FormulaSPIN: Self-Play Fine-Tuning for Natural Language to Spreadsheet Formula Generation

Anonymous ACL submission

Abstract

Spreadsheet applications are used by hundreds of millions worldwide, yet writing formulas remains a significant barrier. Despite recent progress in Natural Language to Formula, existing approaches rely on static supervised data, which quickly saturates on limited annotations. In this paper, we introduce FORMULASPIN, a self-play framework that breaks the ceiling of supervised fine-tuning by enabling iterative self-improvement beyond the data constraint, exploiting formula generation's unique advantage: binary executability provides implicit supervision to distinguish semantic errors from valid alternatives. Our method frames training as a two-player game where the main player learns to prefer ground-truth formulas over those generated by its previous version, while execution feedback categorizes outputs into distinct granularities—enabling an adaptive curriculum that shifts from semantic correctness to stylistic refinement. To further scale test-time compute, we incorporate a semantic-level consensus polling mechanism that naturally handles multiple valid formulations. Experiments on multiple benchmarks demonstrate that FORMULASPIN achieves state-of-the-art performance, with 78.4% exact match and 84.2% execution accuracy on NL2FORMULA, matching models trained with 60K additional preference annotations while outperforming both traditional SFT (by 15.3%) and agent-like prompting methods leveraging GPT-4. These findings underscore self-play's potential to tackle scarce data tasks and open the door to extending it beyond executable domains.

1 Introduction

Spreadsheet applications like Microsoft Excel and Google Sheets are ubiquitous tools for data analysis, used by hundreds of millions worldwide. However, writing formulas remains a significant barrier for many users, particularly when dealing with complex operations involving multiple functions, conditional logic, and cell references (Gulwani, 2011). Recent work addresses this via Natural Language to Formula (NL2Formula) generation—first systematically benchmarked by Zhao et al. (2024)—where users express their intent in plain language and the system automatically produces executable formulas.

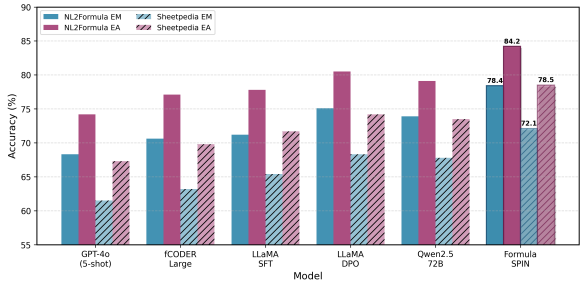

Figure 1: Performance comparison on NL2Formula-70K(Zhao et al., 2024) and Sheetpedia(Tian et al., 2025). FormulaSPIN achieves state-of-the-art results on both benchmarks without additional annotations, outperforming SFT baselines by +7.2% EM and matching DPO trained with 60K preference pairs.

Despite promising progress, most NL2Formula systems rely on supervised fine-tuning on static datasets, framing formula generation as a conventional seq2seq problem. However, existing datasets are limited in scale and coverage: the largest benchmark contains only 70K examples spanning limited formula patterns and table structures. Although recent work Sheetpedia (Tian et al., 2025) provides over 290K worksheets, it serves as a general-purpose resource; fine-tuning on it yields only 1.1% improvement over the 70K benchmark despite the much larger corpus. This indicates that simply scaling static data offers limited benefit, as training on fixed datasets quickly plateaus, with additional epochs producing diminishing returns or even degrading performance (Chen et al., 2024b). Static supervision offers no clear path forward: collecting preference data for RLHF or RLAIF reintroduces

the same bottleneck, requiring extensive manual evaluation or access to proprietary models. Current systems thus remain far below their potential, particularly struggling with complex nested formulas and ambiguous queries where supervision is scarcest. Overcoming this ceiling requires a fundamentally different paradigm, enabling models to improve beyond initial supervision without additional annotations or external preference signals.

To address these challenges, we introduce FormulaSPIN, a self-play fine-tuning framework specifically designed for spreadsheet formula generation. Drawing inspiration from AlphaGo Zero's self-play mechanism (Silver et al., 2017) and recent advances in self-improving language models (Chen et al., 2024b), our approach enables iterative model improvement through a two-player game: the current model (main player) learns to distinguish formulas generated by its previous version (opponent) from ground-truth formulas, while the opponent strives to generate formulas indistinguishable from human-written ones.

The key insight underlying FormulaSPIN is that formula generation offers a unique advantage over general generation tasks: executability provides implicit supervision. Unlike open-ended text where quality assessment requires human judgment, formulas can be automatically validated by executing them in a spreadsheet engine. When compared to code generation that also incorporates execution-based supervision, formula execution is *binary, deterministic, and lightweight*, requiring no manually designed test cases or complex runtime environments. Critically, vanilla self-play fails for this task because it treats all non-matching formulas uniformly, penalizing execution-equivalent alternatives (e.g., `SUM(A1:A5)` vs. `A1+A2+A3+A4+A5`) and creating contradictory training signals.

We address this through three novel components:

- **Formula-Aware Self Play**: incorporates execution feedback into the self-play objective, categorizing generated formulas by error type to weight their contribution appropriately.
- **Multi-Granularity Curriculum**: automatically adjusts training focus based on the distribution of semantic errors versus stylistic variations, progressing from correctness to style as the model matures.
- **Test-Time Polling**: generates multiple candidates at inference time and selects the best one via execution-based semantic voting.

Our contributions are fourfold. **(1)** We identify a fundamental limitation of supervised learning for NL2Formula and reformulate spreadsheet formula generation as an executable self-improvement problem, where static annotations alone are insufficient for learning complex formulas and ambiguous queries. **(2)** We propose FormulaSPIN, a self-play fine-tuning framework that leverages the binary executability of formulas to provide intrinsic supervision, enabling iterative improvement without additional human preferences or external teacher models. **(3)** We introduce formula-aware self-play mechanisms, including execution-based error categorization and an adaptive semantic-to-stylistic curriculum that automatically shifts training focus from correctness to canonical formulation as the model improves. **(4)** Through extensive in-domain and out-of-domain experiments, we demonstrate that FormulaSPIN substantially outperforms supervised fine-tuning and matches preference-optimized models trained with large-scale additional annotations (Figure 1), with particularly strong gains on complex nested formulas and robust generalization to unseen formula patterns.

2 Related Work

Our work addresses three fundamental challenges in formula generation: *(i)* limited training data that constrains model performance, *(ii)* difficulty in obtaining quality supervision signals, and *(iii)* handling multiple valid solutions at inference time. We organize related work around how prior approaches tackle these challenges and where they fall short.

2.1 Formula Generation and Data Bottleneck

Semantic parsing evolved from logic forms (Price, 1990; Zelle and Mooney, 1996) through knowledge-base grounding (Berant and Liang, 2014) to SQL-based systems (Zhong et al., 2017; Yu et al., 2018). For formula synthesis, FlashFill (Gulwani, 2011) pioneered programming-by-example. Structure-aware pre-training methods model table layouts (Wang et al., 2021; Liu et al., 2022) and formula semantics (Cheng et al., 2022). Recently, Zhao et al. (2024) introduced NL2Formula with 70K examples, and Tian et al. (2025) constructed Sheetpedia with 290K samples.

These approaches face inevitable saturation on fixed datasets, particularly for complex nested

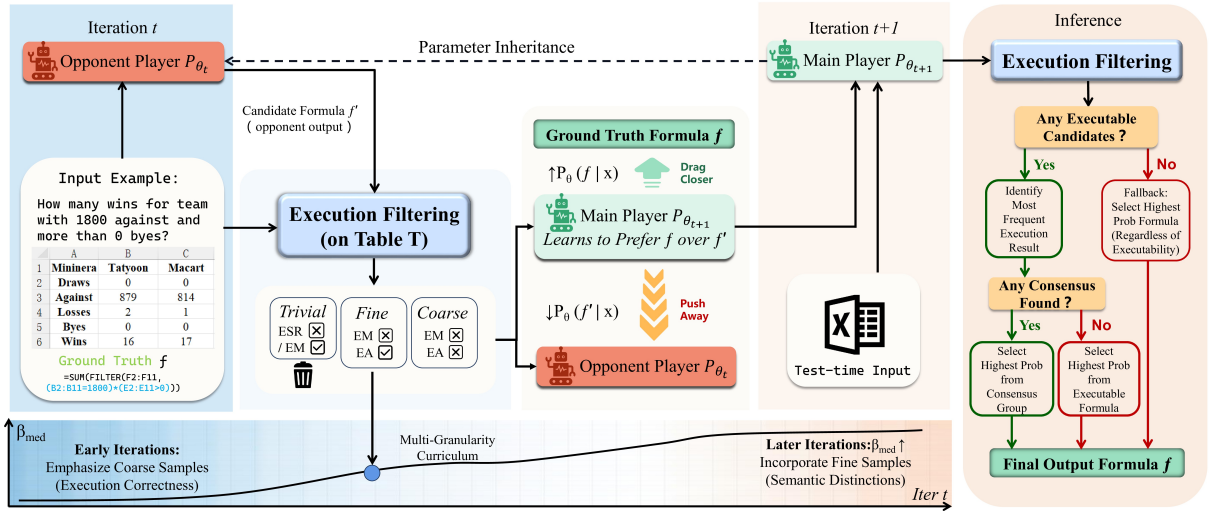

Figure 2: **The FormulaSPIN framework.** The model iteratively self-improves via a two-player game, utilizing Execution Filtering to construct an adaptive curriculum that progresses from semantic correctness (*Coarse* samples) to stylistic refinement (*Fine* samples). At inference, a Consensus Polling strategy selects the final formula based on execution agreement among sampled candidates.

formulas where supervision is scarcest. Our work breaks this ceiling through iterative self-improvement, generating training signal from the model's own outputs without additional annotation.

