

A Training Details

Hyper-parameters. We train all three customized models (Loc-Gen, Val-no-assert, Val-assert) using the same settings. Table 2 summarizes the key hyper-parameters.

Table 2: Training hyper-parameters for the Qwen-2.5-Coder-14B model

Hyperparameter	Value
Peak learning rate	1×10^{-5}
Warmup ratio	0.10 of total steps
LR scheduler	Cosine decay (with 10% linear warmup)
Batch size (per GPU)	1 (effective batch size = 1×12 accumulations)
Weight decay	1×10^{-5}
Number of training epochs	3.0
Maximum sequence length	32768 tokens

B Ablation Study

B.1 Whether Long Reasoning Help

Design. To compare the impact of reasoning length in the patching task, we ran an ablation study on the SWE-bench–Verified dataset using Claude-3.7-Sonnet. For a fair comparison and to focus specifically on patch generation capabilities, we use GPT-4o localization results as consistent input as 4.2 to patch generation. We then vary the maximum reasoning output length, setting it to 0 tokens ("no-reason"), 2K tokens ("short-reason") and 8K tokens ("long-reason"), and evaluate performance with the pass@1 metric, which counts issues successfully resolved by the first generated patch.

Results. The pass@1 results were 43.8% for the short reasoning setting(2K) and 44.2% for the long reasoning setting(8K), while 40.6% for the no-reasoning settings. These results show that considering the random deviation of the model inference, extending the reasoning budget beyond 2K tokens yields no appreciable gain in our experiments.

B.2 Localization

We use our localization model to conduct the ablation study on data size, model size, and testing-phase scaling strategy.

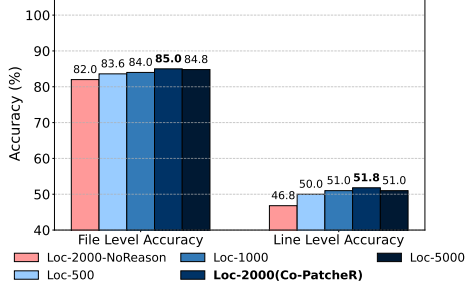
Data size. We randomly sample a subset of 500, 1K, and 2K cases from our 5K training set and train four models in total using our recipe. We report both the file-level and line-level localization performance of these models in Figure 6a. The result shows that the performance keeps increasing from 500 to 1K and from 1K to 2K, and the model performance for localization no longer increases as we further increase the training data to 5K. As such, we select 2K as our final training data size. We further train a non-reasoning model for localization (SFT with ground truth files and lines). Our result shows that a non-reasoning model trained with 2K training data performs even worse than our reasoning model trained with 500 samples. It further shows that the reasoning model is more data-efficient.

Model size. We change our base model to Qwen-2.5-Coder-7B and Qwen-2.5-Coder-32B (same model family with different sizes) and retrain our model with the same localization training data. The localization results in Figure 6b show that a larger model indeed improves the performance. However, considering that the improvement of the 32B model over the 14B model is not significant, we still choose the smaller one.

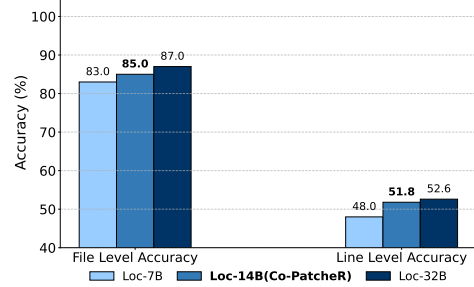
Testing-phase scaling. We test two scaling performances here. The generation setting is kept fixed, whereas the localization model predicts $N=1, 2, 4, 6, 8, 10$ times for each issue, including both line-level and file-level. For each setting, we 1) fix the top 5 files as file-level results and merge the line-level localization outputs as the line-level results accuracy. 2) generate 20 patches with the generation model with the different localization results as input, then record the pass@20 resolved rate(the percentage of which at least one of the 20 candidate patches fixes the issue)—this serves as

Table 3: Effect of reasoning length on patch generation (pass@1) on the SWE-bench–Verified dataset.

