KAGGLE BENCHMARKS PLATFORM

Bug Report & Platform Feedback

WWTP Autonomous Management Benchmark

Mehmet ISIK
Data Scientist & Kaggle Grandmaster
February 2026

1. Executive Summary

This report documents platform-level bugs and constraints discovered during the development of a Wastewater Treatment Plant (WWTP) autonomous management benchmark on the Kaggle Benchmarks (kbench) platform. The benchmark tests whether LLMs can autonomously manage critical industrial infrastructure through multi-turn role-playing conversations with image analysis, sensor data evaluation, and independent judge scoring.

During development and testing across 22 models, I identified 9 distinct platform issues. Some caused silent registration failures, others produced misleading error messages that wasted significant debugging time. One particularly impactful finding involved a model safety filter silently rejecting legitimate industrial terminology, which took approximately 5 hours and 7 dedicated debug notebooks to isolate. All findings include reproducible evidence, root cause analysis, and constructive suggestions for improvement.

#	Bug	Severity	Status	Workaround
1	Silent registration failure (assertion strings)	**Critical**	Resolved	Short labels (<35 chars)
2	Task name permanent cache	**Critical**	Resolved	Use fresh task names
3	Code size limit (~15K chars)	**High**	Resolved	External JSON templates
4	Chat serialization truncation	**Medium**	Open	Limit to 4 chats
5	Concurrent API rate limit masking	**Critical**	Open	Run same-provider sequentially
6	Image format validation inconsistency	**Medium**	Resolved	Use real PNG format
7	Stuck cancelling sessions	**High**	Open	None (requires backend)
8	Dataset version propagation delay	**Medium**	Open	Re-attach dataset
9	Safety filter on industrial terminology	**Critical**	Resolved	Use alternative terminology

2. Bug Details

Bug #1: Silent Registration Failure (Assertion String Length)

Severity: Critical

Impact: Task executes successfully with score 1.0 but silently fails to register on Benchmarks page. No error message displayed.

Description: When assertion expectation strings exceed approximately 35 characters, the Kaggle backend's post-processing step fails silently during task registration. The task shows "Failed" status for about one second, then disappears. The notebook output shows a perfect execution with all assertions passed.

Root Cause: The backend has an undocumented maximum length for the expectation parameter in assertions.assert_true(). Long strings (e.g., full judge reasoning text >200 chars) cause registration to fail with no feedback.

Solution: Use short labels like "J1: Sensor comparison" instead of full descriptions. Keep all expectation strings under 35 characters.

***Suggestion:** Display an error message such as "Assertion expectation string exceeds maximum length (35 characters)" instead of failing silently.*

Bug #2: Failed Task Name Gets Permanently Cached

Severity: Critical

Impact: A task that fails once can never succeed again under the same name, even with completely rewritten code.

Description: My Scenario 9 task failed on the first run due to a Python KeyError (my bug in the code). After fixing the code, I ran it 7 more times. Every single run executed successfully (scores ranged 0.40 to 1.0), but Kaggle kept marking it as FAILED. I tried reducing prompt size, cutting LLM calls from 7 to 4, adding word limits, truncating data — nothing worked.

Root Cause: The backend caches a failed task name and auto-rejects all subsequent versions saved under that name, even when the code is completely rewritten. No error message indicates this.

Solution: Renamed the task from s9_wwtp_flood_emergency to s9_wwtp_flood_cascade. Instant success on the first try with identical code.

Suggestion: *Either allow task names to be reused after code changes, or display an error message like "Task name locked due to previous failure — use a new name."*

Bug #3: Code Size Limit (~15,000 Characters)

Severity: **High**

Impact: Registration fails silently when task code exceeds approximately 15,000 characters.

Description: When converting multi-LLM scenarios to single-LLM format using full original prompts, registration failed silently. The code was around 18,000 characters. After compacting prompts while keeping the same functionality, it dropped to around 7,000 characters and registered successfully. The threshold appears to be between 15,000 and 18,000 characters.

Solution: Moved long prompt templates to an external JSON file loaded from the dataset at runtime, keeping the task code compact.

Suggestion: *Display an error message like "Task definition exceeds maximum size" with the actual limit.*

Bug #4: Chat Serialization Truncation

Severity: **Medium**

Impact: Tasks with 5-7 kbench.chats.new() calls show only 3-4 conversations in the result JSON output.

