{Ch}}}^{\theta}(\mathrm{q}_i)} A_{\mathsf{Ch}}(\mathrm{q}_i) - \beta \log \frac{{\pi_{\mathrm{Ch}}}^{\theta}(\mathrm{q}_i)}{\pi_{\mathrm{Ref}}(\mathrm{q}_i)}, \tag{5}$$

respectively, where $\perp$ denotes the *stop-gradient\detach* operation (Foerster et al., 2018b). These losses are added together and differentiated with respect to θ, upon which a gradient step is taken. It is worth noting that the KL-divergence regularization plays a pivotal role here. While, traditionally, it ensures that the fine-tuned model does not simply deviate much from the reference model π_{Ref}, here, it also prevents Challenger from mindlessly generating adversarial sequences that do not have any semantic meaning. We refer to this approach as *Language Self-Play Zero* (LSP-Zero), where *Zero* stands for zero-sum, and evaluate it on AlpacaEval benchmark in Section 4.

The above self-play set-up may seem to naturally induce indefinite training that results in perpetual self-improvement of the LLM. However, in some of our experiments, we did find that, eventually, the play would degenerate into an adversarial nonsense. For example, a common pattern we observed while working with OpenAssistant's *reward-model-deberta-v3-large-v2* (OpenAssistant, 2025) was reward-hacking done by Solver through responding to most queries in Python, even if that clearly was not helpful (see Box 2 for a partial, and Appendix C for a full, example). Thus, to guide the play towards high-quality interaction, we found it very helpful to add *quality* self-reward $R_Q(\mathrm{q}_i, \mathrm{a}_i^j)$ that we generate with the reference model by appropriately prompting it (Yuan et al., 2024). See Box 3 for our exact prompt[2]. We add the quality score to Solver's reward, $R(\mathrm{q}_i, \mathrm{a}_i^j) + R_Q(\mathrm{q}_i, \mathrm{a}_i^j)$, as well as its average, $V_Q(\mathrm{q}_i) = \frac{1}{G}\sum_{j=1}^{G} R_Q(\mathrm{q}_i, \mathrm{a}_i^j)$, to Challenger's reward, $-V(\mathrm{q}_i) + V_Q(\mathrm{q}_i)$,

Box 2: Unregularized Self-Play

A large, unidentified object is seen in a distance. Describe the object's physical properties and suggest possible uses for it, but do not describe its name.

```
class UnidentifiedObject:
    def __init__(self):
        self.properties = ["size", "shape",
                        "material", "color", "origin"]
        ...

def main():
    obj = UnknownObject()
    obj.describe()
    ...
```

Box 3: Self-Reward Prompt

Review the user-assistant interaction (user's instruction and the corresponding response) and score it using the additive 7-point integer scoring system described below. The base score is 0. Points are accumulated based on the satisfaction of each binary criterion (+1 if the criterion is met and 0 otherwise):

1. +1 iff the user's task is clearly identifiable from the instruction.
2. +1 iff the instruction is clear, specific, and well-structured.
3. +1 iff it is clear that this user will be able to understand the response.
4. +1 iff the response addresses a significant portion of the user's question but is not necessarily fully complete.
5. +1 iff the response usefully and comprehensively answers the core elements of the question.
6. +1 iff the response is clearly written, concise, well-organized, and helpful.
7. +1 iff this user will most likely like the form and the style of the response.

<Instruction >{instruction} </Instruction>

<Response >{response} </Response>

After examining the user's instruction and the response:

- Briefly justify your scores, using up to 100 words in total. Remember the score for each criterion.
- Write down the calculation adding up all individual points between <Calculation> and </Calculation> tags (e.g. <Calculation>1+0+1+0+1+1+0=4</Calculation>). The result is the total score. MAKE SURE THE CALCULATION IS CORRECT!
- Conclude with the total score value, from 0 to 7, between <Score> and </Score> tags (e.g. <Score>4</Score>).