Setting	Max reasoning tokens	pass@1 (%)
No-reason	0	40.6
Short-reason	2K	43.8
Long-reason	8K	44.2



(a) The top@5 file- and line-level accuracy versus training data size.



(b) The top@5 file- and line-level accuracy versus model size.

Figure 6: Ablation studies on data size (A) and model size (B) for localization.

the upper bound of the localization module. To clarify the relationship between our localization and patching metrics, we first note that line-level localization accuracy is defined by whether the model’s returned lines fully cover all modified lines in the golden patch—only perfect coverage counts as “correct”. In other words, our line-level metric is very strict: it says a location is “correct” only if the predicted lines match the golden patch line for line. But an issue can be fixed without copying the golden patch exactly—any edit that makes the tests pass is accepted. So a patch can solve the issue even when its edited lines do not align with the golden patch, which is why the *pass@20* success rate can be higher than the measured line-level accuracy. As illustrated in Figure 7, supplying more localization samples broadens the search space, boosting the chance that the correct file appears in the *pass@20* candidates. This confirms our intuition that “wider” localization (more samples) can offset a smaller model size just as “wider” generation does, without incurring the cost of a larger backbone. However, pushing the number N too high again shifts the burden to generation and can confuse the generation by a large number of false positives of code chunks, resulting in a decrease in performance.

B.3 Validation

Combine training. In the original pipeline, we train two separate PoC-generation models—Val-assert model and Val-no-assert model—and use them to create four PoCs (two of each type) for validating the candidate patches produced by the generator. To test whether a single, mixed validator can do better, we merge the two training sets, train one unified model, and invoke it with two prompt templates (assert / no-assert) to generate the same four PoCs. Everything else is held constant: the localization and generation results, the dynamic tester, and the ranking heuristic. We then compare the *best@20* resolved rate after validation has selected the final patch as our result. Table 4 indicates that keeping the PoC generation model separate still yields the higher accuracy; combining the data into a single model does not translate into better PoC generation due to a lack of diversity.

Table 4: Split vs. unified PoC-generation models (*best@20* on SWE-bench–Verified).

Training scheme	PoC prompts	best@20 (%)
Split models (Co-PatcheR)	2 assert + 2 no-assert	43.20
Unified model	2 assert + 2 no-assert	42.60

Data size. We randomly sample a subset of 500 and 1K from our 2K training set and train in total of three models for our *Val-no-assert* models training. We report the *best@20* final resolved rate performance of these models in Figure 8a. The result shows that the performance keeps increasing from 500 to 1K and from 1K to 2K. As such, we select 2K as our final training data size.

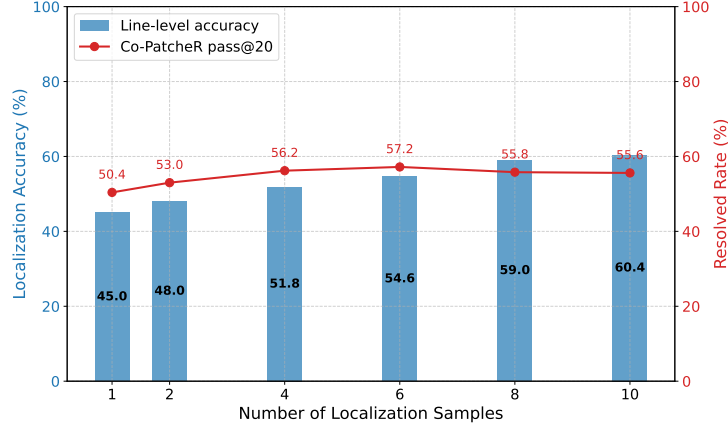


Figure 7: Resolved rate and line-level accuracy for # of localization samples.