Description: The platform appears to truncate conversation logs when more than 3-4 separate chat sessions are used within a single task. Execution is not blocked, but the logged output is incomplete, making debugging significantly harder. This was consistently reproducible across multiple scenarios and models.

Suggestion: *Either increase the serialization limit or document it clearly so developers can plan their chat structure accordingly.*

Bug #5: Concurrent API Rate Limit Masking

Severity: Critical

Impact: Models appear to fail with misleading errors when multiple models from the same API provider run concurrently.

Description: When running a benchmark task, the platform launches all selected models simultaneously. When multiple Anthropic models (e.g., Claude Sonnet 4 + Claude Sonnet 4.5) run at the same time, they share the same API quota. The faster model completes successfully, but slower models hit rate limits mid-execution and receive empty API responses.

The critical problem is how these failures are reported:

Error Displayed	Actual Cause	User Perception
TypeError: 'NoneType' object is not subscriptable	API rate limit — empty response	"SDK bug or my code is broken"
FileNotFoundError: No such file or directory	Rate limit during file I/O setup	"Dataset path is wrong"

Evidence: Claude Sonnet 4.5 failed with NoneType error when run alongside Claude Sonnet 4 and Gemini Flash. After removing other Anthropic models and running Sonnet 4.5 alone, it scored 1.00 on the first attempt. Claude Sonnet 4 similarly succeeded (0.85) when run in isolation with time.sleep(3) between turns.

Debugging cost: This issue consumed approximately 4 hours of debugging time because the error messages pointed to code bugs rather than API rate limiting.

***Suggestion:** It might be helpful to surface rate limit errors explicitly with a message like "API rate limit exceeded for provider [Anthropic] — try running fewer models from this provider simultaneously, or retry after a cooldown." Staggering concurrent runs from the same API provider could also help prevent this issue.*

Bug #6: Image Format Validation Inconsistency

Severity: Medium

Impact: Same image files work on Google models but fail on Anthropic models with format mismatch errors.

Description: Image files with .png extension but containing JPEG data (a common result of renaming files without format conversion) are handled differently by model providers. Google's Gemini models process them without error, while Anthropic's Claude models reject them with "Image does not match the provided media type image/png."

Root Cause: The kbench SDK reads the file extension to determine media type but does not validate that the actual file content matches. Google's API is tolerant of this mismatch; Anthropic's API is strict.

Solution: Converted all images to genuine PNG format using proper image processing (PIL/Pillow). After conversion, Anthropic models processed images successfully.

Suggestion: *Adding a validation step in content_types.images.from_path() that checks file header bytes against the declared media type could help warn developers of mismatches before sending to the API.*

Bug #7: Stuck Cancelling Sessions

Severity: **High**

Impact: Blocks all new task submissions due to concurrent run limits.

Description: 34 benchmark sessions became stuck in "Cancelling..." state for over 18 hours. Clicking on these sessions shows "Notebook Deleted" but they cannot be terminated. The platform enforces a limit of 20 concurrent runs, and these stuck sessions count against that limit, effectively blocking all new task submissions with the error: "Task versions and model versions would create over 20 concurrent runs."

Resolution: No user-side workaround exists. Requires Kaggle backend intervention to clear stuck sessions.

Suggestion: *A timeout mechanism for cancelling sessions (e.g., force-terminate after 30 minutes) and a user-accessible "Force Cancel" button for stuck sessions could be very helpful.*

Bug #8: Dataset Version Propagation Delay

Severity: **Medium**

Impact: Some model runners access old dataset versions while others access the current version.

Description: After updating a dataset (e.g., replacing JPEG-as-PNG images with genuine PNGs), some models still accessed the old version during their runs. This was confirmed by observing that Google models (which ran successfully with old JPEG-as-PNG files) continued to work, while newly uploaded genuine PNG files were not found by other runners. Running os.listdir() in the notebook confirmed files were present, but runner containers appeared to have cached the old dataset version.

Workaround: Remove the dataset from notebook inputs and re-attach it, then Save Task again. Allow extra time between dataset update and task execution.

