

Compositional Tool-Sequence Generation in Small Language Models

Stanford CS224N {Custom}, Mentor: Advit Deepak. 1 Late Day pooled from Maanit and Benji.

Maanit Goel

Department of Computer Science
Stanford University
maanit@stanford.edu

Benji Warburton

Department of Electrical Engineering
Stanford University
benjiw@stanford.edu

Abstract

We study whether supervised fine-tuning (SFT) enables small language models (SLMs) to perform compositional tool-sequence generation rather than relying on local action-pattern memorization. We define the task as instruction-to-program generation in a controlled action domain-specific language (DSL) and evaluate Qwen3 models on a synthetic dataset with evaluation splits spanning IID and OOD regimes, including length generalization and a held-out adjacent-pair split that removes specific tool transitions from training targets. We thus isolate whether models can compose unseen local tool transitions while operating over a fixed syntax and familiar tool vocabulary. SFT substantially improves end-to-end program generation over prompt-only baselines on exact match, ignore-magnitude exact match, step-level F1, length control, and tool edit distance. However, exact match on the held-out adjacent-pair split remains very low even after SFT. Our results suggest that IID SFT clearly improves semantic sequence generation and continuation, but does not solve compositional generalization of unseen tool transitions.

1 Introduction

As language models are increasingly deployed in agentic settings, reliable tool calling has become a central practical concern. In these settings, a model must do more than identify a single tool: it must map natural-language intent into a correct *sequence* of structured tool calls, often under strict syntax constraints. This is especially important for small language models (SLMs), whose lower inference cost makes them attractive for real-world agentic use, but whose ability to generalize multi-step tool composition remains unclear.

A strong tool-calling system must solve at least two distinct problems. First, it must produce outputs that satisfy a required formal interface. Second, it must make the correct semantic decisions: selecting the right tools, ordering them correctly, and assigning appropriate arguments at each step. Standard benchmark performance does not cleanly separate these capabilities, and strong results may still reflect memorization of common local action patterns rather than systematic multi-step generalization.

This project studies that distinction in a controlled instruction-to-program setting. We define a small action domain-specific language (DSL) in which models translate natural-language instructions into bracketed programs such as `[forward: 60] [right: 15]`. We then evaluate pretrained and fine-tuned Qwen3 models under three conditions: in-distribution performance, length generalization to longer programs, and a held-out adjacent-pair setting in which specific action bigrams are removed from training targets. The held-out adjacent-pair split is our main test of compositionality, since it asks whether a model can compose unseen local transitions while still operating over familiar tools and syntax.

Our goal is therefore not just to measure whether supervised fine-tuning improves benchmark accuracy, but to test *what kind* of improvement it produces. By combining constrained decoding with

controlled split design, we isolate semantic sequence prediction from surface-format validity and evaluate whether fine-tuning improves genuine compositional tool-sequence generation.

2 Related Work

Our work bridges prior research in tool use in language models, instruction tuning for small models, and controlled compositional generalization.

Recent work has shown that targeted fine-tuning can make small language models competitive with or even stronger than larger models on tool-calling benchmarks [1]. More broadly, the practical promise of smaller open models has been reinforced by model families such as LLaMA [2], instruction-tuning efforts such as Stanford Alpaca [3], and more recent open model releases such as Qwen3 [4]. However, strong benchmark performance does not by itself establish systematic compositionality. If evaluation data contains many local action patterns also seen during training, a model may appear competent while relying primarily on memorized transitions rather than genuine multi-step generalization.

This concern mirrors prior work on compositional generalization in NLP. Benchmarks such as SCAN [5] and COGS [6] were designed to distinguish memorization of local templates from more systematic generalization to withheld structures. Inspired by that line of work, we adapt the same basic philosophy to tool-sequence generation. Instead of holding out semantic constructions, we hold out *adjacent tool transitions*. This lets us ask whether a model that has seen the individual tools and analogous transitions between different tool pairs can compose tool sequences in unseen orders. Thus, our contribution is a controlled evaluation framework and empirical analysis of what current SFT buys under a fixed syntax controller: better local semantic decisions, better continuation, or genuinely improved compositional generalization.

3 Approach

We study structured instruction-to-program generation. Given a natural-language instruction, the model must output a bracketed action program

$$[a_1 : v_1][a_2 : v_2] \dots [a_T : v_T],$$

where $a_t \in \{\text{forward, backward, left, right, bark}\}$. Distance bins are $\{10, 30, 60, 100\}$, turn-angle bins are $\{15, 45, 90, 180\}$, and bark always takes value 0. Programs have length $T \in [1, 4]$. Outputs must strictly conform to the DSL grammar: concatenated bracketed calls with no extra tokens and only valid tool/value combinations.