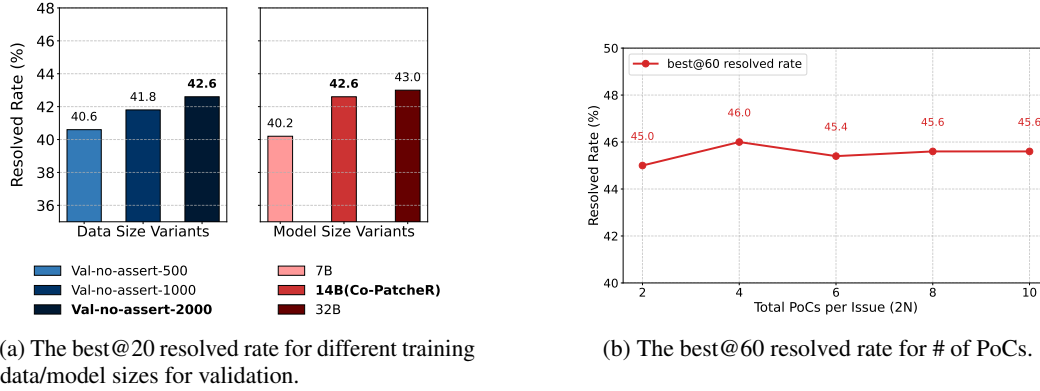


Figure 8: More ablation studies on the validation component.

947 **Model size.** We change our base model to Qwen-2.5-Coder-7B and Qwen-2.5-Coder-32B (same
 948 model family with different sizes) and retrain our *Val-no-assert* model with the same training data.
 949 The localization results in Figure 8a show that a larger model indeed improves the performance.
 950 However, considering that the improvement of the 32B model over the 14B model is not significant,
 951 we still choose the smaller one.

952 **Testing-phase scaling.** We reuse the same 60 candidate patches from the generation and vary only the
 953 number of PoCs supplied to the dynamic test. Specifically, we let both *Val-no-assert* and *Val-assert*
 954 generate $N=1, 2, 3, 4, 5$ PoCs each, yielding $2N=2, 4, 6, 8, 10$ PoCs per issue. For every $2N$, we
 955 measure the *best@60* (The final resolved rate of the patches that chosen from 60 cases) resolved rate
 956 after validation chooses a single patch from the 60 candidates. As shown in Figure 8b, increasing
 957 from $N=1$ (2 PoCs total) to $N=2$ (4 PoCs) per model brings a clear boost, but adding more PoCs
 958 provides no further gains. The plateau occurs because the same validation model tends to emit highly
 959 similar PoCs; once two distinct checks are present, additional ones rarely catch new failures and
 960 instead lengthen the test phase without improving accuracy.

961 C Majority Voting vs. ORMs

962 **Design.** After dynamic testing, we often obtain several patch candidates that all pass the same subset
 963 of PoCs and functionality tests. Our default system selects the final patch via majority voting. To
 964 assess whether more reranking methods help, we keep every other component fixed—the 60 patch
 965 candidates, the four PoCs used for dynamic validation, and all hyper-parameters from 4.1—and only
 966 swap the tie-breaking strategy: *Majority Vote* (what we use). *ORM Score*, we feed each surviving
 967 candidate into the ORM model from SWE-Gym [36] and pick the highest-scoring patch. *Claude*
 968 *Vote*, we prompt Claude-3.7-Sonnet with the issue description, the localization results, and the patch

969 candidates, and let it rank the patches; the top suggestion is selected. This setup isolates the impact
 970 of the tie-breaking itself, because every method starts from exactly the same candidate pool and the
 971 same dynamic-testing evidence.

972 **Results.** With 60 patching candidates, majority voting achieved the best accuracy (46.00% best@1),
 973 ahead of the ORM score (36.40%) and the Claude vote (45.00%). Because majority voting is cost-
 974 free(neither needs commercial credits nor computation resources), it remains the simplest and most
 975 effective option when many patches are still in play. Besides, its performance is better than even the
 976 SOTA commercial LLMs when dealing with a large pool, demonstrating its effectiveness.

