BENCHAGENTS: AUTOMATED BENCHMARK CREATION WITH AGENT INTERACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluations are limited by benchmark availability. As models evolve, there is a need to create benchmarks that can measure progress on new generative capabilities. However, creating new benchmarks through human annotations is slow and expensive, restricting comprehensive evaluations for any capability. We introduce BENCHAGENTS, a framework that methodically leverages large language models (LLMs) to automate benchmark creation for complex capabilities while inherently ensuring data and metric quality. BENCHAGENTS decomposes the benchmark creation process into planning, generation, data verification, and evaluation, each of which is executed by an LLM agent. These agents interact with each other and utilize human-in-the-loop feedback from benchmark developers to explicitly improve and flexibly control data diversity and quality. We use BENCHAGENTS to create benchmarks to evaluate capabilities related to planning and constraint satisfaction during text generation. We then use these benchmarks to study seven state-of-the-art models and extract new insights on common failure modes and model differences.

1 INTRODUCTION

AI advancements are progressing rapidly, with new models frequently showing enhanced capabilities. Evaluation datasets are essential for testing these claims, but they are expensive to produce and can quickly become saturated, due to the fast pace of model improvements (Balachandran et al., 2024), or contaminated (Zhang et al., 2024a). In the absence of benchmarks, new capabilities are often demonstrated with anecdotal, qualitative examples or small, non-comprehensive test sets; this offers limited insight into actual model performance. This highlights the need for scalable, dynamic benchmarking methods to enable fast and reliable model evaluation.

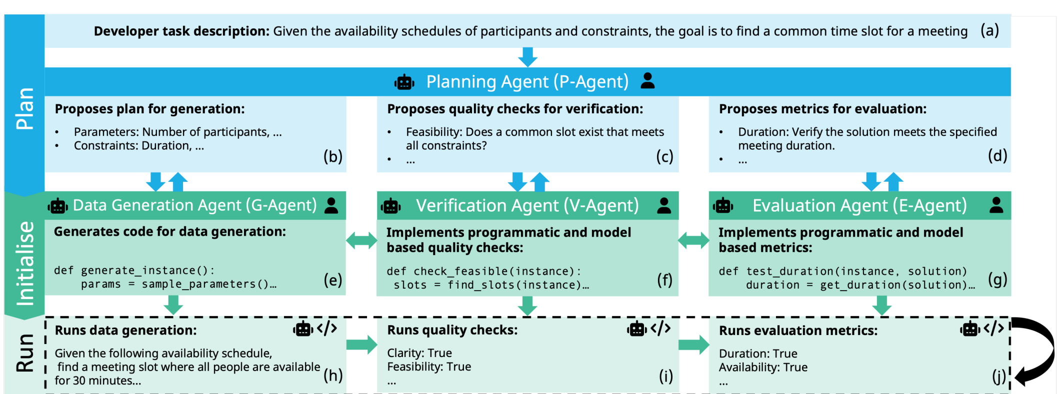


Figure 1: Overview of BENCHAGENTS. P-AGENT generates a plan for data generation and communicates this to G-AGENT. G-AGENT writes code for data generation and communicates to all agents. P-AGENT generates plans for evaluation and verification and communicates these to the respective agents. E-AGENT and V-AGENT write code for evaluation and verification. For each instance, generation, verification and evaluation processes are run.

Traditionally, benchmark creation involved designing data requirements and recruiting human annotators to provide test instances. Whilst ensuring quality, this process is costly, time-consuming, and difficult to scale. Previous work proposed methods for synthetic test data generation via prompt templates (Wang et al., 2024; Xia et al., 2024a; Yuan et al., 2024) or by using programmatic workflows for narrow domains (Zhu et al., 2024; Zhang et al., 2024b). These methods, however, are unable to easily generalize to a broader set of complex and generative tasks. Parallel efforts for training data synthesis have been proposed (Li et al., 2023b; Mitra et al., 2024; Li et al., 2023a). However, they do not usually transfer for generating evaluation datasets, due to stricter quality and diversity requirements.