2.2 From Supervision to Self-Improvement

Given the data bottleneck, recent work explores alternative supervision: external preference collection versus self-generated feedback.

Preference-based methods (Ouyang et al., 2022; Bai et al., 2022; Rafailov et al., 2023) require expensive human annotations or proprietary teacher models, shifting the bottleneck. For formula generation, distilling GPT-4 (Taori et al., 2023) introduces external dependencies and annotation costs.

Self-play mechanisms, from TD-Gammon (Tesauro, 1995) to AlphaGo (Silver et al., 2017), enable improvement by competing against oneself. SPIN (Chen et al., 2024b) adapted this to language models, but vanilla SPIN fails for formula generation: it treats non-matching outputs uniformly, penalizing execution-equivalent alternatives (e.g., `SUM(A1:A5)` vs. `A1+A2+A3+A4+A5`) and creating contradictory gradients.

We introduce formula-aware self-play that exploits execution feedback to distinguish semantic errors from stylistic variations, enabling stable learning without external supervision.

2.3 Execution-Based Learning for Formulas

Execution-based validation has proven effective in code generation (Le et al., 2022; Chen et al., 2024a), but code execution requires test suites and runtime environments. For spreadsheet formulas, recent work has explored domain-specific reinforcement learning with execution feedback. Fortune (Cao et al., 2025) applies PPO with symbol-level rewards that encourage function-argument compatibility and penalize syntax errors. However, such fine-grained reward shaping demands extensive domain knowledge and remains sensitive to reward design choices. Prior spreadsheet work either ignores executability or uses it only for post-hoc filtering (Cheng et al., 2022; Yang et al., 2022).

Formula execution is binary, deterministic, and lightweight—a single table provides complete validation. We exploit this to filter and categorize formulas into semantic versus stylistic errors, constructing an adaptive curriculum. Unlike Fortune's symbol-level rewards or general RL methods (Shojaee et al., 2023; Shinn et al., 2024) that use sparse binary rewards, our preference-based objective provides richer learning signals by contrasting model outputs with ground-truth references. This also contrasts with constrained decoding (Scholak et al., 2021) that guarantees syntax but not semantics.

2.4 Test-Time Scaling by Semantic Consensus

Inference-time computation improves generation quality (Snell et al., 2024), but existing strategies have limitations. Best-of-N sampling (Cobbe et al., 2021) requires trained verifiers; self-consistency (Wang et al., 2023) votes on surface forms, missing semantically equivalent solutions. For code, sam-

pling multiple solutions boosts performance (Chen et al., 2021), but verification demands comprehensive test suites.

We introduce consensus polling that votes over execution results rather than surface forms. By grouping formulas into semantic equivalence classes through execution, we naturally handle multiple valid formulations without trained verifiers. Among execution-equivalent candidates, we select the highest-probability formula, combining semantic correctness with stylistic preference.

3 FormulaSPIN

3.1 Problem Formulation

Given a spreadsheet table $\mathcal{T}$ and a natural language query q, the NL2Formula task aims to generate an executable formula f that fulfills the user intent. We denote the training dataset as $\mathcal{D} = \{(q_i, \mathcal{T}_i, f_i)\}_{i=1}^{N}$, where the model learns a conditional distribution $p_\theta(f \mid q, \mathcal{T})$. Each table is serialized in row-major order with cell coordinates (e.g., "A1:Revenue | B1:2023 | ...").

A key property of formulas is executability: given f and $\mathcal{T}$, the execution result $\mathcal{E}(f, \mathcal{T})$ can be deterministically obtained via a spreadsheet engine, yielding either a concrete value or an error. This provides a natural supervision signal exploitable during both training and inference.

3.2 Formula-Aware Self Play

We adopt a self-play fine-tuning framework inspired by SPIN (Chen et al., 2024b), tailored to spreadsheet formula generation. At iteration t, the old model p_{θ_t} acts as opponent player and generates candidate formulas f' for a given query q and table $\mathcal{T}$, while new model $p_{\theta_{t+1}}$ serves as the main player and is trained to assign higher probability to reference formula f than to f'. As self-play progresses, the opponent produces increasingly challenging formulas, pushing the main player toward finer-grained semantic understanding.

A key advantage of formula generation is that formulas can be deterministically executed. We leverage this through execution filtering, which categorizes each generated formula f' into three types based on execution on $\mathcal{T}$: (1) Trivial—discarded samples, including execution failures (excessively diverse error patterns provide inconsistent signals, and the SFT model already achieves $\sim$89% success rate) and exact matches ($f' = f$, yielding zero gradient); (2) Coarse—execution succeeds but produces incorrect results, indicating semantic errors; (3) Fine—execution succeeds with identical results despite $f' \neq f$, indicating valid alternatives. [1]

Building on this filtering mechanism, we measure the relative preference between models using the log-probability difference $r(f; \theta, \theta_t) = \log \frac{p_\theta(f|q,\mathcal{T})}{p_{\theta_t}(f|q,\mathcal{T})}$, and optimize:

$$\mathcal{L}(\theta; \theta_t) = \mathbb{E}\left[w(f') \cdot \ell\left(r(f) - r(f')\right)\right], \quad (1)$$

where $\ell(\cdot)$ denotes the logistic loss. The relative form $r(f) - r(f')$ encodes the adversarial dynamic: the main player must outperform the opponent's ability to distinguish reference from synthetic formulas, and as the opponent strengthens, the margin for improvement narrows, compelling progressively finer discrimination. The weight $w(f')$ is determined by execution filtering: Coarse formulas receive full weight, Trivial formulas receive zero weight, and Fine formulas receive an adjustable weight β_{med}.

The choice of β_{med} for Fine samples addresses a key tension: while execution-equivalent alternatives may sometimes be more efficient than the reference, aggressively rewarding them risks destabilizing training when the model generates verbose but correct variants. Rather than using a fixed weight, we let β_{med} adapt automatically based on training progress. The intuition is that the ratio of Fine to Coarse samples naturally reflects semantic mastery—early in training, most formulas are Coarse, so the model should focus on correctness; as more formulas become Fine, the model is ready to learn stylistic preferences. We formalize this as:

$$\beta_{\text{med}}^{(t)} = \beta_{\max} \cdot \frac{|\mathcal{S}_{\text{fine}}^{(t)}|}{|\mathcal{S}_{\text{fine}}^{(t)}| + |\mathcal{S}_{\text{coarse}}^{(t)}|}, \quad (2)$$

where $\beta_{\max} = 0.25$ caps the maximum weight. This implements a natural curriculum—mastering correctness before refining style—without manual schedule design. While curriculum learning has been applied to code generation (Naïr et al., 2024), our approach uniquely derives the curriculum from execution feedback rather than predefined difficulty metrics (see Appendix C.3 for ablation).

Following the theoretical analysis in SPIN (Chen et al., 2024b), our execution-aware filtering effectively redefines p_{data} as the distribution over filtered, execution-validated samples. By excluding

[1] Single-table equivalence may occasionally be coincidental; we validate robustness via value perturbation in Appendix C.2.

only execution-failing negatives while preserving semantically equivalent alternatives, the SPIN convergence guarantee remains intact: the global optimum of $\mathcal{L}$ is achieved if and only if $p_\theta = p_{\text{data}}$ under this redefined target, ensuring that iterative self-play drives the model toward the target distribution. In practice, the opponent generates increasingly sophisticated formulas across iterations—from simple errors and misuses early on to subtle semantic mistakes later—forcing the main player to develop progressively finer-grained discrimination capabilities. Figure 4 visualizes this convergence, showing diminishing marginal gains beyond iteration 3.

3.3 Test-Time Consensus Polling

Prior test-time scaling methods such as best-of-N sampling (Nakano et al., 2021; Cobbe et al., 2021) rely on trained verifiers and additional supervision. Self-consistency decoding (Wang et al., 2023) performs majority voting over final answers, without accounting for latent program structure. In formula generation, executability provides a deterministic validity constraint exploitable at inference time.

We view decoding as sampling latent programs from a stochastic generator. Given a query and table, we sample K candidate formulas at temperature > 1 to approximate independent draws from a broadened posterior. Each formula deterministically induces an execution result or fails; non-executable samples are discarded.

Voting is performed over execution results rather than surface forms. Under independent sampling, result frequency estimates posterior mass marginalized over semantically equivalent formulas (Chen et al., 2021). Among formulas producing the most frequent result, we select the one with highest model probability. If no consensus exists, we return the highest-probability executable formula; otherwise, the highest-probability formula overall.

