

A SWE-BENCH-VERIFIED-S

SWE-Bench-verified-mini⁴ is a subset of SWE-Bench-Verified, containing 50 instead of 500 datapoints, requiring 5GB instead of 130GB of storage, while maintaining a similar distribution of performance, test pass rates, and task difficulty as the original dataset. Building on SWE-Bench-verified-mini, we augment it with 25 additional instances to better approximate the distribution and performance characteristics of the full dataset, resulting in our constructed benchmark, SWE-Bench-Verified-S.

Table 4: Instance Id in SWE-Bench-Verified-S

django__django-11790	django__django-11815
django__django-11848	django__django-11880
django__django-11885	django__django-11951
django__django-11964	django__django-11999
django__django-12039	django__django-12050
django__django-12143	django__django-12155
django__django-12193	django__django-12209
django__django-12262	django__django-12273
django__django-12276	django__django-12304
django__django-12308	django__django-12325
django__django-12406	django__django-12708
django__django-12713	django__django-12774
django__django-9296	sympy__sympy-13852
sympy__sympy-12481	sympy__sympy-17318
sympy__sympy-16766	sympy__sympy-15976
sympy__sympy-13974	sympy__sympy-13798
sympy__sympy-13647	sympy__sympy-20916
sympy__sympy-12489	sympy__sympy-24562
sympy__sympy-23824	sympy__sympy-23950
sympy__sympy-24661	sympy__sympy-16792
sympy__sympy-18189	sympy__sympy-12096
sympy__sympy-24539	sympy__sympy-13757
sympy__sympy-19495	sympy__sympy-18698
sympy__sympy-19346	sympy__sympy-17139
sympy__sympy-15809	sympy__sympy-22456
sphinx-doc__sphinx-10323	sphinx-doc__sphinx-10435
sphinx-doc__sphinx-10466	sphinx-doc__sphinx-10673
sphinx-doc__sphinx-11510	sphinx-doc__sphinx-7590
sphinx-doc__sphinx-7748	sphinx-doc__sphinx-7757
sphinx-doc__sphinx-7985	sphinx-doc__sphinx-8035
sphinx-doc__sphinx-8056	sphinx-doc__sphinx-8265
sphinx-doc__sphinx-8269	sphinx-doc__sphinx-8475
sphinx-doc__sphinx-8548	sphinx-doc__sphinx-8551
sphinx-doc__sphinx-8638	sphinx-doc__sphinx-8721
sphinx-doc__sphinx-9229	sphinx-doc__sphinx-9230
sphinx-doc__sphinx-9281	sphinx-doc__sphinx-9320
sphinx-doc__sphinx-9367	sphinx-doc__sphinx-9461
sphinx-doc__sphinx-9698	

B HYPERPARAMETERS OF MCTS

The Monte Carlo Tree Search (MCTS) algorithm used in this study employs several hyperparameters as following [15]:

⁴<https://huggingface.co/datasets/MariusHobbbahn/swe-bench-verified-mini>

Table 5: MCTS Hyperparameters

Hyperparameter	Description	Default
<i>Main Search Parameters</i>		
c_param	UCT exploration parameter	1.41
max_expansions	Max children per node	3
max_iterations	Max MCTS iterations	20
provide_feedback	Enable feedback	True
best_first	Use best-first strategy	True
value_function_temperature	Value function temperature	0.2
max_depth	Max tree depth	20
<i>UCT Score Calculation Parameters</i>		
exploration_weight	UCT exploration weight	1.0
depth_weight	Depth penalty weight	0.8
depth_bonus_factor	Depth bonus factor	200.0
high_value_threshold	High-value node threshold	55.0
low_value_threshold	Low-value node threshold	50.0
very_high_value_threshold	Very high-value threshold	75.0
high_value_leaf_bonus_constant	High-value leaf bonus	20.0
high_value_bad_children_bonus_constant	High-value bad children bonus	20.0
high_value_child_penalty_constant	High-value child penalty	5.0
<i>Action Model Parameters</i>		
action_model_temperature	Action model temperature	0.7
<i>Discriminator Parameters</i>		
number_of_agents	Number of Discriminator Agents	5
number_of_round	Number of debate rounds	3
discriminator_temperature	Discriminator temperature	1

C ABLATION SUPPLEMENT

In our ablation study, as presented in Table 6, we replaced the front-end components of our framework preceding the edit agent with LocAgent, which resulted in a Pass@1 drop to 37.4%. This comparison shows that our approach outperforms the current SOTA localization plugin LocAgent in end-to-end issue resolution, highlighting both the advantages and the effectiveness of our method.