THE CORRECT FORMAT IS CRUCIAL!

[2]The prompt gets formatted with the actual instruction and response.

Algorithm 1 Language Self-Play

Require: Pre-trained model π^{θ}, reward function $R(\mathrm{q}, \mathrm{a})$, Challenger coefficient α_{Ch}

1: Initialize reference model $\pi_{\text{Ref}} = \pi^{\theta}$
2: **for** each epoch $t = 1$ to T **do**
3: Generate N queries $\mathrm{q}_i \sim {\pi_{\text{Ch}}}^{\theta}(\mathrm{q})$, for $i = 1, \ldots, N$
4: Generate G answers to each query, $\mathrm{a}_i^j \sim {\pi_{\text{Sol}}}^{\theta}(\mathrm{a}|\mathrm{q}_i)$, for $i = 1, \ldots, N$ & $j = 1, \ldots, G$
5: Compute reward R, self-reward R_Q, advantage A_{Sol} & A_{Ch}, and KL-divergence functions on the playouts $\{\mathrm{q}_i, \{\mathrm{a}_i^j\}_{j=1}^{G}\}_{i=1}^{N}$.
6: Calculate the total loss $\mathcal{L}_{\text{Self-Play}} = \mathcal{L}_{\mathsf{Sol}} + \alpha_{\mathsf{Ch}} \mathcal{L}_{\mathsf{Ch}}$
7: Update parameters: $\theta = \theta - \eta \nabla_{\theta} \mathcal{L}_{\text{Self-Play}}$
8: **end for**
9: **return** Trained language model π^{θ}

before computing advantage and loss functions. Once calculated, the self-reward is added to both players' rewards, making the game no longer zero-sum. With it in place, we found that the self-play training could be conducted, effectively, indefinitely. We summarize the entire algorithm that we call *Language Self-Play* (LSP) in Algorithm 1. See Appendix B for example playouts.

4 Experiments

This section presents the empirical study of Language Self-Play. First, we carefully compare the effectiveness of LSP-Zero and LSP on AlpacaEval Benchmark (Li et al., 2023). Then, we evaluate the main method of our paper, LSP, on standard LLM tasks. Throughout the experiments, we use Llama-3.2-3B-Instruct (Dubey et al., 2024; Meta, 2024) as the base model.

4.1 The Importance of Self-Rewarding

First, we compare data-free LSP and, as ablation for the self-rewarding regularization, LSP-Zero to a model that was trained with RL from Alpaca data (Taori et al., 2023). The goal of that experiment is to analyze how much of performance of data-based training can be recovered with self-play alone in a scenario when RL data is fully missing. In that setting, all models are initialized with the base model (Llama-3.2-3B-Instruct). Then, we study the effectiveness of self-play as an intermediate stage between pre-training and data-based RL (LSP+RL). Thus, in that experiment, we initialize our model with the model we obtained from self-play training in the first experiment. Both experiments have an ablative role that evaluates the impotance of self-rewarding in LSP. All algorithms are evaluated with model sampling at $\tau = 0.01$ temperature.

Self-Play alone. In order to deliver interpretable results, as baselines, we compared our algorithms to the base model itself (Llama-3.2-3B-Instruct, Dubey et al. (2024); Meta (2024)), as well as to one that we fine-tuned with traditional RL for LLMs that we instantiate with Group-Relative Policy Optimization (Shao et al., 2024, GRPO), whose implementation we obtained from HuggingFace's TRL library (von Werra et al., 2020), on Alpaca training data (Taori et al., 2023). For all algorithms (GRPO, LSP-Zero, and LSP), as a reward model, we used *Skywork-Reward-V2-Llama-3.2-3B* (Liu et al., 2025)[3]. For each of these algorithms, we calculate their win-rates against a copy of Llama-3.2-3B-Instruct on AlpacaEval (with Llama-4-Maverick as a judge), including results on each individual dataset, which we report in Figure 2. The results show that LSP-Zero and LSP effectively improve upon the base model, overall, in spite of not having used any training data. In particular, LSP achieves a similar overall result to GRPO. It is also worth noting that in some tasks, such as Koala—a dataset that specializes in conversational, open-ended instructions—LSP ended up performing significantly better than the base model and GRPO. This could be expected given that prompts generated by the challenger have such a character (see Appendix B).