Original contribution. Our primary methodological contribution is a controlled adjacent-bigram generalization split, inspired by prior compositional benchmarks such as COGS and SCAN, that removes specific tool transitions from training targets while preserving the individual availability of the corresponding tools. This isolates whether models rely on local bigram memorization or can systematically compose tool sequences. Our original experimental contributions are the controlled DSL benchmark, the held-out adjacent-pair split, the corrected length-matched held-control set, and the continuation-targeted later-step-loss-weighting ablation. We use standard pretrained Qwen3 checkpoints, LoRA-based parameter-efficient fine-tuning via Hugging Face/PEFT, and a standard causal language modeling objective rather than proposing a new model architecture. We implemented the synthetic data generator, split construction, constrained decoding mask, evaluation scripts, and later-step loss weighting ourselves.

3.1 Evaluation splits

We construct four splits:

1. **IID:** Train and test are drawn from the same distribution.
2. **Length generalization:** Evaluation uses the subset of IID test containing exclusively programs of length 2–4.
3. **Held-out adjacent pair:** Test examples require at least one forbidden transition, from the set of tool bigrams excluded from IID training data.

4. **Held-control:** A non-forbidden control split drawn from the remaining test pool and exactly length-matched to `held_test`.

The ordered adjacent tool transitions held out of the IID training targets are:

```
left→forward, right→forward, forward→left, forward→right,
backward→left, backward→right, forward→bark, left→bark.
```

3.2 Prompting and decoding

Current evaluation prompts use the template

```
Instruction: {instr}
Action sequence:
```

with few-shot demonstrations, when present, formatted as

```
Instruction: {instr}
Action sequence: {prog}
```

and accompanied by a fixed task specification:

```
You are converting instructions into an action sequence program.
Output only the program, with no extra text.
Program format: concatenate bracketed calls with no separators.
Each call format: [tool:value]
Allowed tools: forward, backward, left, right, bark
Allowed values for forward/backward: 10, 30, 60, 100
Allowed values for left/right: 15, 45, 90, 180
Allowed value for bark: 0
Example format only: [left:45][forward:60][bark:0]
```

Constrained decoding (syntax control). To remove surface-form failures as a confounding variable, we apply constrained decoding via a token-level finite-state mask that restricts generations to DSL-valid continuations. Because constrained decoding controls syntax, improvements from prompting or SFT should be interpreted as changes in *semantic sequence prediction* (tool/value selection and when to stop), not learning the bracketed format itself.

4 Experiments

4.1 Dataset

All quantitative results reported in the main paper use the same shared synthetic dataset. The dataset is procedurally generated by first sampling symbolic action programs and then paraphrasing them into natural-language instructions. Each row contains an instruction, a canonical DSL program, the underlying action list, the program length, and a template identifier.

The training set contains 12,000 IID examples, with a 1,500-example IID dev set used during SFT. Evaluation uses four splits: `iid_test` (1500), `len_test` (1100), `held_test` (601), and `held_control` (601). Pretrained prompt-only baselines use few-shot demonstrations drawn from `iid_train`; SFT models are trained on `iid_train` and evaluated zero-shot on all four splits.

The corrected `held_control` split is exactly length-matched to `held_test`, making the `held_test` versus `held_control` comparison a cleaner test of whether unseen adjacent transitions create additional difficulty beyond sequence length alone. Full split length distributions and split-construction details are provided in Appendix A.

4.2 Models

We evaluate pretrained Qwen3-Instruct models at 0.6B, 1.7B, and 4B parameters under 0-, 4-, and 16-shot prompting. We chose these models to provide a controlled scale sweep within a single model

family, balancing experimental tractability with enough variation to test whether improvements in compositional tool-sequence generation come primarily from supervised fine-tuning or from increased model capacity. The 0.6B and 1.7B models are representative of the small-language-model regime most relevant for low-cost agentic deployment, while 4B provides a higher-capacity comparison point without introducing cross-family architectural differences.

We also report IID-trained SFT results for Qwen3-0.6B, Qwen3-1.7B, and Qwen3-4B. In addition, we include a continuation-targeted later-step-loss-weighting ablation on Qwen3-0.6B, motivated by early runs indicating that the main bottleneck was continuation and later-step correctness rather than first-step tool identification.

4.3 Metrics

Our primary metric is **exact match** (EM): the full predicted program must match the gold program exactly, including magnitudes. We further report diagnostic metrics that disentangle failure modes:

- **Exact match ignoring magnitude:** full tool sequence must match, but magnitudes are ignored.
- **Step F1:** position-wise action F1 including magnitudes.
- **Step F1 ignoring magnitude:** position-wise tool-only F1.
- **First-call accuracy:** first predicted action must exactly match the first gold action.
- **First-tool accuracy:** first predicted tool must match, ignoring magnitude.
- **Length accuracy:** predicted number of actions matches the gold length.
- **Tool edit distance:** Levenshtein edit distance between predicted and gold tool sequences, ignoring magnitudes.