We propose BENCHAGENTS[1], a *multi-agent evaluation framework* for automated, high-quality, and diverse benchmark creation. BENCHAGENTS breaks down benchmark creation into four components and instantiates each component via dedicated LLM agents as shown in Fig. 1. The Planning Agent creates a high-level plan/specification based on the problem and user requirements, breaks it down to tasks, and communicates the plan with the other agents. The plans may contain elements including but not limited to *parameters* and their values to guide the data generation and enable dis-aggregations along important dimensions, definitions for *quality checks* during data verification, and *metrics* for model evaluation. The Data Generation Agent implements the plan programmatically and generates diverse benchmark data. Next, the Verification Agent formulates and executes fine-grained data quality checks to ensure quality control for the generated examples. Finally, the Evaluation Agent produces evaluation code and prompts for one or more metrics used for assessing target model performance.

Grounding the data generation on a shared plan (that consists of a structured set of parameters) across agents enables *precise control on the diversity of the data distribution*. It also facilitates the creation of data quality checks and model evaluation criteria that are consistent with the initial plan. Whenever applicable, BENCHAGENTS by design allows for additional developer feedback at each stage of the process to ensure transparency, control and quality in the produced benchmarks.

We demonstrate the utility of BENCHAGENTS, by generating benchmarks on two complex and generative problem settings - calendar scheduling (BA-CALENDAR) and constrained long-form text generation (BA-TEXT) - each with $2,000$ test instances. These are tasks where current benchmarks are lacking and state-of-the-art (SOTA) LLMs perform poorly. We then evaluate seven SOTA LLMs on both benchmarks. The generated benchmarks enable fine-grained dis-aggregations along multiple important dimensions, such as complexity. We find that (i) all LLMs struggle with joint constraint satisfaction across both datasets, with performance decreasing as the number of constraints increases; (ii) LLMs differ in their prioritisation of constraints when all cannot be met; and (iii) failures often involve constraints requiring numerical or logical reasoning.

In summary, our contributions are:

- We introduce BENCHAGENTS—a multi-agent framework which utilises interacting LLM agents to design and create benchmarks for complex and generative capabilities, while ensuring data and metric quality (§ 3).
- Using BENCHAGENTS, we create two diverse and high-quality benchmarks, BA-CALENDAR and BA-TEXT, to evaluate LLMs on two complex problems (§ 4).
- Evaluating seven SOTA LLMs on the two benchmarks, we offer insights on models' capabilities for generative and complex tasks (§ 6).

2 RELATED WORK

A growing body of literature looks at leveraging algorithms and LLMs to automate parts of benchmark creation. This can be divided in two areas: dynamic benchmark creation for narrow domains, and extending existing benchmarks.

Dynamic Benchmark Creation: Zhang et al. (2024b) create a data generation algorithm that selects images and scene graphs from a corpus and generates input-output pairs based on question-answer templates for custom multi-modal evaluations. Yuan et al. (2024) propose `AutoBench` for aligning

[1]Our code implementing BENCHAGENTS is available at https://anonymous.4open.science/r/BenchAgents-752D

vision-language model evaluation, annotating images with question-answer pairs using LLMs for skill-based analysis. Zhu et al. (2024) design an evaluation data generation algorithm for reasoning tasks using graphs. These methods offer a dynamic way for users to produce fine-grained evaluation data based on pre-defined tasks. However, the data generators are *manually designed, limiting generalisability and scalability*.

Benchmark Extension: Li et al. (2024) propose `AutoBencher` to optimize existing benchmarks to improve diversity and quality. It does so by generating question-answer pairs by proposing topics and retrieving relevant information from databases using an LLM. `AutoBencher` is apt for improving existing benchmarks, but is *non-trivial to extend* outside of question-answering domains to generative settings. Wang et al. (2024) present a multi-agent framework for dynamically augmenting benchmarks for scalability and robustness. Xia et al. (2024a) look at evolving existing coding benchmarks into different coding domains using LLM-based augmentation and verification with manual examination. Though dynamic and scalable, these approaches *mandatorily require a seed dataset* to bootstrap the process. Table 1 summarizes these comparisons.

Method	Controllable Parameters	Automated Verification	Supports No Seed Dataset	Automated Benchmark Design	Generative Settings
Xia et al. (2024a)	✔	✔	✘	✘	✔
Zhang et al. (2024b)	✔	-	✘	✘	✘
Wang et al. (2024)	✔	✔	✘	✘	✘
Yuan et al. (2024)	✔	✔	✘	✘	✘
Li et al. (2024)	✔	✔	✘	✘	✘
Zhu et al. (2024)	✔	-	✔	✘	✘
BENCHAGENTS	✔	✔	✔	✔	✔

Table 1: Comparison of automated benchmark creation frameworks.