Table 6: Ablation study results showing the contribution of different components.

Method	Pass@1	Δ
SWE-Debate	41.4%	-
w/o Multiple Chain Generation	31.4%	-10.0%
w/o Multi-Agent Debate	37.2%	-4.2%
w/o Edit plan	35.4%	-6.0%
w Locagent	37.4%	-4.0%

D RESULTS ON DIFFERENT MODELS

As presented in Table 7, we evaluate SWE-Debate on SWE-Bench-Verified using GPT-4o. Remarkably, our method maintains strong performance on GPT-4o and surpasses the current state-of-the-art for this model, underscoring its broad applicability and effectiveness. On SWE-bench Lite, As presented in Table 8, the same configuration(SWE-debate + GPT-4o) reaches a localization accuracy of 79.33%, which is a 5.97% absolute improvement over the GPT-4o baseline.

These results confirm the effectiveness of our approach across diverse models and datasets on both issue-resolution (with at least 5.15% relative improvement) and localization performance (with at least 5.06% relative improvement).

Table 7: Main effectiveness results on SWE-Bench-Verified.































Method	Model	Pass@1
SWE-Agent	 GPT-4o (2024-05-13)	23.0%
	 Claude-3.5 Sonnet	33.6%
	 DeepSeek-V3-0324	38.8%
SWE-Search	 DeepSeek-V3-0324	35.4%
Moatless Tools	 DeepSeek-V3-0324	34.6%
Agentless	 GPT-4o (2024-05-13)	36.2%
	 DeepSeek-V3-0324	36.6%
AutoCodeRover	 GPT-4o (2024-05-13)	38.4%
CodeAct	 GPT-4o (2024-05-13)	30.0%
SWESynInfer	 Claude-3.5 Sonnet	35.4%
	 GPT-4o (2024-05-13)	31.8%
	 Lingma SWE-GPT 72B	30.2%
OpenHands	 DeepSeek-V3-0324	38.8%
SWE-Debate	 DeepSeek-V3-0324	41.4%
	 GPT-4o (2024-05-13)	41.0%

Table 8: Localization Performance on SWE-Bench-lite.

Method	Model	Acc@1 (File)
Agentless	 GPT-4o (2024-05-13)	67.15
	 Claude-3.5 Sonnet	72.63
SWE-Agent	 GPT-4o (2024-05-13)	57.30
	 Claude-3.5 Sonnet	77.37
	 DeepSeek-V3-0324	67.00
SWE-Search	 GPT-4o (2024-05-13)	73.36
	 Claude-3.5 Sonnet	72.63
CodeActAgent	 GPT-4o (2024-05-13)	60.95
	 Claude-3.5 Sonnet	76.28
LocAgent	 Qwen2.5-7B (FT)	70.80
	 Qwen2.5-32B (FT)	75.91
	 Claude-3.5 Sonnet	77.74
KGCompass	 Claude-3.5 Sonnet	76.67
SWE-Debate	 DeepSeek-V3-0324	81.67 (+3.93)
	 GPT-4o (2024-05-13)	79.33 (+5.97)

E COST REPORT

Appendix Table 9 summarizes the cost analysis of three key hyperparameters: 1. **Number of chains**: Increasing the number of generated chains from 10 to 25 steadily raises the average tokens per issue and overall wall time. 2. **Chain depth**: Greater chain depth likewise leads to higher token consumption and longer run-time. 3. **Debate agents**: Expanding the number of debate agents from 3 to 7 has only a minor effect, with tokens and time remaining nearly unchanged. Overall, larger numbers of chains and deeper chains incur higher computational costs, whereas the number of debate agents has little impact on cost.