Why evaluate magnitudes separately from tools? Correct tool identity alone is not sufficient for successful tool-sequence generation in our DSL. A prediction such as `[forward:30]` instead of `[forward:60]` is still semantically wrong, even though the tool is correct, because the argument determines the executed action. We therefore report both tool-only metrics and magnitude-sensitive metrics to separate two capabilities: choosing the correct action type and grounding that action in the correct discrete argument value. Given our enforcement of syntactic validity and legal tool-value pairings via constrained decoding, remaining errors reflect semantic failures.

4.4 Experimental details

All main SFT models are fine-tuned exactly once on the same IID training split and then evaluated zero-shot on every test split. In other words, the 0.6B, 1.7B, and 4B SFT models are all trained only on `iid_train`; none are separately retrained for `len_test`, `held_test`, or `held_control`. This makes all reported OOD results true transfer evaluations from a single IID SFT run per model.

Training examples, formatted with 'Instruction:' and 'Action sequence:' like evaluation prompts, are and optimized with standard causal language modeling loss, masking prompt tokens so that training focuses on target program tokens. Prompt-only pretrained evaluations use greedy decoding with constrained generation, `temperature=0.0`, `max_new_tokens=64`, the fixed task specification described above, and 0/4/16-shot prompting with diverse few-shot selection. All SFT runs use the same prompt framing, output-only loss masking, LoRA-based fine-tuning, and zero-shot constrained decoding at evaluation.

Each run used LoRA with $r = 16$, $\alpha = 32$, `dropout = 0.05`, and adapters to the attention projection layers (`q_proj`, `k_proj`, `v_proj`, `o_proj`) and feed-forward layers (`gate_proj`, `up_proj`, `down_proj`) of the base model. Each SFT run also used 3 epochs, learning rate 2×10^{-5} , per-device batch size 8, gradient accumulation 4, effective batch size 32, maximum length 128, bf16 enabled, and epoch-level evaluation/checkpointing. In the continuation-targeted ablation (see: 5.6), we keep the base IID SFT setup fixed and change only the later-step loss weights.

Model + shots	Avg EM	Avg EM _{¬mag}	Avg Step F1	Avg Step F1 _{¬mag}	Avg LenAcc	Avg ToolED
Qwen3-0.6B, 0-shot	0.0090	0.0436	0.0445	0.2008	0.0824	2.3995
Qwen3-0.6B, 4-shot	0.0132	0.0569	0.0987	0.4301	0.0707	2.1083
Qwen3-0.6B, 16-shot	0.0138	0.0549	0.1055	0.4283	0.0679	2.1178
Qwen3-1.7B, 0-shot	0.0057	0.0270	0.0420	0.2176	0.0667	2.4291
Qwen3-1.7B, 4-shot	0.0127	0.0537	0.0947	0.4276	0.0667	2.1341
Qwen3-1.7B, 16-shot	0.0127	0.0537	0.0947	0.4276	0.0667	2.1394
Qwen3-4B, 0-shot	0.0117	0.0538	0.0923	0.4268	0.0667	2.1550
Qwen3-4B, 4-shot	0.0163	0.0578	0.1087	0.4592	0.0958	2.3224
Qwen3-4B, 16-shot	0.0120	0.0543	0.1031	0.4591	0.1037	2.1675

Table 1: Prompt-only pretrained averages across the four evaluation splits. EM_{¬mag} and Step F1_{¬mag} ignore magnitudes. ToolED is tool edit distance, where lower is better.

5 Analysis

Main question. Our primary question is whether SFT improves *compositional generalization* of tool sequences, as measured by exact-match performance on the held-out adjacent-pair split. Because constrained decoding controls DSL validity, the relevant gains or failures reflect semantic sequence prediction and continuation rather than formatting.

5.1 Prompt-only pretrained baselines

Table 1 summarizes prompt-only pretrained performance averaged across the four evaluation splits. These baselines are extremely weak at end-to-end multi-step generation. The strongest average prompt-only configuration by exact match is Qwen3-4B at 4 shots, but it still reaches only 1.63% exact match, 5.78% exact match ignoring magnitude, 9.58% length accuracy, and 2.3224 tool edit distance. Few-shot prompting improves local tool identification much more than full-program generation, so prompt-only baselines are useful mainly as first-step baselines rather than competitive solutions for compositional multi-step generation.

Split-wise pretrained results reinforce this interpretation. Nonzero exact match is concentrated entirely on `iid_test`; all prompt-only models score 0.0000 exact match on `len_test`, `held_test`, and `held_control`. At the same time, 4-shot prompting often yields high first-tool accuracy, indicating that prompt-only models frequently identify the correct initial action type while failing on continuation and full-program structure. Full split-wise pretrained tables are deferred to Appendix C.