Table 5: Effect of different tie-breaking strategies after dynamic testing.

Final patch selection	best@60 (%)
Majority Vote	46.00
ORM Score	36.40
Claude Vote	45.00

977 D Prompts for Each Component

978 D.1 Localization Prompt

979 D.1.1 File Localization Prompt

User Prompt:

Please look through the following GitHub problem description and Repository structure, and provide a list of files that one would need to edit to fix the problem.

```
### GitHub Problem Description ###
{problem_statement}
###
```

```
### Repository Structure ###
{structure}
###
```

After analyzing the problem, provide the full path and return at most {file_number} files. The returned files should be separated by new lines, ordered by most to least important, and wrapped with ``

For example:

```
``
file1.py
file2.py
``
```

980

981 D.1.2 Line Localization Prompt

User Prompt:

Please review the following GitHub problem description and relevant files, and provide a set of locations that need to be edited to fix the issue.

The locations should include exact line numbers that require modification.

Pay attention! You should identify the method responsible for the core functionality of the issue. Focus on areas that define or enforce foundational behavior rather than case-specific issues.

982

```
### GitHub Problem Description ###
{problem_statement}
```

```
###
{file_contents}
```

```
###

{last_search_results}
```

After analyzing the problem, please provide the class name, function or method name, or the exact line numbers that need to be edited.

If you want to provide the line number, please give me a number in the middle every time.

If you need to edit multiple classes or functions, please provide all the function names or the line numbers in the class.

You should always include a class or function; do not provide just the line numbers without the class or function name.

If you want to include a class rather than a function, you should always provide the line numbers for the class.

Here is the format you need to strictly follow, don't return any code content or other suggestions, don't forget the "```":

Examples:

```

```
full_path1/file1.py
class: MyClass1
line: 51
```

```
full_path2/file2.py
function: MyClass2.my_method
line: 12
```

```
full_path3/file3.py
function: my_function
line: 24
line: 156
```
```

983

984 **D.2 Generation Prompt**

985 **D.2.1 Patch generation Prompt**

User Prompt:

We are currently solving the following issue within our repository.

You are a maintainer of the project. Analyze the bug thoroughly and infer the underlying real problem, using your inherent knowledge of the project. Focus on resolving the root logic issue rather than suppressing symptoms.

Note that if the issue description mentions file names or arguments for reproduction, the fix must be generalized and not restricted to specific arguments. If the issue description includes a recommended fix, adapt it to align with the codebase's style and standards. Ensure your fix maintains structural integrity, considering interactions across code sections, nested structures, function calls, and data dependencies. Prefer solutions resilient to future structural changes or extensions.

986

The following is the issue description:

```
— BEGIN ISSUE —  
{problem_statement}  
— END ISSUE —
```

Below are the code segments from multiple files relevant to this issue. Each file is clearly marked. Decide carefully and only modify necessary segments. Preserve original indentation and formatting standards strictly.

```
— BEGIN FILES —  
{content}  
— END FILES —
```