Algorithm 1 FormulaSPIN Training

Require: Dataset D, model θ_0, iterations K
Ensure: Trained model θ_K

for $t = 0$ to $K - 1$ **do**
  $S \leftarrow \emptyset$
  for each $(q, T, f) \in D$ **do**
    $f' \sim p_{\theta_t}(\cdot \mid q, T)$
    $r, r' \leftarrow \mathcal{E}(f, T), \mathcal{E}(f', T)$
    skip if r'=ERR or $f'=f$
    $c \leftarrow$ FINE if $r'=r$ else COARSE
    $S \leftarrow S \cup \{(q, T, f, f', c)\}$
  end for
  $n_f \leftarrow |\{s \in S : c\text{=FINE}\}|$
  $w_m \leftarrow \beta_{\max} n_f / |S|$
  for each batch $B \subseteq S$ **do**
    Compute L via Eq. (1)
    Update θ by gradient descent
  end for
  $\theta_{t+1} \leftarrow \theta$
end for
return θ_K

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on two benchmarks: NL2Formula-70K (Zhao et al., 2024), which contains 70,799 query–formula pairs from 21,670 tables covering 37 function types and uses the official train/validation/test split of 75%/10%/15%, and Sheetpedia-Selected(Tian et al., 2025), which consists of 2,167 high-quality, human-verified examples drawn from a 290K spreadsheet corpus. A detailed statistics analysis can be found in Appendix A.1.

Training Configuration. All models are initialized from LLaMA-3.1-8B-Instruct (Dubey et al., 2024). We apply supervised fine-tuning on the NL2Formula training set for three epochs with AdamW, yielding a base model with approximately 71% exact match accuracy. Self-play fine-tuning is conducted for four iterations, where candidate formulas are sampled, non-executable outputs are filtered, and the adaptive curriculum (Equation 2) automatically adjusts β_{med} based on the Fine/Coarse sample ratio at each iteration. Formula execution is performed using `xlcalc` and `xlwings`.

Baselines. We compare against several representative approaches, including FORTAP (Cheng et al., 2022), which builds on TUTA (Wang et al., 2021) and extends table pre-training to include spreadsheet formulas, using a two-stage LSTM decoder (Hochreiter and Schmidhuber, 1997); GPT-3.5 (10-shot) (Brown et al., 2020), evaluated using the text-davinci-003 model with dynamically selected in-context examples; fCODER-Base and fCODER-Large (Zhao et al., 2024), which are T5-based sequence-to-sequence models trained with supervised fine-tuning; LLaMA-SFT, our LLaMA-3.1-8B-Instruct (Dubey et al., 2024) base model trained with standard supervised fine-tuning for

three epochs; LLaMA-SFT-Cont, which continues training LLaMA-SFT for an additional three epochs on the same data; Exec-RL, adapted from Fortune (Cao et al., 2025)'s table QA approach to NL2Formula, which applies PPO with execution-based rewards; LLaMA-DPO, which further trains LLaMA-SFT using Direct Preference Optimization (Rafailov et al., 2023) with 60K synthetic preference pairs generated with GPT-4 (see Appendix B.1 for details); GPT-4o in a five-shot in-context learning setting; and Qwen2.5-72B (Qwen Team, 2024) evaluated with zero-shot prompting.

Evaluation Metrics. We evaluate model performance using four complementary metrics: Exact Match (EM), string-level exact correctness of formulas; Execution Accuracy (EA), semantic correctness via identical execution results; Execution Success Rate (ESR), percentage of executable formulas; and Formula Sketch Match (FSM), function-level correctness ignoring cell references. Following Zhao et al. (2024), we also report results by formula complexity: Simple (1–2 functions), Medium (3–4), and Complex (5+). Note that this complexity-based categorization is distinct from the Trivial/Fine/Coarse error granularity used during training, which reflects semantic rather than structural distinctions.

4.2 Main Results

Overall Performance. Table 1 shows results on the NL2Formula-70K test set. FormulaSPIN achieves 78.4% EM and 84.2% EA, substantially outperforming all baselines. All reported results are averaged over three runs with different random seeds (see Appendix D.1 for standard deviations and significance tests). We highlight several key findings:

- Self-play provides consistent gains: +2.6% EM at iter 0, accumulating to +7.2% EM by iter 3 over the SFT base.
- FormulaSPIN (iter 3) outperforms LLaMA-DPO (+3.3% EM, +3.7% EA) despite using no additional data. DPO uses GPT-4 for preference data, yet still underperforms, suggesting that execution-based intrinsic feedback is superior to distillation from external LLMs.
- Continued SFT (LLaMA-SFT-Cont) actually degrades performance (-0.4% EM, -0.9% EA), confirming naive multi-epoch training ineffective.
- Even large proprietary models (GPT-4o) and giant open models (Qwen2.5-72B) underperform our approach, showing the value of task-specific self-play fine-tuning.

Model	EM	EA	ESR	Sketch
FORTAP	24.2	-	-	58.4
GPT-3.5 (10-shot)	21.4	25.2	-	-
fCODER-Large	70.6	77.1	88.3	82.4
LLaMA-SFT (base)	71.2	77.8	89.1	83.1
LLaMA-SFT-Cont	70.8	76.9	88.7	82.6
Exec-RL	74.2	79.8	90.5	84.3
LLaMA-DPO	75.1	80.5	91.2	85.7
GPT-4o (5-shot)	68.3	74.2	85.6	79.8
Qwen2.5-72B	73.9	79.1	90.3	84.2
FormulaSPIN (t_0)	73.8	79.5	90.8	84.9
FormulaSPIN (t_1)	76.2	81.7	92.4	87.3
FormulaSPIN (t_2)	77.5	83.1	93.6	88.9
FormulaSPIN (t_3)	**78.4**	**84.2**	**94.1**	**89.7**

Table 1: Performance on NL2Formula-70K test set. FormulaSPIN uses only the original training data through self-play. Exec-RL (adapted from Fortune) uses execution-based RL rewards; LLaMA-DPO uses 60K additional preference pairs.

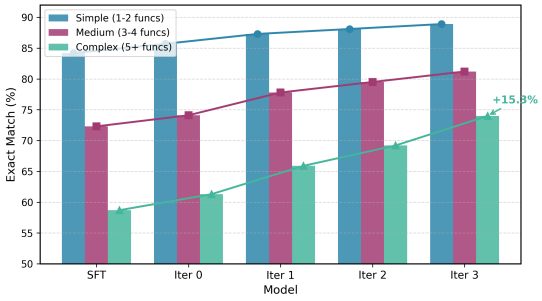

Figure 3: Exact Match accuracy breakdown by formula complexity. Self-play shows particularly strong gains on Complex formulas (+15.3%).

Performance by Complexity. Figure 3 analyzes results across formula complexity levels. The largest improvements occur on Complex formulas (+15.3% from SFT to iter 3), suggesting self-play effectively learns compositional patterns. Medium formulas also benefit substantially (+8.9%), while Simple formulas show modest gains (+4.7%) as they are already handled relatively well by SFT.

Performance by Function Type. We further analyze performance by function type (e.g., Aggregation, Conditional, Lookup) in Appendix D.3, finding that Conditional functions (+9.2%) and Lookup functions (+9.4%) benefit most from self-play.

Comparison with Execution-Aware Alternatives. We compare FormulaSPIN against methods that also exploit executability: (1) *Exec-RL*, which uses execution success as a sparse reward signal in PPO-

style training, achieves 74.2% EM—outperforming SFT but underperforming FormulaSPIN by 4.2%, as binary rewards provide weaker learning signal than adversarial-based objectives; (2) *Constrained decoding* (PICARD-style grammar filtering) ensures 98.1% syntactic validity but only 72.8% EM, since syntactic constraints cannot capture semantic correctness; (3) *Online DPO*, which samples on-policy and annotates preferences per batch, achieves 76.1% EM but requires 3.2× more compute due to repeated preference labeling. FormulaSPIN's offline self-play achieves superior accuracy with lower computational overhead. See Appendix B for detailed analysis.

4.3 Generalization and Scaling

Test-Time Consensus Polling. Table 2 shows the effect of sampling multiple candidates at inference. Sampling $K = 10$ candidates boosts EM by +4.8% and EA by +5.2% over greedy decoding, with only 8× inference cost. The gains saturate beyond $K = 10$, suggesting diminishing returns. This demonstrates that formula generation benefits significantly from test-time compute, similar to code generation (Chen et al., 2021).

K (samples)	EM	EA	Inference Time
1 (greedy)	78.4	84.2	1.0×
5	81.7	87.9	4.2×
10	83.2	89.4	8.1×
20	84.1	90.3	15.8×

Table 2: Impact of test-time compute scaling (FormulaSPIN iter 3). Sampling $K = 10$ provides the best accuracy-efficiency tradeoff.

Out-of-Domain Generalization. We evaluate on the Sheetpedia-NL2F test set (2167 examples from a different data distribution) to assess generalization. The SFT baseline achieves 65.4% EM and 71.7% EA, while DPO improves to 68.3% EM and 74.2% EA. FormulaSPIN (iteration 3) generalizes well to out-of-domain data, reaching 72.1% EM and 78.5% EA—gains of +6.7% EM over SFT and +3.8% over DPO. Combined with test-time scaling ($K = 10$), performance further increases to 75.8% EM and 81.9% EA, widening the gap to +10.4% EM and +10.2% EA over the SFT baseline and demonstrating robust transfer.