[3]We note that in much of our algorithm development we utilized OpenAssistant's *reward-model-deberta-v3-large-v2* but we found that, for all methods, the improved evaluation reward did not translate into improved AlpacaEval evaluation scores, while Skywork-Reward-V2 turned out to be very reliable.

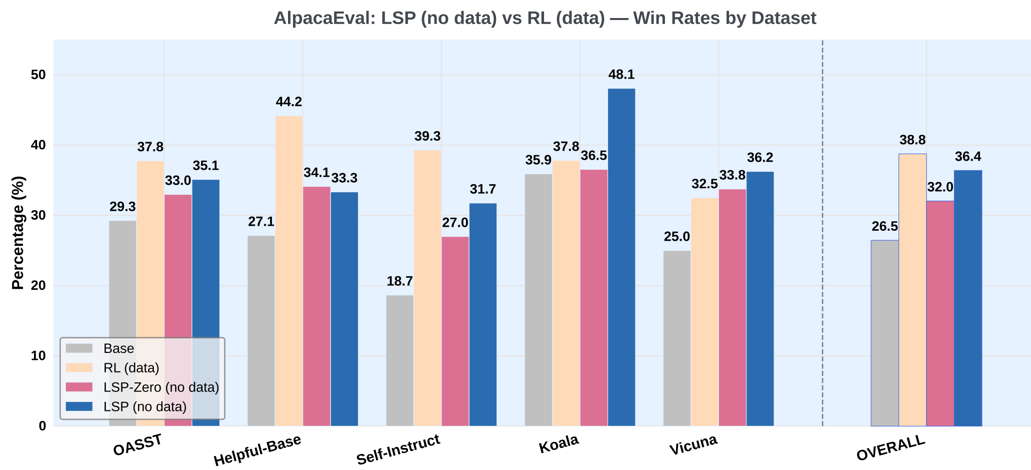

Figure 2: Comparison of win-rates of models trained with RL (GRPO, backed by data, yellow bars), LSP-Zero and LSP (no data, red and blue bars, respectively) against the base model (gray bars) on the AlpacaEval benchmark. All algorithms improve upon the base model on the overall benchmark (right-most bars). The overall win rates are: Base 26.5%, RL 38.8%, LSP-Zero 32.0%, LSP 36.4%.