5.2 SFT improves end-to-end performance, but the main compositional result remains negative

Table 2 reports split-level results for the main unweighted IID-trained SFT models. Relative to prompt-only baselines on the same dataset, SFT improves every important end-to-end metric by a large margin. Averaged over the four splits, Qwen3-0.6B SFT reaches 4.76% exact match, 37.15% exact match ignoring magnitude, 46.27% length accuracy, and 1.0785 tool edit distance, while Qwen3-1.7B SFT is similar overall and slightly stronger on tool-only step metrics. Qwen3-4B SFT is a negative result: despite strong first-step behavior, it remains weak end-to-end, with only 3.11% average exact match, 10.55% length accuracy, and 1.9547 tool edit distance.

Our main compositionality result comes from `held_test`. On this split, where correct generation requires unseen adjacent tool transitions, exact match remains very low even after SFT: 0.67% for Qwen3-0.6B, 1.50% for Qwen3-1.7B, and 0.50% for Qwen3-4B. Thus, IID SFT substantially improves structured sequence generation, but does not yield robust systematic composition of unseen transitions.

The other OOD splits show that this is not only a forbidden-transition problem. On `len_test`, exact match drops sharply relative to IID for all three SFT models, showing that longer unseen programs alone are enough to degrade strict end-to-end accuracy. The exactly length-matched `held_control` split provides a cleaner comparison: for the 0.6B and 1.7B SFT models, performance is clearly better on `held_control` than on `held_test`, which strengthens the interpretation that unseen adjacent transitions create additional difficulty beyond sequence length alone.

Model	Split	EM	EM _{mag}	Step F1	F1 _{mag}	LenAcc	ToolED
Qwen3-0.6B SFT	iid_test	0.1240	0.5333	0.2815	0.7719	0.6047	0.7827
Qwen3-0.6B SFT	len_test	0.0282	0.3636	0.2430	0.6889	0.4609	1.0673
Qwen3-0.6B SFT	held_test	0.0067	0.2729	0.1997	0.6407	0.3710	1.2696
Qwen3-0.6B SFT	held_control	0.0316	0.3161	0.2625	0.6692	0.4143	1.1947
Qwen3-1.7B SFT	iid_test	0.1220	0.5133	0.2946	0.7797	0.5447	0.8213
Qwen3-1.7B SFT	len_test	0.0264	0.3491	0.2617	0.7044	0.3909	1.1073
Qwen3-1.7B SFT	held_test	0.0150	0.3594	0.2204	0.6942	0.4010	1.1963
Qwen3-1.7B SFT	held_control	0.0266	0.2745	0.2744	0.6691	0.3195	1.3328
Qwen3-4B SFT	iid_test	0.1053	0.2987	0.2540	0.6583	0.3000	1.4407
Qwen3-4B SFT	len_test	0.0091	0.0445	0.2118	0.5350	0.0455	1.9636
Qwen3-4B SFT	held_test	0.0050	0.0466	0.1605	0.5050	0.0483	2.1913
Qwen3-4B SFT	held_control	0.0050	0.0283	0.2177	0.4949	0.0283	2.2230

Table 2: Split-level results for the main unweighted SFT models. EM_{mag} and F1_{mag} ignore magnitudes. LenAcc denotes length accuracy. ToolED is tool edit distance, where lower is better.

Metric	Best prompt-only	Best SFT
Avg EM	0.0163	0.0520
Avg EM _{mag}	0.0578	0.4353
Avg Step F1	0.1087	0.2611
Avg Step F1 _{mag}	0.4592	0.7520
Avg Length Acc	0.0958	0.5263
Avg ToolED	2.3224	0.8757

Table 3: Same-dataset comparison between the best prompt-only baseline (Qwen3-4B @ 4) and the best current SFT result (later-step-weighted Qwen3-0.6B). ToolED is tool edit distance, where lower is better.

5.3 Same-dataset comparison: SFT is far stronger than prompt-only

The clearest empirical comparison in the project is same-dataset prompt-only versus SFT. The best average prompt-only configuration is Qwen3-4B at 4 shots, while the best overall SFT result is the later-step-weighted Qwen3-0.6B run (Table 3). On the same dataset, SFT improves strict exact match by more than 3 $\times$, exact match ignoring magnitude by roughly 8 $\times$, length accuracy by more than 5 $\times$, and reduces average tool edit distance by more than 1.4 edits.

5.4 Failure analysis: continuation is the central bottleneck

Position-wise analysis makes the main failure mode especially clear. In the prompt-only regime, models often achieve high first-tool accuracy with 4-shot prompting, yet still have 0.0000 exact match on every OOD split and near-zero length accuracy there. This indicates that identifying the correct first tool is much easier than sustaining the correct full program.