Suggestion: *Ensuring all runner containers receive the same dataset version for a given task execution, and displaying the dataset version each runner is using in the output logs, could help prevent confusion.*

Bug #9: Safety Filter Silently Rejects Industrial Terminology

Severity: Critical

Impact: Claude Opus models silently reject prompts containing legitimate industrial infrastructure terminology, returning empty responses that crash the SDK.

Description: Claude Opus 4.5 and Opus 4.1 consistently failed on the WWTP benchmark with TypeError: NoneType at the very first prompt. The same code worked perfectly on Claude Sonnet 4, Sonnet 4.5, all Gemini models, and all Qwen models. Initially this appeared to be a rate limit issue (Bug #5), but after extensive systematic testing across 7 dedicated debug notebooks, I discovered that the root cause was entirely different: Opus's safety filter was silently rejecting the prompt content.

Investigation process: The debugging followed a systematic elimination approach over 7 separate test notebooks:

1) opus_test: Confirmed Opus works with simple prompts (scored 1.00).

2) opus_debug_v1: Ran 5 progressively complex prompts. Tests 1-3 passed, Test 4 (S1-like prompt) failed.

3) opus_debug_v2: Placed the S1 prompt as the first and only call. Still failed — eliminated chat count as the cause.

4) opus_debug_v3: Tested 3 variants. Short version passed, MASTER AGENT keyword passed, full original failed.

5) opus_debug_v4: Tested location, DRONE AGENT, and length separately. "MASTER AGENT at Adana WWTP" failed.

6) opus_safety_test: Neutral version passed, industrial version failed — confirmed content filtering.

7) opus_keyword_test: Isolated individual keywords. "Gas Storage Area" alone triggered the filter.

Root Cause: The phrase "Gas Storage Area" triggers Opus's safety filter, causing the API to return an empty response instead of a content-filtered message. The kbench SDK then crashes with TypeError when trying to parse the empty response. Notably, Claude Sonnet models process the exact same prompt without any issue.

Keyword isolation evidence:

Prompt Content	Opus 4.5	Sonnet 4.5
"You are MASTER AGENT. Order inspection."	✓ 1.00	✓ 1.00
"...Order DRONE AGENT to inspect..."	✓ Pass	✓ Pass

"...check Gas Storage Area, Valve 3"	✗ Fail	✓ Pass
"...check Biogas Unit, Valve 3"	✓ 1.00	✓ 1.00

Solution: Replaced "Gas Storage Area" with "Biogas Unit" — which is actually the more precise technical term in wastewater treatment. Opus immediately scored 1.00 with this change.

Why Opus but not Sonnet? Opus is Anthropic's most capable and most expensive model. More powerful models typically have stricter safety filters. The combination of "Gas Storage" with industrial inspection terminology likely crosses Opus's safety threshold for critical infrastructure scenarios, while Sonnet's lower filter threshold tolerates it.

Why this didn't occur in multi-LLM: In the multi-LLM architecture, Opus served only as the MASTER AGENT, issuing high-level commands. The detailed industrial terminology ("Gas Storage Area", "leaks", "corrosion") was sent to Gemini (DRONE role), which has no such filter. In single-LLM mode, the same model plays every role and encounters all terminology.

Debugging cost: Approximately 5 hours and 7 dedicated debug notebooks. The silent failure with no error message made this exceptionally difficult to diagnose.

Suggestions: *Two improvements could significantly help developers in similar situations:*

1) When a safety filter blocks a response, it would be very helpful to return an explicit message (e.g., "Content filtered due to safety policy") instead of an empty response that causes SDK crashes.

2) It could be worth considering that terms like "gas storage", "leak detection", and "valve inspection" are everyday vocabulary in water treatment, energy, and industrial operations — not indicators of harmful intent.

3. Testing Environment & Results

The WWTP benchmark was tested across 22 models available on the Kaggle kbench platform. The benchmark (Scenario 1: Routine Morning Inspection) involves 5 conversation turns, 1 image analysis, 3 hard-coded assertions, and 10 LLM judge criteria.