3 DESIGN OF BENCHAGENTS

BENCHAGENTS automates benchmark creation for complex NLP tasks using LLM agents, while accommodating developer-in-the-loop (DIL) feedback. An LLM agent is defined as the combination of an LLM and an *agent configuration* (i.e., a set of prompts) provided by the developer or another agent. An LLM agent is specialised for a particular task in the workflow of benchmark creation.

At a high-level, BENCHAGENTS takes as input a description of the task to be evaluated and optionally a seed set of prompts representing the type of evaluation benchmark intended. BENCHAGENTS then uses multiple LLM agents—Planning Agent for benchmark planning (P-AGENT: § 3.1), Data Generation Agent for instance generation (G-AGENT: § 3.2), Verification Agent for instance quality verification (V-AGENT: § 3.3) and Evaluation Agent for response evaluation (E-AGENT: § 3.4)—sequentially, for benchmark creation. The final output of BENCHAGENTS consists of (i) verified and diverse instances, and (ii) metrics to evaluate outputs of a (target) model on these benchmark instances. By dividing responsibilities across agents, BENCHAGENTS enables more precise debugging of the benchmark creation process.

BENCHAGENTS follows a *hybrid* LLM and code execution approach to support automation. Such an approach is effective as some automation tasks are best handled by code (even if the code is generated by an LLM) while others are more easily managed through LLM calls. An overview of the framework is described in Fig. 1, considering the task of calendar planning as an example.

3.1 PLANNING AGENT (P-AGENT)

Each instance BENCHAGENTS generates contains (i) a *prompt* to be used for (target) model evaluation, (ii) task-specific *parameters*, and (iii) *constraints*. Parameters are defined as variables on which a prompt is grounded. In Fig.1 box b, parameters include the number of participants or the earliest meeting start time. In contrast, constraints are defined as restrictions placed on the solution to a prompt. For example, in Fig.1 box b, the meeting duration constrains the space of possible to solutions to those with a specific meeting duration.

P-AGENT takes a task description as an input and a set of optional seed prompts and proposes a plan for the other agents to execute. As shown in Fig.1 box b, for data generation, P-AGENT proposes and defines multiple parameters including the range and distribution of values for each parameter. In addition to parameters, the plan also includes a set of constraints. After knowing the set of input parameters and query constraints, G-AGENT can then proceed with sampling the inputs from the parameters' range and the corresponding queries from the constraints' range (more details in § 3.2). This controlled sampling process is essential in ensuring benchmark data diversity.

P-AGENT also guides V-AGENT by proposing a suite of quality checks that each instance in the benchmark should pass, including clarity, completeness, consistency, feasibility, and complexity (Fig.1 box c); more details are in § 3.3. At their core, these checks ensure that the generated instances are exemplar representatives of the task, and that they can support reliable evaluations. Finally, P-AGENT proposes evaluation metrics to E-AGENT for assessing the quality of model responses on the generated benchmark prompts (Fig.1 box d); more details are in § 3.4.

Upon plan creation, developers can further steer the benchmark creation to better align it with their measurement goals by refining elements like parameters (and ranges), constraints, or metrics.

3.2 DATA GENERATION AGENT (G-AGENT)

G-AGENT transforms the plan into concrete benchmark instances (Fig. 1 box e). To do so, G-AGENT designs the template for the final instance generation prompt. This template is flexible i.e., it can support augmenting the prompt to ensure semantic equivalence but syntactic diversity. To populate the prompt template, G-AGENT, first generates the code needed for parameter sampling given the ranges from P-AGENT (Fig. 1 box b). It then adapts the code to sample and apply the constraints in the plan.

Since generation may be challenging when the range of parameters and constraints is large and conflicts may arise, the current template allows developers to prioritise which parameters and constraints should be prioritised at generation time.

3.3 VERIFICATION AGENT (V-AGENT)

V-AGENT analyses generated instances and evaluates their fitness for use in the benchmark. To do so, it uses the following *quality checks* by generating code for programmatic checks or by generating prompts for model-based checks:

1. Clarity: The prompt should be understandable and unambiguous to developers and target models.

2. Completeness: The prompt should contain all the constraints mentioned in the plan. For the example in Fig. 1, meeting duration should be present in all prompts.