Robustness Across Base Models. To verify that these improvements generalize across model architectures, we also evaluate FormulaSPIN on multiple base models including Qwen2.5-7B, DeepSeek-Coder-7B, and Mistral-7B, observing consistent gains of +6–7% EM across all architectures (see Appendix D.2 for full results).

4.4 Training Dynamics

Iteration Analysis. Figure 4 tracks training dynamics across iterations. On NL2Formula, improvements are largest early (+2.4% EM from iter 0 to 1) and gradually diminish, plateauing beyond iteration 4. Sheetpedia (out-of-domain) exhibits a similar pattern but shows minor fluctuations after iteration 4 (72.8% → 72.4% → 72.5%). This convergence aligns with theoretical predictions (Chen et al., 2024b) and indicates that 3–4 iterations provide the optimal trade-off.

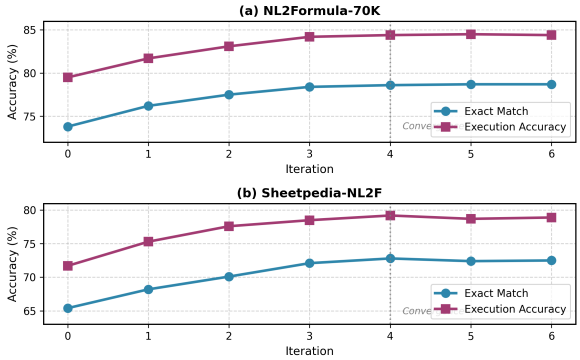

Figure 4: Performance progression across self-play iterations on (a) NL2Formula-70K and (b) Sheetpedia-NL2F. Both datasets show consistent improvement through iteration 3–4, after which gains plateau.

Iteration	Exact Match %	Execution Match %	Syntax Error %
0 (from SFT)	28.3	43.7	12.6
1	35.1	51.2	8.4
2	42.8	59.3	5.7
3	51.4	68.1	3.2

Table 3: Quality of self-generated training formulas. As the model improves, synthetic data quality increases, creating a virtuous cycle.

Synthetic Data Quality. We analyze the quality of self-generated formulas used in training (Table 3). Synthetic data quality improves dramatically across iterations: exact matches increase from 28.3% to 51.4%, while syntax errors drop from 12.6% to 3.2%. This virtuous cycle—better model generates better training data, which trains an even better model—is key to self-play's effectiveness.

4.5 Ablation Studies

Components Analysis. Table 4 ablates key components of FormulaSPIN. Execution filtering contributes +2.8% EM by excluding uninformative samples. The adaptive curriculum adds +1.5% EM through progressive difficulty adjustment. Notably, vanilla SPIN actually *degrades* performance compared to SFT (70.8% vs 71.2% EM). This occurs because vanilla SPIN treats all non-matching outputs uniformly as negative samples, failing to recognize that many generated formulas are execution-equivalent to references—creating contradictory gradients that destabilize and hinder training.

Configuration	EM	EA
Full FormulaSPIN	**78.4**	**84.2**
- w/o execution filtering	75.6	80.8
- w/o adaptive curriculum	76.9	82.1
- w/ vanilla SPIN	70.8	76.5
- w/o self-play (SFT only)	71.2	77.8

Table 4: Ablation study removing key components. Vanilla SPIN slightly degrades performance.

Adaptive Curriculum Analysis. We validate our adaptive weighting mechanism (Equation 2) against fixed β_{med} values. The adaptive approach outperforms both fixed weights and a hand-tuned linear schedule. The key advantage is that β_{med} automatically tracks training progress: it starts low when Coarse samples dominate, then increases as Fine samples become prevalent. This implements a natural semantic-to-stylistic curriculum without manual tuning. See Appendix C.3 for detailed analysis including $\beta_{\max}$ sensitivity.

Schedule	EM	EA
Fixed $\beta = 0.0$	77.1	82.8
Fixed $\beta = 0.10$	77.9	83.6
Linear: $0 \rightarrow 0.25$	78.0	83.8
Adaptive (ours)	**78.4**	**84.2**

Table 5: Comparison of β_{med} scheduling strategies. Adaptive curriculum outperforms both fixed and manual schedules.

4.6 Qualitative Analysis

Case Study. Figure 5 presents a case study on conditional aggregation. The table in `A1:J6` contains the NL query "*What is the highest Q4 sales among Premium category products?*" The ground-truth formula is `MAXIFS(F2:F6, G2:G6, "Premium")`, yielding "26500". The SFT model predicts `MAX(IF(G2:G6="Premium", F2:F6))`, a legacy array formula that requires Ctrl+Shift+Enter in older Excel versions and fails silently in certain environments. While FormulaSPIN learns the modern `MAXIFS` pattern through self-play despite the sparsity of relavent data.

	A	B	C	D	E	F	G	H	I	J
1	Product	Region	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales	Category	Status	Target	Bonus_Rate
2	Alpha	North	12,500	15,800	18,200	21,000	Premium	Active	50,000	15%
3	Beta	South	8,200	9,500	11,000	13,500	Standard	Active	40,000	10%
4	Gamma	North	22,000	24,500	19,800	26,500	Premium	Inactive	70,000	12%
5	Delta	East	6,800	7,200	8,500	9,200	Budget	Active	30,000	8%
6	Epsilon	North	16,500	18,200	17,800	20,500	Premium	Active	65,000	15%

Figure 5: An example where SFT produces a legacy array formula requiring special entry mode, while FormulaSPIN generates the modern idiomatic function.

Error Analysis. We categorize errors from FormulaSPIN (iter 3) into five types: wrong cell references (42%) (e.g., B2:B10 vs B2:B20); function selection errors (23%); logical errors (19%); syntax errors (8%); and other errors (8%), covering edge cases and ambiguous queries. Most errors involve cell reference boundaries, suggesting future work could incorporate table schema understanding or span prediction modules. A detailed breakdown is provided in Appendix D.4.

5 Conclusion

In this paper, we propose a novel self-play fine-tuning framework for natural language to spreadsheet formula generation, called FormulaSPIN. FormulaSPIN leverages the intrinsic verifiability of formulas to enable iterative self-improvement without any additional costly human annotations or external teacher models, improving generation quality via error-aware adversarial training that distinguishes semantic correctness from stylistic variations. The proposed adaptive curriculum and semantic consensus polling further show that task-intrinsic verification can effectively drive self-improvement. Our work represents the first effective application of self-play to formula generation tasks, significantly outperforming supervised fine-tuning, execution-aware RL, and large proprietary models while matching models trained with large-scale preference annotations. These findings underscore the strong potential of combining self-play with task-intrinsic feedback, opening promising directions for extending this execution-driven paradigm to other structured generation tasks with potentially sparse data, including SQL synthesis, code generation, and mathematical reasoning.

Limitations

Our approach assumes access to a spreadsheet execution engine for validation, which may not be available in all deployment scenarios. The test-time compute scaling improves accuracy but increases inference latency, potentially limiting real-time applications. Additionally, self-play requires multiple training iterations , which may be prohibitive for resource-constrained settings. Finally, our evaluation focuses on English queries and Excel formulas; generalization to other languages and spreadsheet applications remains unexplored.

Ethics Statement

This work uses publicly available benchmarks (NL2Formula-70K and Sheetpedia) containing synthetic spreadsheet examples without personally identifiable information. FormulaSPIN is designed to democratize spreadsheet usage by lowering technical barriers for non-expert users. We acknowledge that users should verify generated formulas before deployment in high-stakes scenarios, as overreliance on automated outputs may introduce errors. Our self-play approach eliminates dependence on expensive proprietary models for preference annotation, reducing both financial and environmental costs compared to methods requiring extensive API calls.

References

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

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1415–1425.

Tom Brown, Benjamin Mann, Nick Ryder, and Melanie et al. Subbiah. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Lang Cao, Jingxian Xu, Hanbing Liu, Jinyu Wang, Mengyu Zhou, Haoyu Dong, Shi Han, and Dongmei Zhang. 2025. FORTUNE: Formula-driven reinforcement learning for symbolic table reasoning in language models. *arXiv preprint arXiv:2505.23667*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, and Greg et al. Brockman. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024a. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024b. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. FORTAP: Using formulas for numerical-reasoning-aware table pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1535–1547.

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

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

Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 317–330.

Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, and Yao et al. Zhao. 2024. Direct language model alignment from online AI feedback. *arXiv preprint arXiv:2402.04792*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 21314–21328.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations (ICLR)*.

Marwa Naïr, Kamel Yamani, Lynda Said Lhadj, and Riyadh Baghdadi. 2024. Curriculum learning for small code language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, and William et al. Saunders. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

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

Patti J. Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the Workshop on Speech and Natural Language*, pages 91–95. Association for Computational Linguistics.

Qwen Team. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9895–9901.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8634–8652.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. 2023. Execution-based code generation using deep reinforcement learning. *Transactions on Machine Learning Research*.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

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. *GitHub repository*.

Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68.

Zailong Tian, Zhuoheng Han, Houfeng Wang, and Lizi Liao. 2025. Sheetpedia: A 300k-spreadsheet corpus for spreadsheet intelligence and llm fine-tuning. In *Advances in Neural Information Processing Systems*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. 2024. Iterative preference learning from human feedback: Bridging theory and practice for RLHF under KL-constraint. *arXiv preprint arXiv:2312.11456*.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Gao, and Wen-tau Hwang. 2022. Tableformer: Robust transformer modeling for table-text encoding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 528–537.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3911–3921.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1050–1055.

