

Reproducibility Report for *AgentGroupChat*

Abstract

This report presents a reproducibility study of the open-source project *AgentGroupChat* (paper: arXiv:2403.13433v2). Following the course requirement, I selected one clearly scoped target claim from the paper, ran the released codebase, reproduced a meaningful subset of the reported evaluation, and then implemented one controlled modification with measured impact. The target claim is Table 3, which evaluates LLM instruction-following and context-understanding abilities through nine quantitative metrics. I chose Table 3 over Table 2 because it is significantly cheaper to run and easier to isolate into a repeatable benchmark harness, while still being central to the paper’s methodology.

I first studied the project architecture and prompt templates, then implemented and validated an OpenAI-compatible reproduction script based on the repository’s own prompt files and scenario assets. My reproduced scores for GPT-4-Turbo and GPT-3.5-Turbo partly match the trend in the paper, but exact numbers are not recoverable. The major reason is that the paper’s original benchmark dataset and exact evaluation logs are not fully released; this is also consistent with the repository TODO note that benchmark artifacts were not fully open-sourced. Therefore, the current reproduction should be interpreted as *approximate behavioral replication*, not exact numerical replication.

As the controlled modification, I added a new model (gpt-5.2) to the same benchmark pipeline and compared the measured impact under the same script and scoring rules. The result demonstrates that the framework can evaluate model swaps, but it also shows how endpoint differences, model drift, prompt-format fragility, and sample-size differences can dominate absolute scores. The report concludes with practical recommendations for future reproducibility users.

1 Assignment Context and Goal

The assignment requires an individual reproducibility effort on an agentic AI system, including four core elements: running an existing project, reproducing at least one reported result or behavior, making one small but meaningful modification, and reporting what changed with evidence. I selected *AgentGroupChat* because it is explicitly agentic in the practical course sense: it operates as a multi-step simulation with memory, planning, dialogue actions, and multi-agent interaction loops rather than a single prompt-response call.

The specific target was chosen after reading both the paper and the repository. The project reports two quantitative evaluations near the center of its claims: Table 2 (role-alignment behavior

in inheritance dispute simulation) and Table 3 (instruction and environment understanding of the LLM core). In this work I focused on Table 3 as the primary reproducibility target. This decision is methodologically justified. Table 2 depends on repeated full simulation rounds with high token cost and broad interaction dynamics. Table 3, in contrast, isolates model-level capabilities with measurable pass/fail outputs and can be reconstructed from prompt templates and scenario files already available in the repository. This makes Table 3 a better fit for limited compute budgets while still testing a core claim in the paper.

The report is structured to match the submission expectation: project summary, setup notes, reproduction target and metric definitions, reproduced results versus paper values, controlled modification and post-modification outcomes, a debug diary of blockers and fixes, and a final assessment of what is reproducible and what is not.

2 Project Understanding: What the System Does

AgentGroupChat is a simulation framework designed to study how language interactions among role-playing agents can produce emergent group behavior. In the paper, the authors define four narrative scenarios (Inheritance Disputes, Law Court Debates, Philosophical Discourses, and Movie Casting Contention), and each scenario is run as staged multi-agent interaction. The architecture combines environment-level game stages with an internal agent architecture called the Verbal Strategist (VS) Agent.

From the paper and code, the VS Agent consists of persona components and action components. Persona includes fixed role scratch/profile, dynamic belief, relationship judgement, and memory. Action includes choose, speak, summarize, reflect, and vote, with planning and perception logic around these steps. In the repository this architecture is concretely implemented across `main.py`, `character/character_class.py`, and action modules such as `choose.py`, `facechat.py`, `reflection.py`, and `summarization.py`. Prompt templates are externalized under `prompt/prompt_files/`, which made them directly reusable for reproduction.

The simulation environment is scenario-driven. In the succession scenario used in this report, character and resource initial states are loaded from `storage/succession/initial_version/`. The scenario has predefined role metadata, beliefs, and initial resources. The full game includes private chat, confidential meetings, group chat, and settlement/voting stages. This explains why full end-to-end reproduction can be costly and stochastic: many coupled interactions occur before final outcomes.

The paper’s Table 3 is especially relevant because it decouples part of this complexity and evaluates whether an LLM can (1) follow strict output format instructions and (2) correctly understand counts/structure from context. This metric family is close to implementation-level reliability. In agent systems, this reliability is critical because one malformed output can break downstream control flow even when language quality looks good.

3 Reproduction Target and Metric Semantics

The direct target claim in this report is Table 3 of the paper, especially the rows for GPT-4-Turbo and GPT-3.5-Turbo. The paper labels Table 3 as an evaluation of “ability in following instructed output format and understanding the given context.” The metric abbreviations and sample meanings are described by the paper’s Table 4.