3. Consistency: When a parameter or constraint is realised in the prompt, the value should be consistent. For the example in Fig. 1, the "number of participants" parameter should be consistent with the number of participants in the schedules.

4. Feasibility: The constraints should define a feasible problem. For e.g, in Fig. 1, a common time slot should exist that satisfies all constraints.

5. Complexity: The constraints should be associated with a measure of how challenging they make the problem. To capture this, a task-specific metric should be defined by the P-AGENT. For the example in Fig. 1, the metric involves the ratio of feasible slots to all slots.

For the example in Fig. 1, box f highlights the quality checks by V-AGENT for verifying feasibility in calendar scheduling using the specification provided by P-AGENT in box c. The generated code (used for verification) is manually reviewed for correctness. The same is done for the model-based verification methods.

3.4 EVALUATION AGENT (E-AGENT)

E-AGENT evaluates the solutions generated by target models. This is necessary in generative settings since we cannot always simply compare a solution to ground truth. E-AGENT operates based on the evaluation metrics defined by P-AGENT, which are all grounded on the set of constraints present in the plan. More specifically, for each constraint, there exists an evaluation metric that marks whether the constraint was satisfied or not (pass vs. fail) by a proposed solution. E-AGENT can implement both model-based and programmatic metrics. Developers can choose either of the options and mark their preference in the plan. For example, for calendar planning we generated both options and decided to go with programmatic metrics as they were fully implementable for this task. For the example in Fig. 1, E-AGENT is required to check if the solution conforms to the duration constraint (box g). To achieve this, it generates code to extract and check the duration from the solution, given access to parameters and constraints (box d).

In addition to metrics associated to a single constraint, E-AGENT also computes the fraction of constraints satisfied from the whole list of constraints (i.e., "fraction passed") as well as whether all constraints were satisfied (i.e., "pass all").

4 BENCHMARK GENERATION

Next, we describe how we leveraged BENCHAGENTS to generate benchmarks for two challenging tasks: calendar scheduling and constrained long-form text generation. For all agents, we use GPT-4o as the LLM model and the specific agent configurations are reported in Appendices H.1 and I.1.

4.1 CALENDAR SCHEDULING (BA-CALENDAR)

Calendar scheduling is an important task that is relevant for several calendar and mail applications. In addition, it also constitutes a domain where planning and reasoning are important. Previous work (Zheng et al., 2024) has proposed initial benchmarks on the task (NATURALPLAN) but the scheduling part of the benchmark was shown to be saturated in evaluations of the o1-preview model (Valmeekam et al., 2024), with the majority of instances containing only two participants and one day of the week (see Appendix B). Therefore, we generate BA-CALENDAR that simulates a challenging and closer to real-world setting, where the problem involves more constraints.

As part of the plan, P-AGENT proposes (i) various parameters including the number of participants, number of days with availability, days of week, and (ii) multiple constraints like each participant's availability, required meeting duration, buffer times. G-AGENT writes code for generating diverse data based on P-AGENT's proposed parameters and constraints. Fig. 15 (in Appendix H) shows an example prompt from a generated instance. V-AGENT initialises (i) model-based checks for clarity, completeness, and consistency, and (ii) programmatic checks for feasibility. Further, V-AGENT also implements a task-specific programmatic check for constrainedness as a measure of complexity: the ratio of number of feasible solutions to number of time slots where at least one participant is available. Finally, E-AGENT initialises programmatic metrics for the satisfaction of each constraint defined by P-AGENT. For a full list of parameters and constraints along with details of each agent's implementation see Appendix H, where we also differentiate between what was provided by the developer and what was generated by the model.

4.2 CONSTRAINED LONG-FORM TEXT GENERATION (BA-TEXT)

Constrained long-form text generation requires models to plan and produce a long response to a user query that meets all the constraints in the query. This capability is relevant for creative and technical writing, as important pillars of productivity applications. Existing datasets that aim to test these capabilities evaluate only on format constraints (Xia et al., 2024b), short-form solutions (Zhou et al., 2023) and include relatively simple constraints (Yao et al., 2023). In comparison, our BA-TEXT focuses on long-form generations with complex content-based constraints.