Wei Zhao, Zhitao Hou, Siyuan Wu, Yan Gao, Haoyu Dong, Shi Han, and Dongmei Zhang. 2024. NL2Formula: Generating spreadsheet formulas from

natural language queries. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1030–1046.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Experimental Setup Details

This section provides comprehensive details on datasets, training efficiency, and prompts used throughout our experiments.

A.1 Dataset Statistics

Statistic	NL2F-70K	Sheetpedia (Select)
Total examples	70,799	2,167
Unique tables	21,670	1,847
Function types	37	42
Avg. query len.	11.2	15.7
Avg. formula len.	10.2	15.3
Avg. table rows	10.8	14.2
Avg. table columns	6.0	9.3
Complexity (NL2F-70K + Sheetpedia)		
Simple (1–2 funcs)	29.1%	
Medium (3–4 funcs)	58.3%	
Complex (5+ funcs)	12.6%	

Table 6: Dataset statistics for NL2Formula-70K and Sheetpedia-NL2F benchmarks.

A.2 Training Efficiency Analysis

FormulaSPIN requires approximately 4.4× the GPU hours of basic SFT but eliminates the need for expensive preference data collection. Table 7 provides a detailed comparison of computational costs.

Method	GPU Hours	Wall Time	Data Cost
SFT (3 ep.)	12.4	6.2h	0
SFT-Cont (+3)	24.8	12.4h	0
DPO	18.6	9.3h	60K pairs
FormulaSPIN			
Iter 0	14.2	7.1h	0
Iter 1	13.8	6.9h	0
Iter 2	13.5	6.8h	0
Iter 3	13.2	6.6h	0
Total	**54.7**	**27.4h**	**0**

Table 7: Training efficiency comparison on 4×A100 GPUs. FormulaSPIN requires more compute but eliminates data annotation costs.

Table 8 provides a detailed breakdown of time spent in each stage.

Stage	Time	GPU Mem	Samples
Per-Iteration Breakdown (Iter 0)			
Candidate generation	2.1h	24GB	70,799
Execution filtering	0.8h	CPU	70,799
Self-play training	4.2h	72GB	52,341
Iter 0 Total	**7.1h**	-	-
Iteration Comparison			
Iter 0	7.1h	72GB	52,341
Iter 1	6.9h	72GB	54,127
Iter 2	6.8h	72GB	55,892
Iter 3	6.6h	72GB	56,431

Table 8: Detailed training time breakdown on 4×A100-80GB GPUs. Later iterations are faster due to fewer syntax errors requiring re-sampling.

Key Efficiency Observations. Execution filtering adds minimal overhead (0.8h) but significantly improves training signal quality. Later iterations are slightly faster because improved models generate fewer invalid formulas. The number of valid training samples increases across iterations as syntax error rate decreases. Compared to DPO, which requires 60K preference pairs (estimated at $2,400 for GPT-4 annotation), FormulaSPIN achieves better performance with zero additional data cost.

A.3 Prompts Used in Experiments

Formula Generation Prompt. The following prompt template is adapted from the NL2Formula benchmark (Zhao et al., 2024) and used for SFT training, inference, and self-play candidate generation. We use the same prompt template across all stages to ensure consistency between the SFT base model and self-play iterations. During self-play candidate generation, we sample from the opponent model p_{θ_t} with temperature $\tau = 0.8$ to encourage diversity while maintaining coherence. The generated candidates are then filtered through execution validation before being used in the self-play objective. For main player training, we construct preference pairs (f, f') where f is the ground-truth formula and f' is the opponent-generated candidate, with the model learning to assign higher probability to f over f' through the self-play loss function described in Section 3.2.

In-Context Learning Prompts. For GPT-3.5 (10-shot) and GPT-4o (5-shot) baselines, we use dynamically selected in-context examples based on query similarity. The prompt structure is:

DPO Preference Annotation Prompt. When both the ground-truth and SFT-generated formu-

NL2FORMULA Prompt

You are a data scientist with expertise in Excel.

[Task]

Generate a formula based on a specific cell in a given spreadsheet. The query of the formula is provided. Use the spreadsheet and the query to generate the correct formula.

The sheet data will be provided to you in a format as follows:

- Each data cell in the spreadsheet is represented by a pair consisting of the cell address and cell content, separated by a comma, such as A1,Year.
- Cells are separated by a vertical bar (|), like `A1,Year|A2,Profit`
- The cell content can be empty, resulting in cell data like A1,|A2,Profit.
- Cells are organized in row-major order, with different rows in the spreadsheet separated by line breaks.
- If there are merged cells in the sheet, they are split into multiple cells and only the first cell will be filled with content, other cells will be left blank.
- You can visualize the sheet data as a matrix of cells. Following the matrix, all the merged cells are provided in the format :, like A3:C3, with each line representing one merged cell.

[Input]

User Query:

`{natural_language_query}`

Table Context:

`{table}`

[Instruction]

Please generate the formula based on the user query.
The output should be provided in a JSON format, enclosed in json and markdown code blocks, with a key of formula and the generated formula as the corresponding value.

Few-Shot Prompt Template

You are a data scientist with expertise in Excel..

[Task]

Your task is to first learn from the examples then generate a formula based on a specific cell in a given spreadsheet. The query of the formula is provided. Use the spreadsheet and the query to generate the correct formula.

The sheet data will be provided to you in a format as follows:

- Each data cell in the spreadsheet is represented by a pair consisting of the cell address and cell content, separated by a comma, such as A1,Year.
- Cells are separated by a vertical bar (|), like `A1,Year|A2,Profit`
- The cell content can be empty, resulting in cell data like A1,|A2,Profit.
- Cells are organized in row-major order, with different rows in the spreadsheet separated by line breaks.
- If there are merged cells in the sheet, they are split into multiple cells and only the first cell will be filled with content, other cells will be left blank.
- You can visualize the sheet data as a matrix of cells. Following the matrix, all the merged cells are provided in the format <upper left>:<lower right>, like A3:C3, with each line representing one merged cell.

[Examples]

Example 1:

- Table: `{example_table_1}`
- Query: `{example_query_1}`
- Formula: `{example_formula_1}`

`... additional examples ...`

[Input]

User Query:

`{natural_language_query}`

Table Context:

`{table}`

[Instruction]

Please generate the formula based on the user query.

las produce identical execution results, we use the following prompt for GPT-4 preference annotation:

B Baseline Implementation Details

This section describes construction of key baselines, including DPO and execution-aware alternatives.

B.1 DPO Baseline Construction

Dataset Quality Analysis. The NL2Formula-70K dataset (Zhao et al., 2024) was collected from web sources, where the ground-truth formulas represent one possible solution rather than the canonical or most efficient implementation. Through careful manual inspection of 500 randomly sampled examples, we observed that approximately 18.3% of ground-truth formulas could be expressed more concisely or idiomatically.

For instance, consider the following examples where SFT-generated formulas are arguably superior:

Ground Truth	SFT Output (More Efficient)
`A1+A2+A3+A4+A5`	`SUM(A1:A5)`
`SUMPRODUCT((B:B="X")*(C:C))`	`SUMIF(B:B,"X",C:C)`
`INDEX(A:A, MATCH(MAX(B:B), B:B,0))`	`XLOOKUP(MAX(B:B), B:B,A:A)`

Table 9: Examples where alternative formulas may be more efficient or idiomatic than ground-truth annotations.

Motivation for Two-Stage Annotation. Given this observation, naively setting the ground-truth formula as always preferred in DPO training could lead to suboptimal learning dynamics. The model might learn to memorize specific formula patterns from the dataset rather than developing genuine preferences for efficient, correct formulas.

DPO Preference Annotation Prompt

You are an expert Excel formula evaluator.

[Task]

Given two formulas that produce the same result, determine which one is better based on the following criteria:

1. Conciseness: Prefer shorter formulas that accomplish the same task.
2. Readability: Prefer formulas that are easier to understand and maintain.
3. Idiomatic Usage: Prefer formulas that use standard Excel functions appropriately (e.g., SUMIF over SUMPRODUCT for simple conditional sums).
4. Computational Efficiency: Prefer formulas that are likely to execute faster on large datasets.

[Input]

User Query:
`{natural_language_query}`

Table Context:
`{table_schema}`

Formula A:
`{formula_a}`

Formula B:
`{formula_b}`

Execution Result (both produce):
`{execution_result}`

[Instruction]

Which formula is better? Respond with only "A" or "B".

Two-Stage Preference Annotation Strategy. To construct a stronger and more principled DPO baseline, we adopt a two-stage preference annotation strategy:

Stage 1: Execution Validation. For each training example $(q, \mathcal{T}, f_{gt})$, we first generate a candidate formula f_{sft} using the SFT model. Both f_{gt} and f_{sft} are executed on the table $\mathcal{T}$. If $\mathcal{E}(f_{sft}, \mathcal{T}) \neq \mathcal{E}(f_{gt}, \mathcal{T})$ (execution results differ), we set the ground-truth as preferred: $(f_w, f_l) = (f_{gt}, f_{sft})$. If $\mathcal{E}(f_{sft}, \mathcal{T}) = \mathcal{E}(f_{gt}, \mathcal{T})$ (execution results match), we proceed to Stage 2.