SFT improves later-step continuation substantially, but still shows a tradeoff between continuation and conditional later-step accuracy. Table 4 shows that Qwen3-0.6B SFT produces later actions more often, whereas Qwen3-1.7B SFT is more accurate conditional on producing them. Qwen3-4B SFT shows the clearest continuation-collapse failure: it is near-perfect on first-tool prediction at position 1, but rarely continues to positions 2 and 3. One plausible interpretation is that the larger model is highly confident about the first local action type under constrained decoding while remaining over-eager to stop, producing a continuation-collapse pattern rather than a steady improvement in full-sequence generation.

5.5 Qualitative error examples

Raw predictions show three recurring failure modes. First, many models produce only a short prefix of the correct program and then stop; for example, Qwen3-4B SFT predicts [forward:10] for the gold sequence [forward:100] [backward:60] [left:180] [right:15]. Second, even

Model / Pos.	Pred present	Call acc	Tool acc	Call acc present	Tool acc present
0.6B SFT @ 1	1.0000	0.3462	0.8764	0.3462	0.8764
0.6B SFT @ 2	0.8287	0.2061	0.6348	0.2487	0.7658
0.6B SFT @ 3	0.5035	0.1212	0.3409	0.2443	0.6816
1.7B SFT @ 1	1.0000	0.3721	0.9720	0.3721	0.9720
1.7B SFT @ 2	0.5714	0.1635	0.4790	0.2862	0.8380
1.7B SFT @ 3	0.4746	0.1665	0.4129	0.3521	0.8700
4B SFT @ 1	1.0000	0.3791	0.9965	0.3791	0.9965
4B SFT @ 2	0.0438	0.0261	0.0410	0.6317	0.9306
4B SFT @ 3	0.0204	0.0076	0.0204	0.3670	1.0000

Table 4: Position-wise falloff for SFT models, averaged over the four evaluation splits.

when the tool sequence is structurally correct, magnitudes often collapse toward the smallest bins, e.g. Qwen3-0.6B SFT predicts [forward:10] [right:15] for gold [forward:100] [right:45]. Third, later actions are sometimes replaced by a spurious but syntactically valid tool call, often bark; for example, Qwen3-0.6B SFT predicts [forward:10] [backward:10] [bark:0] for gold [forward:100] [backward:60] [left:180] [right:15]. These examples match the quantitative picture: the models frequently identify an appropriate first action, but then either stop too early, under-ground magnitudes, or derail late in the sequence.

5.6 Ablation: later-step loss weighting as a continuation-targeted follow-up

Because early results suggested that the main bottleneck was continuation rather than first-step tool identification, we ran a targeted follow-up ablation on Qwen3-0.6B that increased the loss weight on later action positions:

$$\text{step_weights} = [1.0, 1.35, 1.8, 2.2].$$

As shown in Table 5, this intervention improves the original unweighted 0.6B model on every evaluated split. Averaged over the four splits, it improves exact match by +0.0044, exact match ignoring magnitude by +0.0638, step F1 by +0.0145, step F1 ignoring magnitude by +0.0593, length accuracy by +0.0636, and tool edit distance by -0.2028.

Figure 1 shows the training curve for this later-step-weighted 0.6B run. The optimization makes most of its progress in the first epoch: the training loss drops sharply early on, while validation loss is already near its eventual minimum by the first epoch boundary and changes only marginally afterwards. This matches our checkpoint-selection result that the epoch-1 checkpoint was the strongest on downstream dev metrics, so the ablation gain is not simply a consequence of training longer.

These gains are most visible on the OOD splits, which suggests that poor held_test performance is driven partly by a broader continuation bottleneck rather than only by failure to compose unseen adjacent transitions. At the same time, held-out adjacent-pair exact match remains very low, so the main conclusion is unchanged: targeting later-step prediction helps multi-step generation, but does not by itself produce robust compositional generalization.

6 Conclusion

We investigated whether supervised fine-tuning enables small language models to perform compositional tool-sequence generation rather than relying on local action-pattern memorization. Using a controlled instruction-to-program benchmark on a single shared dataset, we evaluated prompt-only and SFT models on IID, length-generalization, held-out adjacent-pair, and held-control splits. Because constrained decoding controls DSL validity, we interpret results in terms of semantic sequence prediction rather than formatting.