To avoid ambiguity, I restate the nine metrics with implementation semantics as used in reproduction:

Table 1: Operational definition of Table 3 metrics used in this reproduction

Metric	Operational meaning in code-level evaluation
CS	Whether the model returns the complete action space list in the required format for choose prompts.
BUS	Whether belief update values are within allowed numeric bounds.
RUS	Whether relationship update values are within allowed numeric bounds.
NoR (rel)	Whether the number of relationship updates equals the required length.
NoB	Whether the number of belief updates equals the required length.
NoA	Whether the model correctly reports the number of action-history items received.
NoCR	Whether the model correctly reports the number of dialogue rounds in chat history.
NoC	Whether the model correctly counts the number of character IDs in a provided list.
NoR (res)	Whether the model correctly counts the number of resource IDs in a provided list.

A key technical note is that “NoR” appears twice in the paper context: once as relationship-update length under instruction understanding and once as resource count under environment understanding. In my report I disambiguate them as NoR(rel) and NoR(res).

The benchmark score for each metric is computed as exact-match pass rate:

$$\text{Score}_m = 100 \times \frac{\#\text{correct outputs under metric } m}{\#\text{samples evaluated for } m}.$$

These are not task-success or persuasion-quality metrics. They are strict compliance metrics under controlled prompt and parsing rules. This distinction matters when interpreting high or low values.

4 Setup and Reproduction Pipeline

The experiments were run in a Windows local environment with Python and the repository dependencies from `requirements.txt`. The target scenario data came from `storage/succession/initial_version/`, and prompt templates were reused directly from `prompt/prompt_files/`. I did not re-implement the original agent system from scratch; instead, I built a lightweight harness around

existing prompts and schema, which is consistent with standard reproducibility practice and with the assignment guidance.

The core reproduction script is `scripts/reproduce_table3_openai.py`. It invokes OpenAI-compatible chat completion APIs and evaluates the nine metrics by rendering repository prompt templates and parsing model outputs. Default sample sizes are intentionally small for practical compute cost: `n_cs=30, n_update=20, n_noa=30, n_nocr=30, n_noc=20, n_nor_res=20`. This yields 150 calls per model and 450 calls for three models in one full run. The script also supports repeated runs with shifted seeds (`-repeats, -seed-step`) for variance reduction.

Important clarification for reproducibility attribution: the original AgentGroup codebase does not provide a standalone public Table 3 benchmark script for direct rerun. All benchmark code used in this report was implemented by me for this reproduction workflow, and it is stored in `scripts/reproduce_table3_openai.py`.

The pipeline is model-agnostic across OpenAI-compatible endpoints. In addition to base API calling, the script includes practical robustness features needed in real reproduction work: flexible parsing for minor format variation, retry handling for transient HTTP/server errors, truncation-aware retries when responses end with `finish_reason=length`, optional handling or removal of reasoning-specific fields depending on gateway support, and a warning when the returned model identifier does not match the requested one. These additions were not cosmetic; they were introduced because earlier runs produced pathological outputs (all zeros or implausibly inflated pass rates) due to parser brittleness and endpoint behavior.

For ethical and reproducibility hygiene, API keys should be provided through environment variables in final submission artifacts. Even though assignment work often starts with inline testing for convenience, final deliverables should avoid committing secrets.

5 Reproduction Results Against Paper Values

Table 2 compares paper-reported rows with my reproduced rows for GPT-4-Turbo and GPT-3.5-Turbo. The reproduced rows are exactly the values used for this report.

Table 2: Paper values versus reproduced values for Table 3 target rows

Model / Source	CS	BUS	RUS	NoR(rel)	NoB	NoA	NoCR	NoC	NoR(res)
GPT-4-Turbo (paper)	92.3	100.0	100.0	100.0	100.0	77.3	100.0	100.0	100.0
GPT-4-Turbo (reproduced)	96.7	100.0	100.0	100.0	100.0	46.7	100.0	100.0	100.0
Δ (reproduced - paper)	+4.4	0.0	0.0	0.0	0.0	-30.6	0.0	0.0	0.0
GPT-3.5-Turbo (paper)	95.4	100.0	85.3	100.0	100.0	38.6	76.9	94.6	98.1
GPT-3.5-Turbo (reproduced)	93.3	95.0	90.0	90.0	100.0	46.7	96.7	40.0	15.0
Δ (reproduced - paper)	-2.1	-5.0	+4.7	-10.0	0.0	+8.1	+19.8	-54.6	-83.1

The GPT-4-Turbo row is partially close and partially divergent. Eight metrics match closely or exactly, while NoA is substantially lower than the paper value. This is a known stress point in this benchmark style: NoA requires exact counting from long memory-like text, and it is sensitive

to model truncation behavior and prompt-format interpretation. In other words, small endpoint or prompt-output differences disproportionately affect this single metric.

The GPT-3.5-Turbo row is farther from the paper. Some columns are reasonably aligned, but NoC and NoR(res) are much lower. This indicates that strict counting tasks in compact formats can fail unpredictably for smaller/older model snapshots under OpenAI-compatible gateways. It also confirms that score reproduction is not only a function of model family name; model snapshot, routing policy, and output parser tolerance have large effects.

At a summary level, the mean absolute deviation from paper values is low for GPT-4-Turbo and much higher for GPT-3.5-Turbo. This pattern supports the claim that partial behavioral reproduction is possible, but exact table-level replication is not currently attainable with the public artifacts alone.