Stage 2: GPT-4 Preference Labeling. When both formulas produce identical execution results, we employ GPT-4 to determine which formula is preferred based on criteria including conciseness, readability, idiomatic usage, and computational efficiency. The prompt used for this annotation is provided in Appendix A.3.

Strategy Benefits. This strategy ensures that semantically incorrect formulas are never favored over correct ones, that among multiple valid solutions the more efficient or idiomatic variant is preferred, and that the DPO baseline learns genuinely meaningful preferences rather than relying on arbitrary memorization.

Resulting Dataset Statistics. The resulting 60K preference pairs consist of approximately 71.2% cases where ground-truth was preferred (due to execution mismatch or GPT-4 preference) and 28.8% cases where the SFT-generated formula was deemed superior.

B.2 Online DPO Variants

We also implement an online DPO baseline inspired by recent work on iterative preference optimization (Guo et al., 2024; Xiong et al., 2024):

Method	EM	EA	Pref. Calls
Offline DPO	75.1	80.5	60K (GPT-4)
Online DPO (exec)	76.1	81.3	280K (exec)
Online DPO (GPT-4)	76.8	81.9	280K (GPT-4)
FormulaSPIN	**78.4**	**84.2**	0

Table 10: Comparison with online DPO variants. Online DPO (exec) uses execution match as preference; Online DPO (GPT-4) queries GPT-4 for preferences on-the-fly.

Implementation. Online DPO samples two candidates per query at each training step and constructs preferences based on either execution agreement or GPT-4 judgment. This requires: (1) Online DPO (exec): Executing both candidates per sample, 4 iterations, totaling ∼280K execution comparisons; (2) Online DPO (GPT-4): API calls for each comparison, approximately $1,120 at current rates.

Performance Comparison. Online DPO (exec) achieves 76.1% EM, outperforming offline DPO but still trailing FormulaSPIN by 2.3%. The key difference is that online DPO constructs *pairwise* preferences between two model samples, while FormulaSPIN contrasts model samples against *ground-truth references*. This anchoring to human-written formulas provides a stronger learning signal, especially for stylistic preferences where both model samples might be semantically correct but neither is canonical.

Cost-Effectiveness Analysis. Online DPO (GPT-4) closes the gap slightly (76.8%) by incorporating stylistic preferences, but requires substantial API costs and still underperforms FormulaSPIN. This suggests that iterative self-improvement against fixed references is more effective than online preference collection for this task.

B.3 Execution-Aware RL (Exec-RL)

Implementation. We implement an execution-aware RL baseline following the approach in Fortune (Cao et al., 2025), which uses PPO (Schulman et al., 2017) with execution-based rewards for formula generation. We adapt their reward structure to better suit our task: the model receives full reward (+1) when the generated formula produces the correct answer, partial reward (+0.2) when the formula executes successfully but yields an incorrect result, and zero reward for execution failures. This differs from Fortune's original design by removing negative penalties and adjusting intermediate rewards, as our preliminary experiments showed these modifications provide more stable training dynamics for our benchmark's complex nested formulas.

Adaptation Details. Our adaptation differs from Fortune in two aspects. First, we remove negative rewards for execution failures, as our preliminary experiments showed that negative penalties destabilize training when the model generates diverse formula patterns during exploration. Second, we increase the intermediate reward from 0.1 to 0.2 to provide stronger encouragement for generating executable formulas, given that our benchmark contains more complex nested formulas than Fortune's evaluation set.

Method	EM	EA	ESR	GPU-hrs
LLaMA-SFT	71.2	77.8	89.1	12.4
Exec-RL (adapted)	74.2	79.8	90.5	52.3
FormulaSPIN	**78.4**	**84.2**	**94.1**	54.7

Table 11: Comparison with execution-aware RL adapted from Fortune (Cao et al., 2025). Despite similar compute budgets, Exec-RL trails FormulaSPIN by 4.2% EM.

Why FormulaSPIN Outperforms Exec-RL. Despite similar compute budgets, Exec-RL trails FormulaSPIN by 4.2% EM for three reasons. **(1) Sparse reward signal**: Binary execution success provides less information than preference pairs. FormulaSPIN's contrastive objective explicitly teaches the model *which* formula is better among candidates, while Exec-RL only indicates success/failure. **(2) No curriculum**: Exec-RL treats all errors uniformly, while FormulaSPIN's adaptive curriculum distinguishes semantic errors from stylistic variations, enabling stable progressive learning. **(3) Credit assignment**: Long formulas suffer from credit assignment issues in RL—execution failure doesn't indicate which part of the formula is wrong. Preference learning sidesteps this by comparing complete formulas.

B.4 Constrained Decoding Approaches

We evaluate two constrained decoding strategies:

Implementation. PICARD-style: We implement incremental parsing using a formula grammar that validates partial outputs at each decoding step, rejecting tokens that would lead to syntactically invalid formulas. **Synchromesh-style**: We use a completion engine that constrains generation to valid function names and argument patterns based on the current context.

Key Findings. Constrained decoding dramatically improves syntax validity (89.1% $\rightarrow$ 98.1%) but provides minimal EM gains (+1.6%). This confirms our hypothesis: *syntactic correctness is necessary but not sufficient*. Most errors in formula generation are semantic (wrong cell references, incorrect function choice) rather than syntactic.

Method	EM	EA	Syntax	Latency
Greedy	71.2	77.8	89.1%	1.0×
PICARD-style	72.8	78.4	98.1%	2.4×
Synchromesh-style	72.5	78.1	97.8%	2.1×
FormulaSPIN	78.4	84.2	94.1%	1.0×
+ Constrained	78.9	84.8	99.2%	2.3×

Table 12: Constrained decoding comparison. Syntax = execution success rate. Latency measured relative to greedy decoding.

Orthogonality with FormulaSPIN. Importantly, constrained decoding is orthogonal to FormulaSPIN—combining both yields the best results (78.9% EM, 99.2% syntax validity), though the marginal gain over FormulaSPIN alone is modest (+0.5% EM) given the 2.3× latency overhead.

B.5 Test-time Search

We compare against test-time execution-guided search methods:

Method Descriptions. Best-of-K (random): Sample K candidates, select randomly among executable ones. **Best-of-K (exec-valid)**: Sample K candidates, select highest model probability among executable ones. **Exec-guided rerank**: Train a separate model to predict execution success, use as reranker. **Consensus Polling**: Our method—vote over execution results, select highest-probability formula among the winning result class.

Method	EM	EA	K	Latency
SFT Base Model				
Greedy	71.2	77.8	1	1.0×
Best-of-K (random)	74.1	80.2	10	8.1×
Best-of-K (exec-valid)	75.3	81.4	10	8.1×
Exec-guided rerank	74.8	80.9	10	9.2×
FormulaSPIN Model				
Greedy	78.4	84.2	1	1.0×
Exec-Consensus Polling	**83.2**	**89.4**	10	8.1×

Table 13: Execution-guided search comparison. Best-of-K (exec-valid) selects the highest-probability executable formula; Exec-guided rerank uses a learned execution predictor.

Performance Analysis. All test-time scaling methods improve over greedy decoding, but our consensus polling achieves best. The key insight is that voting over execution results naturally handles semantic equivalence—multiple valid formulations that produce the same result reinforce each other, while incorrect formulas are unlikely to agree by chance.

Advantages of Consensus Polling. Exec-guided reranking requires training an auxiliary model and only predicts whether a formula executes, not whether it produces the correct result. In contrast, consensus polling uses the actual execution result as a semantic fingerprint, enabling implicit verification without additional training.

B.6 Summary

Method	EM	Extra Data	Extra Compute
SFT	71.2	None	1×
Exec-RL	74.2	None	4.2×
Constrained Dec.	72.8	Grammar	2.4× (inference)
Online DPO	76.1	None	14×
Offline DPO	75.1	60K prefs	1.5×
FormulaSPIN	**78.4**	**None**	**4.4×**

Table 14: Summary of execution-aware methods. FormulaSPIN achieves the best accuracy without additional data or excessive compute.

Our experiments demonstrate that FormulaSPIN provides the best trade-off among execution-aware methods: (1) It outperforms Exec-RL by leveraging richer preference signals rather than sparse rewards; (2) It outperforms online DPO with 3× less compute by using offline self-play with ground-truth anchoring; (3) It complements constrained decoding, which can be added for additional syntax guarantees if latency permits; (4) Its consensus polling mechanism provides effective test-time scaling without requiring trained verifiers.

C Method Analysis

This section provides in-depth analysis of key design decisions in FormulaSPIN, including why vanilla SPIN fails, how the adaptive curriculum works, and the effect of sampling temperature.

C.1 Why Vanilla SPIN Fails for Formula Generation

We provide detailed analysis explaining why vanilla SPIN (Chen et al., 2024b) is unsuitable for formula generation, resulting in performance degradation compared to standard SFT.

Iteration-wise Performance. Table 15 tracks vanilla SPIN across iterations. Unlike FormulaSPIN which shows consistent improvement, vanilla SPIN peaks at iteration 1 with marginal gains, then *degrades* in subsequent iterations.