Parameter Tuning Recommendations: Set the initial number of entities according to issue length, with a minimum of three to reduce random path deviation. For long issues, slightly increase both initial and expansion entities but keep the total below ten to avoid introducing irrelevant entities. Apply the same principle to the second-round expansion parameter W . Limit the overall number of chains to at most 40. The number of debate agents can be raised to about seven for complex issues, but exceeding this may overwhelm the discriminator and hinder consensus; a range of three to seven balances diversity and integration. Debate rounds are fixed at three in our framework, which already yields satisfactory results. Chain depth of five, as shown in our ablation study, offers a good trade-off between cost and resolution rate.

Table 9: Cost report on number of chains, chain depth, and debate agents.

Number of Chains				
	10	15	20	25
Per-issue tokens	285.4K	409.6K	518.2K	638.1K
Wall time (min)	18.7	23.9	28.5	33.4
Tool calls	9.65	9.55	9.64	9.53
Chain Depth				
	3	5	7	
Per-issue tokens	339.7K	518.2K	699.5K	
Wall time (min)	21.6	28.5	35.6	
Tool calls	9.33	9.64	10.17	
Debate Agents				
	3	5	7	
Per-issue tokens	515.7K	518.2K	520.3K	
Wall time (min)	28.3	28.5	28.6	
Tool calls	9.58	9.64	9.59	

F PROMPT TEMPLATES

In the following section, we enumerate all the prompts used throughout our entire workflow, from the initial entity extraction to the final plan generation.

Prompt 1: INITIAL ENTITY EXTRACTION PROMPT

You are a code analysis expert. Given an issue description, your task is to identify the most relevant code entities (classes, methods, functions, variables) that are likely involved in the issue.

Important: Only extract entities that are explicitly mentioned or strongly implied by the issue description. Do not invent names that are not referenced in the text.

****Issue Description:****
{issue_description}

****Instructions:****

1. Analyze the issue description to identify:
 - ****Classes****: e.g., ``UserAuthenticator``, ``PaymentProcessor``
 - ****Methods/Functions****: e.g., ``validate_credentials()``, ``process_payment()``
 - ****Variables/Parameters****: e.g., ``user_id``, ``transaction_amount``
 - ****Error Types/Exceptions****: e.g., ``RateLimitExceededError``, ``DatabaseConnectionError``
2. ****Focus on direct mentions****: Only include entities that are clearly referenced in the issue.
3. ****Avoid redundancy****: If multiple terms refer to the same entity (e.g., "the payment handler" and ``PaymentProcessor``), pick the most precise name.
4. ****Prioritize key components****: Rank entities by how central they are to the issue.
5. ****Return only names****: Do not include paths, modules, or extra descriptions.
6. ****Limit to {max_entities} entities****: Select only the {max_entities} most relevant and important entities for this issue.

****Output Format:****

Return a JSON list of exactly {max_entities} entity names in order of relevance (most relevant first):
["entity_name1", "entity_name2", "entity_name3", ...]

****Examples:****

1. ****Issue Description:****
Query syntax error with condition and distinct combination
Description:
A Count annotation containing both a Case condition and a distinct=True param produces a query error on Django 2.2 (whatever the db backend). A space is missing at least (... COUNT(DISTINCTCASE WHEN ...)).

****Output (if max_entities=3):****
["Count", "DISTINCTCASE", "distinct"]

2. ****Issue Description:****
"After upgrading to v2.0, the ``UserSession`` class sometimes fails to store session data in Redis, causing login loops."

****Output (if max_entities=2):****
["UserSession", "Redis"]

3. ****Issue Description:****

"The ``calculate_discount()`` function applies incorrect discounts for bulk orders when ``customer_type = 'wholesale'``."

****Output (if max_entities=3):****
["calculate_discount", "customer_type", "wholesale"]

Note: Return only the simple names like ``__iter__``, ``page_range``, ``MyClass``, ``my_function``, etc. Do not include file paths or full qualified names.

Return exactly {max_entities} entities, prioritizing the most important ones if there are more candidates.

Prompt 2: CODE SNIPPET ENTITY EXTRACTION PROMPT

Based on the following code snippets and problem statement, identify the 4 most relevant entities (files, classes, or functions) that are likely involved in solving this issue.