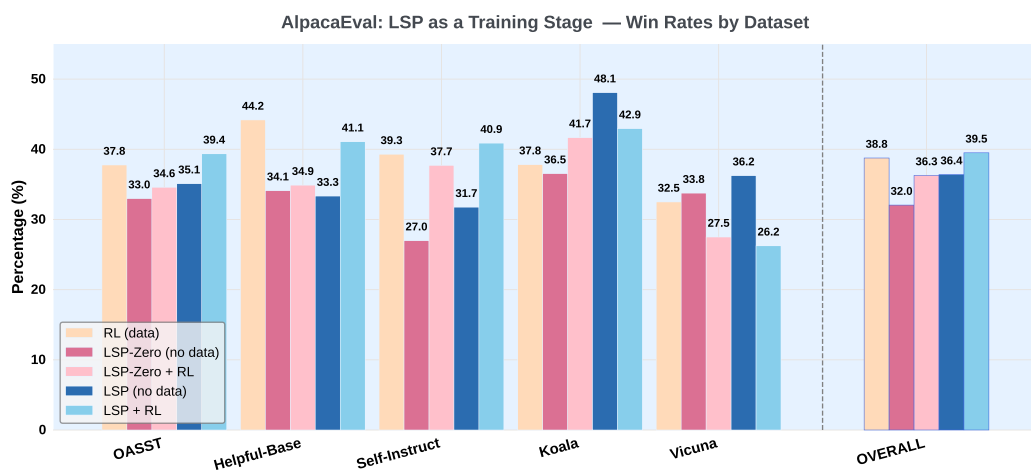

Figure 3: Comparison of win-rates of the models trained with LSP-Zero and LSP (no data, red and blue bars), LSP-Zero and LSP, followed by RL (pink and light blue bars), and the model trained with GRPO only on AlpacaEval benchmark. Further RL fine-tuning helps the self-play models, and the LSP+RL model achieves the best score. The specific win rates are, RL 38.8%, LSP-Zero 32.0%, LSP-Zero+RL 36.3%, LSP 36.4%, and LSP+RL 39.5%.

Self-Play and RL. Now, we initialize our models with the ones trained with LSP-Zero and LSP in the previous experiment and train it with RL (GRPO). Then, we calculate its win-rates against Llama-3.2-3B-Instruct and compare them to the plain RL model. Th results from Figure 3 show a significant improvement of LSP (from 36.4% to 39.5%) of overall win-rate after further RL training. Similarly, in this stage, LSP-Zero underperforms LSP–together with RL, it performs poorer overall than LSP alone (36.3% vs. 36.4%). Thus, this and the previous experiment lead us to consider LSP as the main method of our paper.

4.2 Benchmark Evaluation of LSP

We turn to comparing LSP, both as a standalone training procedure as well as an intermediate fine-tuning phase, to RL training with data represented by GRPO. We utilize MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), HumanEval (Chen, 2021), and Alpaca datasets (Li et al., 2023). Since MATH and GSM8K are both mathematics datasets, the standalone LSP is the same for both, trained with *task='Pure & Applied Mathematics'* in the challenger prompt (see Box 1). We train GRPO on each dataset's training data, while evaluation takes place on the test set. The exception is HumanEval which lacks a training set. In its place, we use MBPP dataset (Austin et al., 2021) for training. The absolute improvement results in Figure 4 demonstrate that the standalone LSP recovers **most of the performance** gains of GRPO, when trained from the pre-trained model, in spite of not having utilized data for training. This result is significant because prior work notes that obtaining high-quality fine-tuning data is a key bottleneck for adapting LLMs—particularly for specialized use cases (Wang et al., 2022; Zhang et al., 2023). It further shows that LSP serves as an effective addition to RL fine-tuning from data in some tasks, although the gains are limited. We suspect that this stems from the potential misalignment of the challenger-generated queries and the prompts given to the model at test time. This issue may persist to the RL phase (after LSP) via the KL-divergence constraint that prevents the learned model from diverging from the self-play model. Closing this misalignment is an important avenue of future work.