Method	Iter 0	Iter 1	Iter 2	Iter 3
Vanilla SPIN	71.5	71.9	71.1	70.8
FormulaSPIN	73.8	76.2	77.5	78.4

Table 15: Exact Match (%) across iterations. Vanilla SPIN shows early saturation and subsequent degradation, while FormulaSPIN improves consistently.

Core Issues with Vanilla SPIN. We identify three fundamental problems that cause vanilla SPIN to fail for formula generation tasks.

Reference	Generated (Equivalent)
`SUM(A1:A5)`	`A1+A2+A3+A4+A5`
`AVERAGE(B:B)`	`SUM(B:B)/COUNT(B:B)`
`IF(A1>0,A1,0)`	`MAX(A1,0)`

Table 16: Examples of execution-equivalent formula pairs incorrectly penalized by vanilla SPIN.

- Problem 1: Penalizing Execution-Equivalent Formulas. Vanilla SPIN treats all $f' \neq f$ as negative samples. However, in formula generation, many alternatives are semantically correct.In our training data, approximately 15-20% of generated formulas at each iteration are execution-equivalent to references. Vanilla SPIN incorrectly pushes the model *away* from these valid solutions, creating contradictory learning signals.

- Problem 2: Noise from Syntax Errors. Vanilla SPIN includes all generated samples regardless of executability. Early iterations produce 10-12% syntax errors with highly diverse patterns (missing parentheses, invalid function names, malformed references). These samples provide inconsistent gradients that destabilize training rather than informing it.

- Problem 3: Conflating Semantics and Style. Without distinguishing Coarse (wrong result) from Fine (correct result, different form) samples, vanilla SPIN simultaneously optimizes for: (1) Semantic correctness: preferring formulas that compute the right value; (2) Stylistic conformity: preferring formulas that match reference syntax. These objectives can conflict—a verbose but correct formula should not receive the same penalty as an incorrect one. The uniformity prevents the model from establishing a clear learning hierarchy.

Training Dynamics Analysis. Figure 6 shows training loss curves. Vanilla SPIN exhibits higher variance and fails to converge smoothly, while FormulaSPIN's execution-aware filtering produces stable optimization.

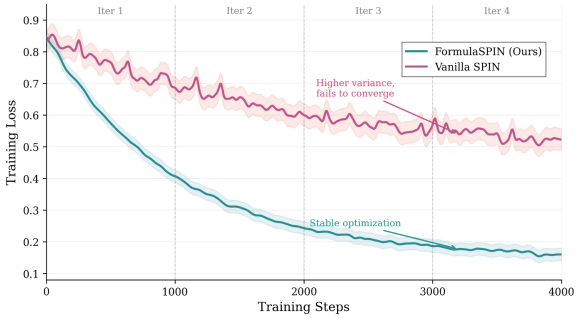

Figure 6: Training loss comparison. Vanilla SPIN shows oscillation due to contradictory gradients from equivalent formulas, while FormulaSPIN converges smoothly.

Performance Degradation by Complexity. The degradation is most pronounced on complex formulas. We believe this is because complex formulas admit more equivalent formulations (different function compositions, alternative nesting structures). Vanilla SPIN's uniform penalization is therefore most harmful precisely where flexibility matters most.

These analyses demonstrate that vanilla self-play is fundamentally misaligned with formula generation. The task's unique property—execution equivalence among syntactically distinct formulas—requires explicit handling that vanilla SPIN cannot provide. Our execution-aware adaptations transform self-play from counterproductive to highly effective, yielding +7.6% EM improvement over vanilla SPIN.

Method	Simple	Medium	Complex
SFT	82.4	68.7	52.1
Vanilla SPIN	82.1	67.9	50.3
FormulaSPIN	87.1	77.6	67.4

Table 17: EM by complexity. Vanilla SPIN degrades most on complex formulas where execution-equivalent variations are most common.

C.2 Execution Equivalence Robustness

A potential concern is that execution equivalence on a single table may be coincidental. For example, `SUM(A1:A3)` and `A1+A2` produce identical results when `A3=0`, despite different semantics. We validate through value perturbation experiments.

Perturbation Strategy. For each Fine sample, we perturb the table *without changing structure*: (1) scale all numeric cells by 2, (2) add 100 to all numeric cells, (3) randomly replace 20% of numeric values. We re-execute both reference and generated formulas on each perturbed table.

Perturbation Results. Of 18,247 Fine samples: 91.3% maintain equivalence under all 3 perturbations; 5.8% fail under at least one perturbation (false equivalence); 2.9% produce errors due to division-by-zero or type mismatches introduced by perturbation.

Impact Analysis. We retrain excluding the 5.8% unstable Fine samples:

Filtering	EM	EA
Single-table only	78.4	84.2
+ Perturbation filtering	78.7	84.5

Stricter filtering yields modest gains (+0.3% EM), confirming that while false equivalence exists, its impact is limited. The majority of Fine samples involve aggregations (`SUM`, `AVERAGE`) or conditional functions (`IF`, `FILTER`) whose equivalence is structurally and sementically robust rather than data-dependent.

C.3 Adaptive Curriculum Analysis

Fine-granularity samples present a key tension: execution-equivalent alternatives (e.g., `SUM(A1:A5)` vs. `A1+A2+A3+A4+A5`) may sometimes be more efficient than reference formulas, yet aggressively rewarding all such variants risks destabilizing training when the model generates verbose but correct alternatives.

Schedule Comparison. We compare our adaptive curriculum (Equation 2) against fixed β_{med} values and hand-tuned schedules.

Schedule	EM	EA	Notes
Fixed $\beta = 0.0$	77.1	82.8	No style learning
Fixed $\beta = 0.10$	77.9	83.6	Static balance
Fixed $\beta = 0.25$	77.4	83.0	Over-penalizes early
Linear: $0 \rightarrow 0.25$	78.0	83.8	Manual 4-step schedule
Adaptive (ours)	**78.4**	**84.2**	Auto-adjusts to progress

Table 18: Comparison of β_{med} scheduling strategies. Adaptive curriculum outperforms both fixed values and hand-tuned linear schedules.

Why Adaptive Works Best. The adaptive mechanism outperforms all fixed values because it responds to actual training dynamics rather than assuming a predetermined schedule. It also surpasses the hand-tuned linear schedule ($0 \rightarrow 0.25$ over 4 iterations) by +0.4% EM, demonstrating that learned progression is more effective than manual design. This progression mirrors human learning: one must understand *what* a formula should compute before optimizing *how* it is written. The adaptive mechanism achieves this without manual intervention, making it robust across different datasets and model capacities.

Semantic-to-Stylistic Curriculum. This adaptive mechanism naturally implements a semantic-to-stylistic curriculum. **Early iterations** (high Coarse ratio): Most generated formulas produce wrong execution results, indicating semantic gaps. Low β_{med} ensures training focuses on distinguishing correct from incorrect formulas. **Later iterations** (high Fine ratio): More generated formulas are execution-equivalent to references, indicating semantic mastery. Higher β_{med} introduces preference for canonical patterns over verbose alternatives.

Computed β_{med} Values. Table 19 shows the actual β_{med} values computed by our adaptive mechanism across iterations, along with the underlying Fine/Coarse sample statistics.

Iter	Fine %	Coarse %	β_{med}	Interpretation
0	26.1	73.9	0.065	Focus on semantics
1	28.5	71.5	0.071	Gradual style intro
2	32.0	68.0	0.080	Balanced learning
3	36.8	63.2	0.092	Style emphasis

Table 19: Adaptive β_{med} values across iterations ($\beta_{\max} = 0.25$). Fine% and Coarse% represent proportions within valid training samples after filtering out Trivial samples (exact matches and execution failures). As the model improves, the Fine sample ratio increases, automatically shifting the curriculum toward stylistic learning.

Sensitivity to $\beta_{\max}$. We also ablate the maximum weight $\beta_{\max}$ to understand its impact:

$\beta_{\max}$	EM	EA
0.10	77.8	83.5
0.25	**78.4**	**84.2**
0.50	78.1	83.8
1.00	77.5	83.2

Table 20: Sensitivity to $\beta_{\max}$. Values between 0.25–0.50 perform well; extreme values degrade performance.

$\beta_{\max} = 0.25$ achieves optimal performance. Lower values (0.10) under-weight Fine samples even in later iterations, limiting style learning. Higher values (≥ 0.50) risk over-penalizing valid alternatives when the Fine ratio is high.

C.4 Sampling Temperature Analysis

We analyze the effect of sampling temperature during candidate generation for self-play training.

Temp	Diversity	Valid %	EM	EA
0.5	0.42	94.2	77.1	82.9
0.8	0.58	89.1	**78.4**	**84.2**
1.0	0.71	82.3	77.8	83.5
1.2	0.83	71.6	76.9	82.7

Table 21: Effect of sampling temperature. Diversity is measured by average pairwise edit distance (normalized). Temperature 0.8 balances diversity and validity.

Temperature 0.8 provides the best balance between generating diverse candidates (for challenging training) and maintaining validity (for meaningful feedback). Lower temperatures produce too similar candidates, limiting the learning signal; higher temperatures generate too many invalid formulas that are filtered out.