****Problem Statement:****
{problem_statement}

****Code Snippets:****
{code_snippets}

****Instructions:****

1. Analyze the problem statement to understand what needs to be fixed/implemented
2. Review the code snippets to identify relevant entities
3. ****PRIORITIZE DIVERSITY****: Select entities from different files whenever possible to ensure comprehensive coverage
4. ****BALANCE RELEVANCE AND DIVERSITY****: Choose entities that are both highly relevant to the issue AND come from different modules/files
5. Avoid selecting multiple entities from the same file unless absolutely necessary
6. Select exactly 4 entities that collectively provide the best coverage for solving the issue
7. For each entity, provide the exact entity ID in the format expected by the codebase

****Selection Strategy:****

- First priority: High relevance to the problem + Different file locations
- Second priority: High relevance to the problem (even if some files overlap)
- Ensure the selected entities represent different aspects or layers of the solution

****Output Format:****

Return a JSON list containing exactly 4 entities, each with the following format:

```
```json
[
 {
 "entity_id": "file_path:QualifiedName or just file_path",
 }
]
```

```

1741 "entity_type": "file|class|function",
1742 "relevance_reason": "Brief explanation of
1743 why this entity is relevant to the
1744 issue",
1745 "diversity_value": "How this entity adds
1746 diversity (e.g., 'different file', '
1747 different layer', 'different
1748 functionality')")
1749 }
1750 ...
1751
1752 **Example:**
1753 ```json
1754 [
1755 {
1756 "entity_id": "src/models.py:UserModel",
1757 "entity_type": "class",
1758 "relevance_reason": "Contains user-related
1759 functionality mentioned in the issue",
1760 "diversity_value": "Model layer from
1761 different file"
1762 },
1763 {
1764 "entity_id": "src/views.py:UserView",
1765 "entity_type": "class",
1766 "relevance_reason": "Handles user
1767 interface logic that may need
1768 modification",
1769 "diversity_value": "View layer from
1770 different file"
1771 },
1772 {
1773 "entity_id": "src/utils/validators.py:
1774 validate_user_input",
1775 "entity_type": "function",
1776 "relevance_reason": "Input validation
1777 logic relevant to the user issue",
1778 "diversity_value": "Utility function from
1779 different module"
1780 },
1781 {
1782 "entity_id": "src/config.py",
1783 "entity_type": "file",
1784 "relevance_reason": "Configuration
1785 settings that may affect user
1786 behavior",
1787 "diversity_value": "Configuration file
1788 from different location"
1789 }
1790]
1791 ...
1792
1793 **Remember:** Maximize both relevance to the issue
1794 AND diversity across different files/modules
1795 to ensure comprehensive localization chain
1796 generation.
1797

```

### Prompt 3: NEIGHBOR PREFILTERING PROMPT

You are a code analysis expert helping to select the most relevant and diverse neighbors for exploring a dependency graph to solve a specific issue.

```

1799
1800 **Issue Description:**
1801 {issue_description}
1802
1803 **Current Entity:** {current_entity}
1804 **Current Entity Type:** {current_entity_type}
1805 **Traversal Depth:** {depth}
1806
1807 **Available Neighbor Entities ({{total_count}} total
1808):**
1809 {neighbor_list}
1810
1811 **Your Task:**
1812 From the {total_count} available neighbors, select
1813 up to {max_selection} most relevant and
1814 diverse entities that would be most promising
1815 to explore next.
1816
1817 **Selection Criteria:**
1818 1. **Relevance to Issue:** How likely is this
1819 neighbor to contain code related to solving
1820 the issue?
1821 2. **Diversity:** Avoid selecting too many
1822 entities from the same file or with similar
1823 names
1824 3. **Strategic Value:** Prioritize entities that
1825 could lead to discovering the root cause or
1826 solution
1827 4. **Entity Type Variety:** Balance between files,
1828 classes, and functions when possible
1829
1830 **Instructions:**
1831 1. Analyze each neighbor entity ID to understand
1832 what it likely represents
1833 2. Consider file paths, entity names, and types to
1834 assess relevance
1835 3. Ensure diversity by avoiding redundant
1836 selections from the same file/module
1837 4. Select entities that complement each other in
1838 exploring different aspects of the issue
1839 5. Return exactly the entity IDs that should be
1840 explored further (up to {max_selection})
1841
1842 **Output Format:**
1843 Return a JSON object with your selection:
1844 ```json
1845 {
1846 "selected_neighbors": [
1847 "neighbor_entity_id_1",
1848 "neighbor_entity_id_2",
1849 ...
1850],
1851 "selection_reasoning": "Brief explanation of
1852 your selection strategy and why these
1853 neighbors were chosen",
1854 "diversity_considerations": "How you ensured
1855 diversity in your selection"
1856 }
1857 ```
1858
1859 Focus on strategic exploration that maximizes the
1860 chance of finding issue-relevant code while
1861 maintaining diversity.

```

**Prompt 4: NODE SELECTION PROMPT**

You are a code analysis expert helping to navigate a dependency graph to solve a specific issue. Given the current context and available neighboring nodes, determine which node would be most promising to explore next.

**\*\*Issue Description:\*\***  
{issue\_description}

**\*\*Current Entity:\*\*** {current\_entity}  
**\*\*Current Entity Type:\*\*** {current\_entity\_type}  
**\*\*Traversal Depth:\*\*** {depth}

**\*\*Available Neighbor Nodes:\*\***  
{neighbor\_info}

**\*\*Context:\*\***

- We are performing graph traversal to find code locations relevant to solving this issue
- Each neighbor represents a related code entity (file, class, or function)
- We need to select the most promising node to continue exploration

**\*\*Instructions:\*\***

1. Analyze how each neighbor might relate to solving the issue
2. Consider the traversal depth and whether we should continue or stop
3. Evaluate which neighbor is most likely to contain relevant code for the solution
4. Return your decision on whether to continue exploration and which neighbor to select

**\*\*Output Format:\*\***  
Return a JSON object with your decision:  
```json  
{
 "should_continue": true/false,
 "selected_neighbor": "neighbor_entity_id or null",
 "reasoning": "Explanation of your decision",
 "confidence": 0-100
}