S1 Results After All Fixes

Model	Score	Status	Notes
Gemini 2.5 Flash	**1.00**	Pass	SCADA role in multi-LLM
Gemini 2.5 Pro	**1.00**	Pass	DRONE role in multi-LLM
Gemini 2.0 Flash	0.85	Pass	
Gemini 2.0 Flash Lite	**1.00**	Pass	
Gemini 3 Flash Preview	**1.00**	Pass	
Gemini 3 Pro Preview	**1.00**	Pass	
Gemma 3 4B	**1.00**	Pass	
Gemma 3 12B	0.77	Pass	
Gemma 3 27B	0.85	Pass	
Qwen 3 235B A22B Instruct	**1.00**	Pass	MEKANIK role in multi-LLM
Qwen 3 Coder 480B	**1.00**	Pass	
Qwen 3 Next 80B Instruct	**1.00**	Pass	
Qwen 3 Next 80B Thinking	**1.00**	Pass	
DeepSeek V3.1	**1.00**	Pass	
DeepSeek V3.2	**1.00**	Pass	
Claude Sonnet 4	0.85	Pass	Fixed via Bug #5 workaround
Claude Sonnet 4.5	**1.00**	Pass	Fixed via Bug #5 workaround
Claude Opus 4.5	**1.00**	Pass	Fixed via Bug #9 workaround
Claude Opus 4.1	TBD	Pending	Bug #9 fix to be applied
DeepSeek R1	N/A	Fail	No vision capability
Gemma 3 1B	N/A	Fail	Too small for complex task

4. Key Insights

4.1 Error Message Clarity

The single most impactful improvement the Kaggle Benchmarks team could make is improving error message clarity. Across all 9 bugs documented in this report, the common thread is that failures are either silent or produce misleading error messages.

A concrete example: Bug #9 (Safety Filter) consumed approximately 5 hours of debugging time across 7 dedicated notebooks. The error "TypeError: NoneType object is not subscriptable" led me through a systematic investigation of rate limits, chat counts, prompt length, keyword combinations, and timing. The actual cause — a safety filter rejecting "Gas Storage Area" — was only discovered through keyword-by-keyword elimination testing. If the API had returned "Content filtered" instead of an empty response, the issue would have been resolved in minutes.

4.2 The Industrial AI Paradox

Bug #9 reveals a critical paradox for industrial AI adoption. The most capable LLMs — the ones best equipped to handle complex industrial reasoning, multi-step decision making, and safety-critical assessments — impose the most restrictive safety filters on legitimate industrial terminology. Keywords that are routine vocabulary for any wastewater treatment plant operator ("gas storage", "leak detection", "valve inspection") trigger silent rejections in top-tier models.

This creates an inverse capability gap: the models most suitable for industrial autonomous management are the ones most likely to refuse the task. For heavy industry applications where AI-driven management is the goal, overly aggressive safety filtering — especially without clear error messages — represents a significant barrier to real-world deployment.

4.3 Prompt Engineering as Industrial Enabler

This experience powerfully reinforced the importance of prompt engineering — not just as an optimization technique for better outputs, but as a critical enabler for industrial AI viability. The difference between a model refusing to participate in an industrial safety scenario and successfully scoring 1.00 came down to a single terminology choice: "Gas Storage Area" vs "Biogas Unit".

Ironically, the safety filter pushed toward more precise industrial terminology. "Biogas Unit" is the technically correct term in wastewater treatment, while "Gas Storage Area" was the less precise colloquial term. In this case, better prompt engineering simultaneously solved the safety filter issue and improved technical accuracy.

5. Conclusion

The Kaggle Benchmarks platform is an impressive and genuinely powerful system. The ability to define complex multi-turn, multi-modal evaluation tasks and run them across 22 models with a single click enables research that was not practical before. The SDK design — kbench.judge_llm, kbench.chats.new(), content_types for image handling — is well thought out and a pleasure to work with.

This entire journey — from multi-LLM architecture to single-LLM benchmark, through 9 platform bugs and countless debug sessions — has been an extraordinary learning experience. It taught me more about LLM behavior, API quirks, safety systems, and prompt engineering than any tutorial or documentation could. Every bug I encountered deepened my understanding of how these systems work under the hood.

These bug reports come from a place of deep appreciation for the platform. I have invested significant time in developing a 10-scenario WWTP benchmark suite because I believe in what this platform makes possible. I am truly grateful to be part of the Kaggle Benchmarks community, and I hope these findings are useful to the team in making the platform even better for the next developer.

Thank you to the Kaggle team for building something this powerful and for giving developers like me the tools to ask meaningful questions about what LLMs can really do.

Mehmet ISIK
Data Scientist | Kaggle Grandmaster
February 2026