Now, carefully analyze the files above. Determine which specific file segments require modifications and provide your edits using the following structured format for easy parsing:

```
<<< MODIFIED FILE: path/to/filename >>>  
```python  
<<<<<<< SEARCH
from flask import Flask
=====
import math
from flask import Flask
>>>>>>> REPLACE
<<< END MODIFIED FILE >>>
...
```

Please note that the \*SEARCH/REPLACE\* edit **REQUIRES PROPER INDENTATION**. If you would like to add the line ` print(x)`, you must fully write that out, with all those spaces before the code!  
Wrap the \*SEARCH/REPLACE\* edit in blocks ```python...```.

987

## 988 D.2.2 Patch Critique Prompt

### User Prompt:

Please continue working on this code patching work. You need to review the patch thoroughly to determine if it successfully fixes the issue without introducing any new bugs, and while handling all possible edge cases.

If you determine that the patch is correct and complete, tell me you confirm that the patch succeeded.

If you think the patch is incomplete, give me the reason and potential fixing suggestions.

You need to think:

1. What edge cases can break the patch? Consider complex cases such as nested structures and recursive patterns. For example, if the patch fixes an issue with an empty string, consider whether None, an empty list, or partially empty data structures might also trigger the bug.
2. Why the patch is incomplete or correct, whether the interaction between the patched part and other parts of the codebase can be handled properly
3. whether the patch only fixes the issue for the specific case mentioned in the issue description or for all similar cases
4. whether the patch follows the codebase's style and standards, using the proper variable types, error or warning types, and adhering to the established format

989

If the patch is perfect, tell me why. If the patch is unfinished or wrong, give me the reason and patch suggestions.  
At the end, you should give me the critical result from you, if yes, give me "Conclusion: Right", otherwise give me "Conclusion: Wrong".

990

### 991 D.3 Validation Prompt

#### 992 D.3.1 PoC Generation Prompt

##### **User Prompt:**

When generating a PoC script, follow these steps **\*\*in order\*\***:

[Optional] Always include assertions in the PoC to make the failure obvious when the script is executed.

[Optional] The assertion should fail if the bug is present, and pass if the bug is not present.

**\*\*Try to extract an existing PoC from the issue description\*\***

\* Scan the **\*\*GitHub issue description\*\*** for Python code blocks or inline snippets that appear to reproduce the bug.

\* If such a snippet exists, **\*\*use it verbatim as the base PoC\*\*** and only make the minimal edits needed to run:

- Remove interactive prompts (`>>>`, `\$`, `In [ ]:`) and any captured output lines.
- Add any missing `import` statements.
- Convert Python2 syntax to Python3, if present.
- Merge multiple fragments into a single runnable file in their original order.

**\*\*If no valid PoC can be extracted, write one yourself\*\***

\* Use the **\*specific\*** classes, functions, or code paths named in the issue to trigger the bug.

\* Keep the script minimal—just enough to demonstrate the failure (e.g., an `assert`, an expected exception, or a visibly incorrect result).

**\*\*General rules for both cases\*\***

\* The PoC **\*\*must be a single, self-contained Python3 file\*\***.

\* If the issue description includes other languages or shell commands, recreate their behavior in Python (e.g., with `subprocess` or file operations).

\* If the snippet refers to external files, create them program matically inside the script.

\* Always include `print()` or `assert` statements so the failure is obvious when the script is executed.

**\*\*Output format\*\***

Return **\*\*exactly\*\*** Python code wrapped in triple backticks, with no other text.

```
```python
{{poc_code here}}
```
```

```
Context Provided to You
{last_time_poc_code}
```

```
{execution_output}
```

```
GitHub Issue Description
— Begin Issue Description —
{problem_statement}
— End Issue Description —
```

993

**User Prompt:**

You are a code reviewer for a GitHub project. Your task is to evaluate whether a patch successfully resolves an issue described in a GitHub issue.

You will be given the issue description, the PoC (Proof of Concept) code that demonstrates the issue, and the PoC execution output before and after the patch.

Your goal is to determine if the patch resolves the issue. Please respond with "Yes" or "No" and provide a brief explanation of your reasoning.

You should not assume that there is other output that is not shown in the Poc Output. The Poc Output is the only output you should consider. You should not assume that a plot is generated by the Poc.

- "Yes" means the patch successfully fixed the issue.
- "No" means the patch did not successfully fix the issue, either because the issue still exists or because the patch introduced new issues.

### Raw Issue Description ###

{issue\_description}

### Poc code ###

Here is the PoC code that demonstrates the issue:

{poc\_code}

### Poc Output before the patch ###

{old\_execution\_output}

### Poc Output after the patch ###

{new\_execution\_output}

**\*\*Response Format:\*\***

Example 1:

<judgement> Yes </judgement>

<explanation> The patch successfully fixed the issue since ... </explanation>

Example 2:

<judgement> No </judgement>

<explanation> The patch did not successfully fix the issue since ...

</explanation>