...

If should_continue is false, set selected_neighbor to null.
If should_continue is true, select the most promising neighbor_entity_id.

Prompt 5: CHAIN VOTING PROMPT

You are an expert software engineer tasked with identifying the optimal modification location for solving a specific software issue.

****Issue Description:****
{issue_description}

****Available Localization Chains:****
{chains_info}

****Your Task:****

Analyze each localization chain as a potential modification target and vote for the ONE chain where making changes would most likely resolve the issue described above.

****Evaluation Criteria:****

1. ****Problem Location Accuracy****: Does this chain contain the actual location where the bug/issue manifests?
2. ****Modification Impact****: How directly would changes to this code path affect the described problem?
3. ****Code Modifiability****: Is the code in this chain well-structured and safe to modify?
4. ****Solution Completeness****: Would fixing this chain likely resolve the entire issue, not just symptoms?
5. ****Risk Assessment****: What are the risks of modifying this particular code path?

****Key Questions to Consider:****

- Which chain contains the root cause rather than just related functionality?
- Where would a developer most likely need to make changes to fix this specific issue?
- Which code path, when modified, would have the most direct impact on resolving the problem?
- Which chain provides the clearest entry point for implementing a fix?

****Instructions:****

1. For each chain, analyze whether modifying its code would directly address the issue
2. Consider the logical flow: which chain is most likely to contain the problematic code?
3. Evaluate implementation feasibility: which chain would be safest and most effective to modify?
4. Vote for exactly ONE chain that represents the best modification target
5. Focus on where to make changes, not just what's related to the issue

****Output Format:****

Return a JSON object with your vote:
```json

```
{
 "voted_chain_id": "chain_X",
 "confidence": 85,
 "reasoning": "Detailed explanation of why this chain is the best modification target for solving the issue",
 "modification_strategy": "Brief description of what type of changes would be needed in this chain",
 "chain_analysis": {
 "chain_1": "Assessment of this chain as a modification target",
 "chain_2": "Assessment of this chain as a modification target",
 ...
 }
}
```

**\*\*Example:\*\***