On the shared dataset, SFT strongly outperforms prompt-only baselines on exact match, ignore-magnitude exact match, step-level F1, tool edit distance, and length control. The cleanest same-dataset result is that the best current SFT model outperforms the best prompt-only baseline by more than $3\times$ on strict exact match, more than $5\times$ on length accuracy, and by more than 1.4 edits on average tool edit distance. However, the main compositionality result remains negative. On the held-out

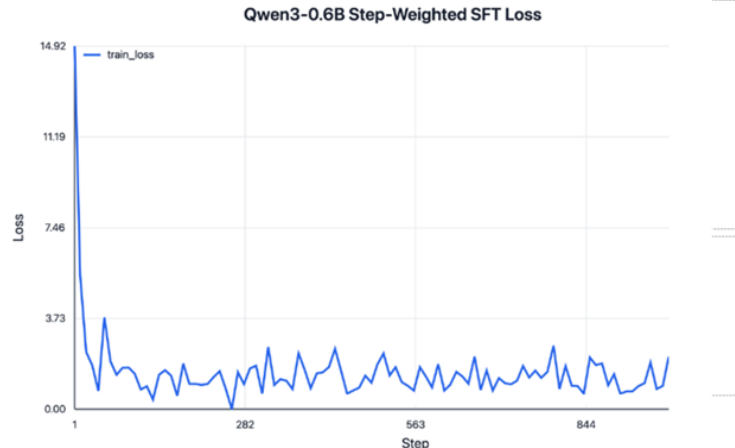


Figure 1: Training curve for the continuation-targeted later-step-weighted Qwen3-0.6B run. The curve shows rapid early optimization and only minor validation-loss changes after the first epoch, consistent with our choice of the epoch-1 checkpoint for final evaluation.

Split	Model	EM	EM $\neg_{mag}$	Step F1	Step F1 $\neg_{mag}$	LenAcc	ToolED
iid_test	0.6B unweighted	0.1240	0.5333	0.2815	0.7719	0.6047	0.7827
iid_test	0.6B step-weighted	0.1253	0.5793	0.2896	0.8140	0.6567	0.6400
iid_test	Δ	+0.0013	+0.0460	+0.0081	+0.0421	+0.0520	-0.1427
len_test	0.6B unweighted	0.0282	0.3636	0.2430	0.6889	0.4609	1.0673
len_test	0.6B step-weighted	0.0345	0.4264	0.2586	0.7463	0.5318	0.8727
len_test	Δ	+0.0064	+0.0627	+0.0156	+0.0574	+0.0709	-0.1945
held_test	0.6B unweighted	0.0067	0.2729	0.1997	0.6407	0.3710	1.2696
held_test	0.6B step-weighted	0.0133	0.3627	0.2231	0.7153	0.4609	1.0200
held_test	Δ	+0.0067	+0.0899	+0.0235	+0.0745	+0.0899	-0.2496
held_control	0.6B unweighted	0.0316	0.3161	0.2625	0.6692	0.4143	1.1947
held_control	0.6B step-weighted	0.0349	0.3727	0.2731	0.7325	0.4559	0.9700
held_control	Δ	+0.0033	+0.0566	+0.0106	+0.0633	+0.0416	-0.2246

Table 5: Per-split ablation for later-step loss weighting on Qwen3-0.6B. Positive deltas indicate improvement, except for ToolED where lower is better. LenAcc denotes length accuracy.

adjacent-pair split, where correct generation requires unseen tool transitions, exact match remains very low even after SFT. Among the main unweighted SFT models, the best held-out adjacent-pair exact match is only 1.50%.

The corrected, exactly length-matched `held_control` split strengthens this conclusion. For the main 0.6B and 1.7B SFT models, performance on `held_control` is clearly better than on `held_test`, even though the two splits now have identical length distributions. We thus infer that the remaining gap reflects difficulty with unseen adjacent tool transitions rather than a simple length artifact.

The continuation-targeted later-step-weighted 0.6B ablation is the strongest positive follow-up result: it improves every evaluated split over the original unweighted 0.6B run, including the held-out adjacent-pair split. This suggests that the highest-leverage next direction is not larger model scale or prompt tuning, but training and checkpoint-selection strategies that explicitly target continuation and later-step correctness.

7 Team Contributions

Maanit Goel co-designed the experimental procedure and code architecture, conducted the central literature review, and led experimental result analysis and paper write-up. Benji Warburton co-designed the experimental procedure and code architecture, executed train and test runs, and contributed to analysis and write-up. All central components of the project were jointly conducted with equal contribution.

References

- [1] Polaris Jhandi, Owais Kazi, Shreyas Subramanian, and Neel Sendas. Small language models for efficient agentic tool calling: Outperforming large models with targeted fine-tuning, 2025.
- [2] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [3] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. GitHub repository.
- [4] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [5] Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2873–2882. PMLR, 2018.
- [6] Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105. Association for Computational Linguistics, 2020.

A Dataset split construction and length distributions

The evaluation uses four splits: `iid_test`, `len_test`, `held_test`, and `held_control`.

The `len_test` split is derived from the IID partition by removing all length-1 examples from the IID test pool; correspondingly, the associated length-generalization training pool contains only one-step programs. This split therefore tests whether a model trained only on one-step programs can generalize to longer programs.