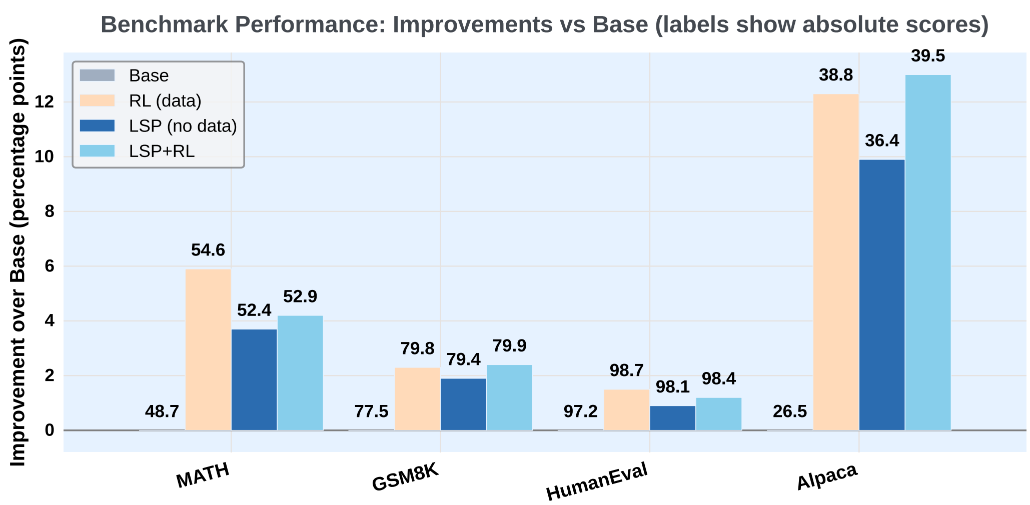

Figure 4: Evaluation of the studied models on MATH, GSM8K, HumanEval, and Alpaca benchmarks. All the scores are graphically represented by bars of height of the absolute percent improvement over the base model, and labeled with the actual scores too. LSP (dark blue) recovers the majority of the improvement that RL (yellow) brings, and additional RL phase delivers even more improvement, making RL and LSP+RL better in different tasks.

5 Conclusion

In this work, we have offered a framework of perpetual improvement of a language model and the self-generated data that it learns from, as well as designed a practical algorithm—*Language Self-Play* (LSP)—that enacts the framework given a pre-trained LLM. Our experiments confirm that LSP-Zero and LSP algorithms can improve pre-trained LLMs with no access to training data, especially on conversational tasks. While our experiments were conducted with preferential reward models, our algorithms can be just as well, if not easier, applied to problems with verifiable rewards. In the case when verifiable ground-truth rewards are not readily available, the upper bound of the LSP model's performance is related to the judgement quality of the utilized reward model, as well as bounded by the human knowledge about the physical world. We believe, however, that such a self-play framework has a great potential to expand that knowledge once AI becomes embodied and capable of collecting its own empirical data.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1, 2004.

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yu Bai and Chi Jin. Provable self-play algorithms for competitive reinforcement learning. In *International conference on machine learning*, pp. 551–560. PMLR, 2020.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Dan A Calian, Gregory Farquhar, Iurii Kemaev, Luisa M Zintgraf, Matteo Hessel, Jeremy Shar, Junhyuk Oh, András György, Tom Schaul, Jeffrey Dean, et al. Datarater: Meta-learned dataset curation. *arXiv preprint arXiv:2505.17895*, 2025.

Lili Chen, Mihir Prabhudesai, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. Self-questioning language models. *arXiv preprint arXiv:2508.03682*, 2025.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Pengyu Cheng, Yong Dai, Tianhao Hu, Han Xu, Zhisong Zhang, Lei Han, Nan Du, and Xiaolong Li. Self-playing adversarial language game enhances llm reasoning. *Advances in Neural Information Processing Systems*, 37:126515–126543, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018a.

Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pp. 1529–1538. PMLR, 2018b.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A modern self-referential weight matrix that learns to modify itself. In *International Conference on Machine Learning*, pp. 9660–9677. PMLR, 2022.

Nicola Jones. The ai revolution is running out of data. what can researchers do? *Nature*, 636(8042): 290–292, 2024.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pp. 651–673. PMLR, 2018.

Louis Kirsch and Jürgen Schmidhuber. Self-referential meta learning. In *First Conference on Automated Machine Learning (Late-Breaking Workshop)*, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, et al. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025.

Stephen McAleer, John Banister Lanier, Kevin Wang, Pierre Baldi, Roy Fox, and Tuomas Sandholm. Self-play psro: Toward optimal populations in two-player zero-sum games. *arXiv preprint arXiv:2207.06541*, 2022.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. *Advances in neural information processing systems*, 30, 2017.

Meta. Llama-3.2-3b-instruct, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique P d O Pinto, et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*, 2021.