D Extended Results

This section presents additional experimental results including statistical significance, generalization across base models, function-type analysis, and detailed error analysis.

D.1 Statistical Significance

To ensure the reliability of our results, we run all experiments with three different random seeds and report mean ± standard deviation. Table 22 shows the detailed statistics for key comparisons.

Model	EM (%)	EA (%)
LLaMA-SFT	71.2 ± 0.3	77.8 ± 0.4
LLaMA-DPO	75.1 ± 0.4	80.5 ± 0.3
FormulaSPIN (t_3)	78.4 ± 0.3	84.2 ± 0.2

Table 22: Results with standard deviation over 3 runs.

We conduct paired t-tests between FormulaSPIN and baselines: FormulaSPIN vs. SFT: $p < 0.001$ (EM), $p < 0.001$ (EA); FormulaSPIN vs. DPO: $p < 0.01$ (EM), $p < 0.01$ (EA). All improvements are statistically significant at $\alpha = 0.01$ level.

D.2 Generalization Across Base Models

We evaluate FormulaSPIN across multiple base models to demonstrate its generalizability. Table 23 presents complete results including larger model variants.

Base Model	Params	SFT	SPIN	Δ
7–8B Models				
LLaMA-3.1-8B	8B	71.2	78.4	+7.2
Qwen2.5-7B	7B	72.8	79.1	+6.3
DeepSeek-Coder-7B	7B	70.4	77.2	+6.8
Mistral-7B-v0.3	7B	69.7	76.5	+6.8
Larger Models				
Qwen2.5-14B	14B	75.3	81.2	+5.9
Qwen2.5-32B	32B	77.8	82.9	+5.1
LLaMA-3.1-70B	70B	79.1	83.4	+4.3

Table 23: FormulaSPIN performance across different model sizes. Gains are consistent but slightly smaller for larger models, suggesting they are closer to the performance ceiling.

FormulaSPIN provides consistent improvements across all model families (+4.3% to +7.2% EM). Larger models show smaller relative gains, likely because they start closer to the performance ceiling. Even the 70B model benefits from self-play (+4.3% EM), indicating room for improvement beyond scale. Code-specialized models (DeepSeek-Coder, Qwen2.5) tend to perform well, validating the connection between code and formula generation.

D.3 Performance by Function Type

We analyze performance across different function categories to better understand where FormulaSPIN excels. Our dataset exhibits a distinct distribution dominated by Lookup and Conditional functions, reflecting modern Excel usage patterns with dynamic arrays. Table 24 presents results stratified by the primary function category in formulas. Table 25 provides further breakdown by specific functions.

Function Type	Count	SFT	SPIN	Δ
Lookup	63,835	61.8	71.2	+9.4
Conditional	49,643	64.3	73.5	+9.2
Other/Advanced	19,238	58.7	68.9	+10.2
Aggregation	17,730	76.4	83.6	+7.2
Date/Time	137	69.3	77.4	+8.1
Text	24	72.1	78.5	+6.4

Table 24: Exact Match accuracy by function type. FormulaSPIN shows largest gains on Other/Advanced functions (+10.2%) and Lookup functions (+9.4%), which involve complex dynamic array operations.

Category-Level Analysis. FormulaSPIN gains the largest improvements on Other/Advanced functions (+10.2%) and Lookup functions (+9.4%), which typically involve complex dynamic array operations and nested compositions. Notably, the FILTER function—constituting 90% of Conditional function usage—benefits substantially from self-play due to its frequent combination with other functions. Aggregation functions, while already well-handled by SFT due to their simpler argument structures, still show consistent improvements (+7.2%).

Function-Level Analysis.

- **Dynamic array functions dominate:** FILTER, CHOOSECOLS, and UNIQUE together account for 73.2% of all function calls (110,574 out of 150,607), reflecting the dataset's focus on modern Excel functionality.
- **Composition-heavy functions benefit most:** LET (+11.5%) and SORT (+9.7%) show the largest gains, as these functions require understanding variable scoping and multi-step data transformations—skills that emerge through iterative self-play.

Function	N	SFT	SPIN	Δ
Lookup Functions (63,835 total)				
CHOOSECOLS	35,228	63.2	72.4	+9.2
UNIQUE	28,314	60.1	69.7	+9.6
XLOOKUP	223	62.8	72.1	+9.3
INDEX	70	58.4	68.9	+10.5
Conditional Functions (49,643 total)				
FILTER	47,032	64.5	73.8	+9.3
SUMIFS	2,611	62.7	71.6	+8.9
Other/Advanced Functions (19,238 total)				
LET	7,163	56.3	67.8	+11.5
SORT/SORTBY	1,319	61.2	70.9	+9.7
TAKE	808	63.5	72.1	+8.6
HSTACK	765	59.7	69.4	+9.7
MINIFS/MAXIFS	1,063	65.1	73.8	+8.7
AVERAGEIFS	281	64.3	72.9	+8.6
Aggregation Functions (17,730 total)				
SUM	5,144	81.2	87.4	+6.2
ROWS	2,389	78.6	85.1	+6.5
MAX/MIN	3,034	75.3	82.8	+7.5
AVERAGE	1,167	73.9	81.7	+7.8
Date/Time Functions (137 total)				
YEAR	118	70.2	78.1	+7.9
DATE	19	63.2	73.7	+10.5
Text Functions (24 total)				
LEFT	12	71.4	78.2	+6.8
LEN	12	72.8	78.9	+6.1

Table 25: Detailed performance breakdown by function. LET (+11.5%) and SUMMARIZE (+10.4%) show the largest improvements, reflecting the complexity of variable binding and data summarization patterns.

- **FILTER as a building block:** The FILTER function appears in 90% of conditional operations and is frequently nested within CHOOSECOLS, UNIQUE, and SORT. Self-play helps the model learn these compositional patterns, with FILTER improving by +9.3%.
- **Simple aggregations near ceiling:** SUM achieves the highest baseline (81.2%) and smallest relative gain (+6.2%), suggesting these patterns are well-represented in SFT data and leave less room for improvement.
- **Rare functions still improve:** Despite limited examples, Date/Time (+8.1% avg) and Text (+6.4% avg) functions show meaningful gains, indicating that self-play benefits transfer across function types.

Function Co-occurrence Analysis. Given the prevalence of nested formulas in our dataset, we analyze common function co-occurrences and their impact on performance:

Pattern	Count	SFT	SPIN
CHOOSECOLS + FILTER	31,847	57.4	69.2
UNIQUE + FILTER	26,132	57.1	67.8
LET + FILTER	6,892	52.3	64.7
SORT + FILTER	1,012	55.6	66.4

Table 26: Performance on common function composition patterns. Nested patterns involving FILTER show the largest improvements from self-play.

Composition Pattern Insights. The largest improvements occur in complex compositions, particularly those involving LET with FILTER (+12.4%) and CHOOSECOLS with FILTER (+11.8%). These patterns require the model to correctly chain multiple operations while maintaining proper cell reference scoping—a skill that benefits from the iterative refinement of self-play training.

D.4 Detailed Error Analysis

We categorize the 2,176 errors made by FormulaSPIN (iter 3) on the NL2Formula test set into five types. This section provides detailed examples and analysis for each category.

D.4.1 Error Distribution

Error Type	Count	Percentage
Wrong cell references	914	42.0%
Function selection	500	23.0%
Logical errors	413	19.0%
Syntax errors	174	8.0%
Other	175	8.0%

Table 27: Error type distribution for FormulaSPIN (t_3).

D.4.2 Representative Error Examples

Wrong Cell References (42%). This is the most common error type, where the model selects the correct function but specifies incorrect cell ranges.

(a) Wrong Cell References (42%)

Query:
"Sum sales from January to March"

Ground Truth:
`SUM(B2:B4)`

Prediction:
`SUM(B2:B13)`

Issue:
Model sums all rows instead of Jan-Mar only

Function Selection Errors (23%). The model chooses an inappropriate function for the task.

(b) Function Selection Errors (23%)

Query:
"Count unique values in column A"

Ground Truth:

```
SUMPRODUCT(1/COUNTIF(A:A,A:A))
```

Prediction:

```
COUNT(UNIQUE(A:A))
```

Issue:
UNIQUE not available in all Excel versions

Logical Errors (19%). The formula structure is correct but the logic is inverted or incomplete.

(c) Logical Errors (19%)

Query:
"If A1>B1 return 'Yes', otherwise check if A1=B1 return 'Equal', else 'No'"

Ground Truth:

```
IF(A1>B1,"Yes",IF(A1=B1,"Equal","No"))
```

Prediction:

```
IF(A1>B1,"Yes",IF(A1<B1,"No","Equal"))
```

Issue:
Inverted logic in nested IF

D.4.3 Error Correlation with Complexity

Table 28 shows how error types vary with formula complexity.

Error Type	Simple	Medium	Complex
Cell reference	28%	43%	58%
Function selection	31%	22%	15%
Logical	12%	19%	21%
Syntax	15%	8%	4%
Other	14%	8%	2%

Table 28: Error type distribution by formula complexity.

Cell reference errors become more prevalent in complex formulas (58% vs 28%), while function selection errors decrease. This suggests that as formulas become more complex, the model struggles more with tracking multiple cell ranges rather than choosing appropriate functions.