The `held_test` split is constructed so that every example contains at least one adjacent tool transition that is excluded from the relevant training targets. Our original `held_control` construction was generated by taking held-out-adjacent-pair-related examples and removing forbidden adjacent transitions. In practice, this produced a non-forbidden comparison set, but it did not match the length distribution of `held_test`. This mattered because longer synthetic sequences are more likely to contain a held-out adjacent transition somewhere in the program, so `held_test` skewed longer than the original control split. As a result, sequence length was not properly controlled for in `held_test` versus `held_control` comparisons.

After identifying this issue, we regenerated `held_control` to exactly match the length distribution of `held_test` while still excluding forbidden adjacent transitions. The corrected `held_control` therefore serves as a cleaner control: it matches `held_test` in sequence-length profile but does not require unseen adjacent tool transitions. This makes the remaining performance gap between `held_test` and `held_control` more interpretable as evidence of difficulty with unseen transitions rather than a simple length artifact.

B Why magnitude-sensitive evaluation matters

Our DSL requires not only choosing the correct action type but also grounding that action in the correct discrete argument bin. For motion actions, the magnitude determines how far or by what angle

Split	n	Avg len	Len 1	Len 2	Len 3	Len 4
iid_train	12000	2.4905	3073	2966	2963	2998
iid_test	1500	2.4880	400	358	352	390
len_test	1100	3.0291	0	358	352	390
held_test	601	3.2762	0	112	211	278
held_control	601	3.2762	0	112	211	278

Table 6: Split sizes and length distributions. The corrected held_control split is exactly length-matched to held_test.

the action is executed. A model that predicts the right tool but the wrong value has therefore only partially solved the task. For example, [left:45] and [left:180] share the same tool identity but correspond to materially different actions.

This distinction is especially important because constrained decoding already guarantees syntactic well-formedness and valid tool-value compatibility. The remaining errors are therefore semantic: either the model chose the wrong action type, or it chose the correct action type but failed to ground the correct argument. Reporting both ignore-magnitude and strict metrics lets us distinguish these cases. In practice, the gap between these metrics is informative about whether a model is learning the compositional action scaffold alone or also the finer-grained parameterization required for fully correct execution.

C Full split-wise pretrained results

For completeness, we report the full split-wise pretrained prompt-only results here. These tables support the main-text claim that prompt-only gains are concentrated in first-step identification and do not translate into robust OOD end-to-end sequence generation.

Split	Metric	Qwen3-0.6B			Qwen3-1.7B			Qwen3-4B		
		0-shot	4-shot	16-shot	0-shot	4-shot	16-shot	0-shot	4-shot	16-shot
iid_test	EM	0.0360	0.0527	0.0553	0.0227	0.0507	0.0507	0.0467	0.0653	0.0480
iid_test	EM _{mag}	0.1520	0.2173	0.2153	0.1080	0.2147	0.2147	0.2153	0.2313	0.2173
iid_test	Step F1	0.0643	0.1221	0.1316	0.0505	0.1173	0.1173	0.1109	0.1393	0.1197
iid_test	F1 _{mag}	0.2784	0.5177	0.5162	0.2587	0.5159	0.5159	0.5158	0.5548	0.5397
len_test	EM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
len_test	EM _{mag}	0.0091	0.0036	0.0009	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
len_test	Step F1	0.0385	0.0947	0.1041	0.0380	0.0908	0.0908	0.0876	0.1008	0.0978
len_test	F1 _{mag}	0.1814	0.4132	0.4112	0.2055	0.4107	0.4107	0.4097	0.4410	0.4396
held_test	EM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
held_test	EM _{mag}	0.0083	0.0050	0.0017	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
held_test	Step F1	0.0341	0.1003	0.1111	0.0477	0.0955	0.0955	0.0904	0.1046	0.0942
held_test	F1 _{mag}	0.1895	0.4349	0.4321	0.2501	0.4312	0.4312	0.4304	0.4571	0.4565
held_control	EM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
held_control	EM _{mag}	0.0050	0.0017	0.0017	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
held_control	Step F1	0.0412	0.0777	0.0752	0.0318	0.0750	0.0750	0.0803	0.0903	0.1008
held_control	F1 _{mag}	0.1540	0.3546	0.3536	0.1561	0.3526	0.3526	0.3515	0.3839	0.4005