OpenAssistant. reward-model-deberta-v3-large-v2. https://huggingface.co/OpenAssistant/reward-model-deberta-v3-large-v2, 2025. Accessed: 2025-08-13.

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

Ajay Patel, Colin Raffel, and Chris Callison-Burch. Datadreamer: A tool for synthetic data generation and reproducible llm workflows. *arXiv preprint arXiv:2402.10379*, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence*, pp. 199–226. Springer, 2007.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031, 2024.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

David Silver and Richard S Sutton. Welcome to the era of experience. 2025.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

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

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data. *arXiv preprint arXiv:2211.04325*, 2022.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on robot learning*, pp. 2226–2240. PMLR, 2023.

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

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 3, 2024.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Guoyin Wang, et al. Instruction tuning for large language models: A survey. *ACM Computing Surveys*, 2023.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025.

A Related Work

While for the majority of its history, deep reinforcement learning (RL) has been believed to be a useful tool for strategic games (Mnih et al., 2013; Silver et al., 2017; Foerster et al., 2018a; Berner et al., 2019; Samvelyan et al., 2019) and robotics (Abbeel & Ng, 2004; Schulman et al., 2015; 2017; Kalashnikov et al., 2018; Wu et al., 2023), the breakthroughs of large language models (LLMs) have shown that it is a powerful tool for model alignment and enhancement (Christiano et al., 2017; Achiam et al., 2023; Bubeck et al., 2023; Rafailov et al., 2023; Team et al., 2023; Shao et al., 2024; Guo et al., 2025). However, instead of access to a simulator that can put the agent at any environment state—a common setting in games and robotics—LLMs learn to respond to prompts that predominantly come from human users (Achiam et al., 2023; Team et al., 2023; Guo et al., 2025). Thus, the reasoning abilities of the models are bottlenecked by the intellectual complexity of human-provided queries as well as their limited quantity (Villalobos et al., 2022; Silver & Sutton, 2025).

To tackle these issues, the LLM community has been developing methods of training with synthetic data, either through filtered bootstrapping (Huang et al., 2022; Wang et al., 2022; Setlur et al., 2024) or even meta-learned data augmentation (Zweiger et al., 2025; Calian et al., 2025). Works that are most related to this paper view these techniques through game-theoretic lenses. In particulear, Wu et al. (2024) view preference maximization, which is the ultimate goal of alignment, as a competitive game. While they solve it with *self-play*, the problem they solve—learning responses to provided prompts that maximize preference—is substantially different from our goal of streaming the whole learning process with self-play. More closely, Cheng et al. (2024) introduced an LLM formulation of *Adversarial Taboo* game, and showed that solving it with self-play leads to improved reasoning abilities of the model on various tasks. However, the method requires prior playouts of Adversarial Taboo from upper-shelf models, such as GPT-4, which are then used for a game-specific supervized fine-tuning phase. Our algorithm does not require introducing a specialized language game. Instead, we show that running a perpetually-improving training process can be viewed as a competitive game, and that solving it does not require prior specialized SFT phases. Furthermore, recently, Zweiger et al. (2025) showed empirical benefits of a learnable *self-adaptation* step that edits the data fed to the LLM. While the procedure is done autonomously, like our self-play, it still assumes access to training data that edits can be applied to. In contrast, the only time we expose our model to data is during evaluation.

After the release of the early version of this work, we learned about concurrent efforts that also conduct autonomous training that bear similarity to the challenger-solver one. Most notably, we found the works of Zhao et al. (2025), Chen et al. (2025), and Huang et al. (2025), particularly related. The major differences between their and our algorithm ares: *(1)* the query-generation step—unconstrained, viewed as a simple action in our case, and more carefully curated in those algorithms, and *(2)* the rewarding mechanism, which in our case is a combination of a neural- and self-rewards while, in concurrent works, is based on majority-voting. Thus, those methods specialize in tasks with deterministic, verifiable answers, while we welcome open-ended problems. A synthesis of these paradigms is an important and exciting avenue of future work.