One additional reason is model-version drift at the provider side. Even when the API model names are still `gpt-4-turbo` and `gpt-3.5-turbo`, their underlying snapshots can be updated after paper publication, so behavior and scores may change without any modification in my local code; this alone can shift several metric points.

6 Controlled Modification and Measured Impact

The controlled modification in this assignment is a model swap/extension on exactly the same benchmark harness: adding `gpt-5.2` as an additional model while keeping prompts, scoring logic, sample sizes, and seed policy unchanged. This is aligned with assignment guidance for small but meaningful modifications and allows direct measurement of impact.

The modified run output is shown in Table 3. For this report I use the final accepted data specified in the experiment workflow.

Table 3: Post-modification benchmark run with added `gpt-5.2`

Model	CS	BUS	RUS	NoR(rel)	NoB	NoA	NoCR	NoC	NoR(res)
<code>gpt-5.2</code>	96.7	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
<code>gpt-4-turbo</code>	96.7	100.0	100.0	100.0	100.0	46.7	100.0	100.0	100.0
<code>gpt-3.5-turbo</code>	93.3	95.0	90.0	90.0	100.0	46.7	96.7	40.0	15.0

The measured impact of the modification is straightforward under the current harness: `gpt-5.2` is strongest overall and especially improves the environment-understanding metrics that were unstable for GPT-3.5 and partly weak for GPT-4 in this run. However, this should not be over-claimed. Since the benchmark here is reconstructed rather than the paper’s original hidden dataset, the result is best interpreted as an internal comparison under a consistent local harness, not as a direct replacement of paper claims.

7 Debug Diary: Main Blockers and Fixes

The first major blocker was a degenerate run that produced all-zero scores. This happened because early parsing logic was too brittle: it assumed exact section headers and strict formatting from every model output. In practice, even valid responses can differ by minor header wording, punctuation, or concise numeric-only output. I fixed this by improving parsers for action space extraction, flexible integer extraction, and update-output parsing while preserving exact-match semantics where required. After this fix, the evaluator started reflecting nontrivial behavior rather than parser failure.

The second blocker was the opposite failure mode: some runs looked unrealistically high, including near-perfect columns that did not align with qualitative expectations. Investigation showed a truncation interaction in long outputs, especially for counting-heavy prompts like NoA. When a gateway truncates completion, the visible output may omit key fields, and naive logic can accidentally over-credit or under-credit. I added explicit handling for `finish_reason=length`, with automatic retry under larger token budgets, and introduced model/provider-specific safeguards. This made runs more stable and reduced obvious artifacts.

The third blocker was endpoint compatibility. OpenAI-compatible gateways do not always support the same optional request fields, and some may remap model names internally. I added a compatibility path that removes unsupported reasoning fields when rejected and warns when returned model IDs do not match requested IDs. This is important for reproducibility transparency: if endpoint routing silently substitutes a different model, table comparison is no longer clean.

A final practical issue is benchmark availability. The paper reports large sample counts (for example, thousands of samples in several Table 4 categories), but the released repository does not include a standalone official benchmark dataset or full table-generation script for exact replication. The repository TODO also mentions benchmark release as pending. Therefore, this assignment necessarily uses a reconstructed, prompt-faithful evaluation subset instead of exact original sampling.

8 Conclusions, Reproducibility Assessment, and Recommendations

This reproducibility exercise succeeded at the level expected for practical agent-system replication under limited resources. I was able to run the project components, reconstruct a meaningful Table 3-style evaluator from released prompts and scenario assets, reproduce key qualitative trends, and add one controlled model modification with measured impact. The workflow is transparent, script-based, and rerunnable.

At the same time, exact numerical reproduction of paper Table 3 should not be claimed. The necessary conditions for exact recovery are not fully available: original benchmark instances, full sampling policy, exact historical model snapshots, and full evaluation logs are missing or only partially represented. In addition, `gpt-4-turbo` and `gpt-3.5-turbo` are likely updated over time, so the same model names may no longer correspond to the exact model behavior used in the paper period. This is the main reason why some columns align closely while others diverge sharply. The result is therefore a rigorous *approximate reproducibility* report, not an exact replication report.

The clearest technical insight from this work is that these metrics mostly evaluate interface reliability: structured output obedience and context-count extraction. They are valuable for agent engineering because they predict pipeline breakage risk, but they are not equivalent to end-task quality or emergent-strategy quality. A model can score high in Table 3-style checks and still fail strategic persuasion objectives in the full multi-agent game.

For future users of this repository, I recommend three improvements to raise reproducibility quality: first, release the exact benchmark dataset and deterministic sampling scripts used to generate Table 3 and Table 2; second, publish model snapshot metadata and endpoint settings used for each table; third, include a standard reproducibility profile with fixed seeds and expected confidence intervals from repeated trials. With those additions, the community could move from approximate reproduction to near-exact table-level verification.

In summary, what is reproducible today is the method shape, prompt logic, and partial metric trend. What is not reproducible today is the exact paper numbers. The controlled modification with `gpt-5.2` demonstrates that the reconstructed evaluator is still useful for comparative analysis, provided claims are framed at the right level.