```

'''json
{{
 "voted_chain_id": "chain_2",
 "confidence": 88,
 "reasoning": "Chain 2 contains the pagination
iterator __iter__ method which is where
the infinite loop issue described in the
problem statement actually occurs.
Modifying the logic in this method to
properly handle the iteration termination
would directly solve the reported bug.",
 "modification_strategy": "Add proper boundary
checking and iteration termination logic
in the __iter__ method",
 "chain_analysis": {{
 "chain_1": "Contains utility functions but
modifications here would not address
the core iteration logic issue",
 "chain_2": "Contains the actual iterator
implementation where the bug
manifests - ideal modification target
",
 "chain_3": "Related display logic but
changes here would not fix the
underlying iteration problem"
 }}
}}

```

#### Prompt 6: ROUND 1 MODIFICATION LOCATION PROMPT

You are an expert software engineer tasked with identifying specific code locations that need to be modified to solve a given issue.

**\*\*Issue Description:\*\***  
{issue\_description}

**\*\*Selected Localization Chain:\*\***  
{chain\_info}

**\*\*Your Task:\*\***  
Analyze the localization chain and identify the specific locations within this chain that need to be modified to solve the issue. Focus on pinpointing the exact functions, methods, or code blocks that require changes.

**\*\*CRITICAL REQUIREMENT FOR INSTRUCTIONS:\*\***

- Each suggested\_approach must be a DETAILED, STEP-BY-STEP instruction
- Include specific code examples, parameter names, and implementation details
- Specify exact lines to modify, functions to add, and variables to change
- Provide concrete implementation guidance that a developer can directly follow
- Include error handling, edge cases, and validation requirements
- Mention specific imports, dependencies, or setup needed

**\*\*Instructions:\*\***  
1. Examine each entity in the localization chain and its code

2. Identify which specific parts of the code are causing the issue or need enhancement
3. Determine the precise locations where modifications should be made
4. Explain why each location needs modification and what type of change is required
5. Prioritize the modifications by importance (most critical first)
6. For each modification, provide DETAILED implementation instructions with specific code examples

**\*\*Output Format:\*\***

Return a JSON object with your analysis:

```

'''json
{{
 "modification_locations": [
 {{
 "entity_id": "specific_entity_id",
 "location_description": "Specific
function/method/lines that need
modification",
 "modification_type": "fix_bug|
add_feature|refactor|optimize",
 "priority": "high|medium|low",
 "reasoning": "Detailed explanation of
why this location needs
modification",
 "suggested_approach": "DETAILED step-
by-step implementation
instructions with specific code
examples, parameter names, exact
function signatures, error
handling, and complete
implementation guidance that can
be directly executed by a
developer"
 }}
],
 "overall_strategy": "Overall approach to
solving the issue using these
modifications",
 "confidence": 85
}}
'''

```

**\*\*Example of DETAILED suggested\_approach:\*\***  
Instead of: "Add proper termination condition"  
Provide: "Modify the \_\_iter\_\_ method in the Paginator class by adding a counter variable 'current\_page = 1' at the beginning. Then add a while loop condition 'while current\_page <= self.num\_pages:' to replace the infinite loop. Inside the loop, yield 'self.page(current\_page)' and increment 'current\_page += 1'. Add try-catch block to handle PageNotAnInteger and EmptyPage exceptions by catching them and breaking the loop. Import the exceptions 'from django.core.paginator import PageNotAnInteger, EmptyPage' at the top of the file."



**Prompt 7: ROUND 2 COMPREHENSIVE MODIFICATION PROMPT**

```

"""
You are an expert software engineer participating
in a collaborative code review process to
determine the best approach for solving a
software issue.

Issue Description:
{issue_description}

Selected Localization Chain:
{chain_info}

Your Initial Analysis:
{your_initial_analysis}

Other Agents' Analyses:
{other_agents_analyses}

Your Task:
Based on the issue, the localization chain, your
initial analysis, and insights from other
agents, provide a refined and comprehensive
analysis of where and how the code should be
modified.

CRITICAL REQUIREMENT FOR REFINED INSTRUCTIONS:
- Each suggested_approach must be EXTREMELY
 DETAILED with complete implementation
 guidance
- Include specific code snippets, exact function
 signatures, and parameter details
- Provide line-by-line modification instructions
 where applicable
- Specify all necessary imports, dependencies, and
 setup requirements
- Include comprehensive error handling and edge
 case considerations
- Mention testing requirements and validation
 steps
- Provide specific examples of input/output or
 before/after code states

Instructions:
1. Review your initial analysis and the analyses
 from other agents
2. Identify common patterns and disagreements in
 the proposed modifications
3. Synthesize the best insights from all analyses
4. Refine your modification recommendations based
 on collective wisdom
5. Provide a more comprehensive and well-reasoned
 final recommendation
6. Ensure each suggested_approach contains
 exhaustive implementation details