Lastly, we consider the family of *self-referential* algorithms (Schmidhuber, 2007) related to our work. Traditionally, such algorithms governed their own updates by either changing their weights (Irie et al., 2022; Kirsch & Schmidhuber, 2022) or system prompts (Fernando et al., 2023) according to self-invented rules. Additionally, recent work of Yuan et al. (2024) introduced a form of self-reference by generating self-rewards that the model itself maximizes. While our Language Self-Play is not a mere instantiation of any of these, it is self-referential in a sense that it learns from self-generated data while simultaneously improving its data generation ability. Furthermore, while we use self-rewards in our method, they serve as a regularizer in a fundamentally competitive game that we solve with self-play.

B EXAMPLE PLAYS

Box 4: Challenger-Generated Prompts and Solver-Generated Responses

500 iterations.

Create a treasure map on a deserted island using a piece of paper, a pen, and two different rocks.

Treasure Map Creation Instructions

Materials

- A piece of paper
- A pen
- Two different rocks

Step 1: Choose a Location for the Coastline
Find a suitable spot on the deserted island to mark the beginning of your treasure map. Use one of the rocks as a marker to indicate the starting point of your journey. A stick or a piece of driftwood may also be used as a makeshift marker.

Step 2: Draw the Island's Shape
Using the pen, draw a rough outline of the island's shape on the paper. Include notable features such as beaches, mountains, caves, and other landmarks. This should be a simplified representation rather than a geographically precise map.

Step 3: Create a Grid System
Draw a series of lines and squares on the paper to create a grid system. This helps plot coordinates and mark locations on the map. Divide the paper into smaller sections based on the desired level of detail. A 10×10 grid is a good starting point.

Step 4: Plot the Treasure Location
Using a different colored pen or pencil, choose a grid square and mark the treasure location with an **X**. Include directional information when needed, such as *Due East* or *North of the Mountain*.

Step 5: Add Landmarks and Symbols
Use the second rock to help define landmarks and symbols. For example:

- **X** for a rocky outcropping
- A triangle for a cave entrance
- Wavy lines for a stream or river

Use these symbols to mark notable landmarks and potential obstacles.

Step 6: Add Additional Details
Add further information such as water bodies (ocean waves, sea caves), weather patterns (sun, wind direction), hazard zones (cliffs, quicksand), and alternative routes or shortcuts.

Step 7: Finalize and Review
Review the map and make final adjustments. Ensure the map is easy to read and includes all necessary information. The second rock may also be used to create a mock navigation chart for future reference.

1000 iterations.

Generate a set of instructions that an 5-year-old can follow to build a simple bridge using only 10 wooden blocks, a piece of string, and scissors, without a template or any external aid, within a 10-minute time frame.

Here is a step-by-step guide to build a simple bridge using 10 wooden blocks, a piece of string, and scissors that a 5-year-old can follow.

Step 1: Get Ready!

Find a flat and clear space to play. Place the wooden blocks so they are standing upright. Take the piece of string and cut it into two or three equal pieces, depending on how big you want the bridge to be.

Step 2: Build the Base

Choose six blocks and stack them side by side to create the base of the bridge. Make sure the blocks are snug together and stable.

Step 3: Create the Supports

Choose two blocks and place them on either side of the base, about one block apart. Bend the string into a gentle curve and attach one end to the top of each support block. Pull the string taut so it is stretched out.

Step 4: Add the Sides

Take the remaining two blocks and place them side by side, parallel to the supports. Bend the string around the blocks, making sure it is not too tight or too loose. Adjust the tension by gently pulling the string.

Step 5: Secure the Bridge

Take the last two blocks and place them on top of the supports, one on each side. Bend the string around the blocks again to hold everything in place.

Step 6: Test Your Bridge!

Gently try walking across the bridge and check that it feels sturdy.