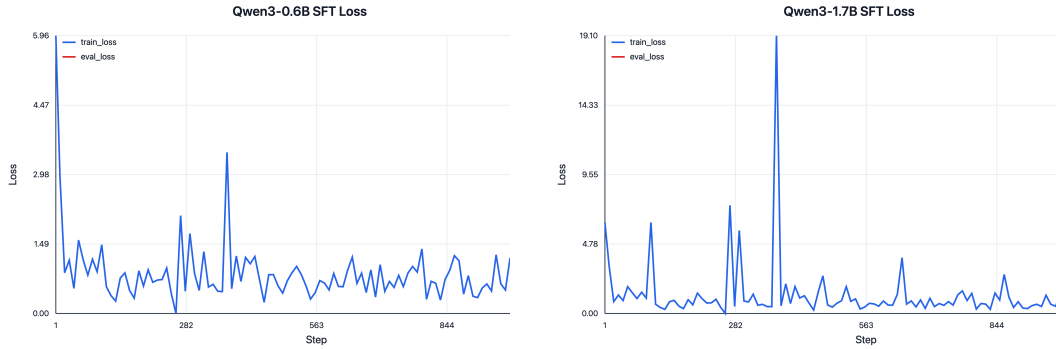
Table 7: Split-wise pretrained prompt-only performance across model sizes and shot counts. EM_{mag} and F1_{mag} ignore magnitudes.

D SFT training curves

For completeness, we include the remaining SFT training curves here. The main-text figure focuses on the step-weighted 0.6B ablation because it directly supports the continuation-targeted intervention. The appendix curves provide context for the other SFT runs and make it easier to compare optimization dynamics across model size and training condition.

Split	Metric	Qwen3-0.6B			Qwen3-1.7B			Qwen3-4B		
		0-shot	4-shot	16-shot	0-shot	4-shot	16-shot	0-shot	4-shot	16-shot
iid_test	1st call	0.0840	0.1827	0.2020	0.0780	0.1787	0.1787	0.1733	0.1967	0.1813
iid_test	1st tool	0.3747	0.7960	0.7967	0.4007	0.7967	0.7967	0.7960	0.8127	0.7993
iid_test	LenAcc	0.2800	0.2700	0.2673	0.2667	0.2667	0.2667	0.2667	0.2907	0.2927
len_test	1st call	0.0655	0.1773	0.2000	0.0755	0.1745	0.1745	0.1727	0.1791	0.1818
len_test	1st tool	0.3127	0.7927	0.7936	0.3991	0.7936	0.7936	0.7918	0.7927	0.7936
len_test	LenAcc	0.0182	0.0045	0.0009	0.0000	0.0000	0.0000	0.0000	0.0327	0.0355
held_test	1st call	0.0549	0.1947	0.2280	0.0998	0.1963	0.1963	0.1880	0.1980	0.1880
held_test	1st tool	0.3344	0.8769	0.8785	0.5042	0.8785	0.8785	0.8769	0.8785	0.8785
held_test	LenAcc	0.0183	0.0067	0.0017	0.0000	0.0000	0.0000	0.0000	0.0216	0.0283
held_control	LenAcc	0.0133	0.0017	0.0017	0.0000	0.0000	0.0000	0.0000	0.0383	0.0582

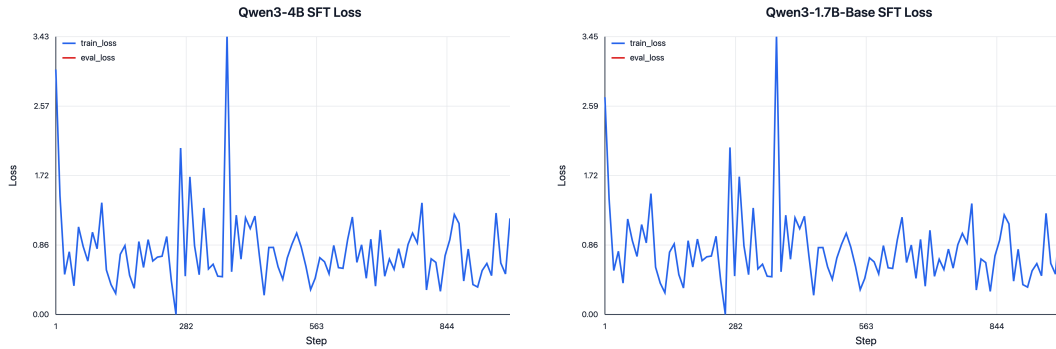
Table 8: Split-wise pretrained prompt-only diagnostic metrics across model sizes and shot counts. LenAcc denotes length accuracy.



(a) Qwen3-0.6B IID SFT

(b) Qwen3-1.7B IID SFT

Figure 2: Training curves for the unweighted IID-trained Qwen3-0.6B and Qwen3-1.7B SFT runs. Both models learn rapidly in the first epoch, but the 1.7B run remains noisier at the batch level while still converging to a low validation loss.



(a) Qwen3-4B IID SFT

(b) Qwen3-1.7B-Base IID SFT

Figure 3: Training curves for the Qwen3-4B IID SFT run and the Qwen3-1.7B-Base ablation. The 4B run fits the training objective but still collapses on continuation at evaluation time, reinforcing that lower LM loss alone is not sufficient for strong multi-step tool-sequence generation.