Output Format:
Return a JSON object with your refined analysis:
```json
{
  "refined_modification_locations": [
    {
      "entity_id": "specific_entity_id",

```

```

      "location_description": "Specific
        function/method/lines that need
        modification",
      "modification_type": "fix_bug|
        add_feature|refactor|optimize",
      "priority": "high|medium|low",
      "reasoning": "Enhanced reasoning
        incorporating insights from other
        agents",
      "suggested_approach": "EXHAUSTIVE step
        -by-step implementation guide
        including: exact code snippets to
        add/modify/remove, complete
        function signatures, all required
        imports, parameter validation,
        error handling, edge cases,
        testing considerations, and
        specific examples of before/after
        states",
      "supporting_evidence": "References to
        other agents' insights that
        support this decision"
    }
  ],
  "overall_strategy": "Comprehensive strategy
    refined through collaborative analysis",
  "confidence": 90,
  "key_insights_learned": "What you learned from
    other agents' analyses",
  "potential_risks": "Potential risks or
    challenges identified through
    collaborative review"
}

```

Remember: Each suggested_approach should be so detailed that a developer can implement it without additional research or clarification.

Prompt 8: FINAL DISCRIMINATOR PROMPT

```

You are the lead software architect making the
final decision on a code modification plan.
Multiple expert engineers have provided their
analyses for solving a software issue.

**Issue Description:**
{issue_description}

**Selected Localization Chain:**
{chain_info}

**All Agents' Final Analyses:**
{all_agents_analyses}

**Your Task:**
Synthesize all the expert analyses and create a
definitive, actionable modification plan that
will solve the issue effectively and safely.

**CRITICAL REQUIREMENTS FOR INSTRUCTIONS:**
- Every instruction MUST be a concrete
  modification action (Add, Remove, Modify,
  Replace, Insert, etc.)

```

```

- NO verification, checking, or validation
  instructions (avoid "Verify", "Ensure", "
  Check", "Maintain", etc.)
- Each instruction should specify exactly WHAT to
  change and HOW to change it
- Focus on direct code modifications that
  implement the solution

**Instructions:**
1. Analyze all the expert recommendations and
  identify the most reliable and consistent
  suggestions
2. Resolve any conflicts between different expert
  opinions using technical merit
3. Create a prioritized, step-by-step modification
  plan with ONLY concrete modification actions
4. Ensure the plan is practical, safe, and
  addresses the root cause of the issue
5. Include specific instructions for each
  modification
6. The output context should be as detailed as
  possible
7. Use action verbs like: "Add", "Modify", "
  Replace", "Insert", "Update", "Change", "
  Remove", "Implement"

**Output Format:**
Return a comprehensive modification plan:
```json
{
 "final_plan": {
 "summary": "High-level summary of the
 modification approach",
 "modifications": [
 {
 "step": 1,
 "instruction": "Concrete
 modification instruction
 using action verbs (Add/
 Modify/Replace/etc.)",
 "context": "File path and specific
 location (e.g., function,
 method, line range)",
 "type": "fix_bug|add_feature|
 refactor|optimize",
 "priority": "critical|high|medium|
 low",
 "rationale": "Why this
 modification is necessary and
 how it contributes to
 solving the issue",
 "implementation_notes": "Specific
 technical details for
 implementation"
 }
],
 "execution_order": "The recommended order
 for implementing these modifications",
 "testing_recommendations": "Suggested
 testing approach for validating the
 modifications",
 "risk_assessment": "Potential risks and
 mitigation strategies"
 },
 "confidence": 95,

```

```

"expert_consensus": "Summary of areas where
 experts agreed",
"resolved_conflicts": "How conflicting expert
 opinions were resolved"
}}
```

**Examples of GOOD instructions:**
- "Add maxlength attribute to the widget
  configuration"
- "Modify the widget_attrs method to include
  max_length parameter"
- "Replace the current field initialization with
  max_length support"
- "Insert validation logic for maximum length"

**Examples of BAD instructions (DO NOT USE):**
- "Verify the max_length setting"
- "Ensure proper validation"
- "Check if the field is configured correctly"
- "Maintain the existing functionality"

Focus on creating a plan that can be directly
executed by a modification agent with clear,
actionable steps.

```