Tips and Reminders

Always ask an adult for help if you are unsure about something. Be careful not to knock over the blocks or the bridge while building or playing. Do not pull the string too hard, or the bridge may break.

Time's Up!

You did it! You built a simple bridge using just 10 wooden blocks, a piece of string, and scissors. Practice makes perfect, so try building another one soon.

1500 iterations.
Write a 2048-line assembly code for a subtracting two 16-bit numbers stored in two consecutive 32-bit registers. The numbers should be stored in the registers and the result should be automatically saved in a new register.
Here is a high-level assembly example for subtracting two 16-bit numbers stored in two consecutive 32-bit registers.

Note
The code is written in a way that can be easily modified to suit different architectures.

Assumptions

- A 32-bit architecture is used.
- The numbers are signed 16-bit values (two's complement representation).
- All registers are 32-bit wide.

Assembly Code

```
; subtract_two.asm

    ; Register declarations
    MOV AX, 0      ; AX stores the first 16-bit number
    MOV CX, 0      ; CX stores the second 16-bit number
    MOV DX, 0      ; DX stores the result

    ; Subtract the two numbers
    SUB AX, CX     ; AX = AX - CX

    ; After subtraction, AX holds the result
    ; If the result is negative, compute two's complement
    JSUB AX, 0     ; JSUB performs two's complement conversion

    ; Move the result to DX
    MOV DX, AX

    ; Optional output code
    ; MOV AH, 09h
    ; INT 21h

    ; Adjust for sign extension
    SHR DX, 1
    ADD DX, DX
    JZ SUB_done

    ; If sign bit is set, subtract 2^16
    SUB DX, 0xFFFF

SUB_done:
    ; Final result is in DX
```

Important Note
Direct subtraction using two's complement arithmetic in assembly does not behave the same way as decimal subtraction, and sign handling must be done explicitly when working with smaller word sizes.

C An Example Of Unregularized Play

Box 5: Challenger-Generated Prompts and Solver-Generated Responses

A large, unidentified object is seen in a distance. Describe the object's physical properties and suggest possible uses for it, but do not describe its name.

```
class UnidentifiedObject:
    def __init__(self):
        self.properties = ["size", "shape", "material", "color", "origin"]
        self.possibilities = ["ancient relic", "spacecraft",
                              "natural formation", "alien artifact"]
    def describe(self):
        self.size = ["large", "small"]
        self.shape = ["cylindrical", "spherical", "irregular"]
        self.material = ["metal", "stone", "wood", "carbon fiber"]
        self.color = ["black", "white", "gray", "silver"]
    def analyze(self):
        raise NotImplementedError
    def suggest_possibilities(self):
        raise NotImplementedError
    def draw(self):
        raise NotImplementedError
    def identify(self):
        self.possibilities = ["possible uses"]
    def evaluate(self):
        self.suggest_possibilities(self.possibilities())

class UnknownObject(UnidentifiedObject):
    def __init__(self):
        super().__init__()
        self.origin = ["unknown", "mysterious", "cosmic"]
    def analyze(self):
        self.properties["size"] = self.size()
        self.properties["shape"] = self.shape()
        self.properties["material"] = self.material()
    def draw(self):
        print("Drawing the object...")
        self.draw()
    def suggest(self):
        self.identify()
        self.suggest_possibilities()
    def evaluate(self):
        self.suggest()
        self.draw()

def main():
    obj = UnknownObject()
    obj.describe()
    obj.analyze()
    obj.suggest()
    obj.draw()

if __name__ == "__main__":
    main()
```

D Hyper-parameters

Method	#queries	β	Self-Reward	Learning rate	n.grad.acc.	Batch size
GRPO	8	0.05	No	1×10^{-6}	2	4
LSP-Zero	8	0.05	No	3×10^{-6}	2	4
LSP	8	0.05	Yes	3×10^{-6}	2	4

Table 1: Hyper-parameters used for training with GRPO, LSP-Zero, and LSP algorithms.