P-AGENT proposes (i) parameters like user, role, task, and (ii) constraints including:

- *Positive constraints*: inclusion of certain content like topics or entities in the generation.
- *Negative constraints*: exclusion of content from the generation.
- *Positional constraints*: inclusion at a specific position (e.g., paragraph) in the generation.
- *Sequencing constraints*: inclusion of certain content in a specific sequence in the generation.
- *Conditional constraints*: inclusion or exclusion based on some conditions.

- *Iterative constraints*: any previously defined constraints applied iteratively.

G-AGENT generates data generation code that aligns with P-AGENT's proposed parameters and constraints. An example prompt from a generated instance may be found in Fig. 18 (Appendix I). Amongst all quality checks from V-AGENT, the only programmatic check is for constrainedness defined as number of constraints applied to total number of constraints. The model-based quality checks include clarity, completeness, consistency, and feasibility. Finally, E-AGENT initialises model-based metrics[2] for the satisfaction of each constraint and topic consistency based on proposed metrics from P-AGENT. For a full list of parameters and constraint examples along with details of the data generation procedure, see Appendix I, where we also distinguish between what was supplied by the agent configuration, model and developer-in-the-loop feedback.

5 BENCHMARK QUALITY ASSESSMENT

To validate the quality and diversity of benchmarks produced by BENCHAGENTS, we conduct a quality assessment consisting of automatic and human based assessments.

5.1 ARE GENERATED INSTANCES HIGH QUALITY?

Metric	BA-CALENDAR	BA-TEXT
Clarity	0.99	0.84
Completeness	0.96	0.94
Consistency	0.96	0.89
Feasibility	0.93	0.73

Table 2: Pass rate for verification quality checks.

We measure quality of the generated benchmark using two measures: (i) conformance with the verification checks specified in § 3.3, and (ii) coverage of important parameters useful for comprehensive evaluation. Table 2 shows the pass rate for each of these quality checks. We observe that the G-AGENT produces high-quality instances for both tasks, indicating good quality of our benchmark. We see higher quality for all criteria in BA-CALENDAR compared to BA-TEXT. For BA-TEXT, we note that the "feasibility" criterion is substantially lower that the other criteria. On investigation, we find that the feasibility test fails sometimes due to conflicting constraints. For example, the G-AGENT has a tendency to generate positional and sequencing constraints that contradict each-other. To control for quality, BENCHAGENTS excludes any instance from the final benchmark that fail any of the quality checks.

We discuss the coverage of various parameters for BA-CALENDAR in Appendix B, and for BA-TEXT in Appendix C. We observe that BENCHAGENTS generates diverse instances ensuring superior coverage, even compared to manually curated benchmarks (see Appendix B).

5.2 ARE MODEL-BASED CHECKS RELIABLE?

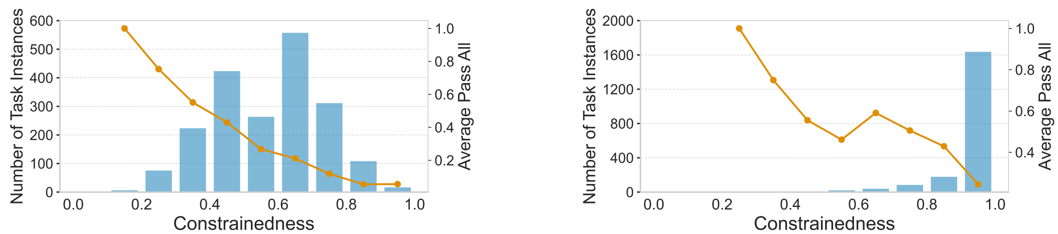
Recall that for certain constraints and parameters, the V-AGENT employs model-based verification checks. We conduct a human assessment of the V-AGENT for the model-based checks to evaluate their reliability. In this study, for each generated dataset, we take 50 instances produced from the G-AGENT (before filtering by the V-AGENT). For each instance, we collect two human annotations for each model-based verification check performed (more details in § 4). For the ground truth, we consider an instance to pass a verification check only if both annotators mark it as passed.

Table 3 reports high accuracy of the tests w.r.t human annotations, suggesting that the model-based checks generated by V-AGENT are reliable. We also report the precision and recall for each dataset in Appendix D.1, Fig. 9. A similar study (with similar conclusions) is conducted to evaluate the reliability of model-based evaluations conducted by E-AGENT in Appendix D.2.

[2]Since the task is open-ended and generative, we instruct the E-AGENT to use LLM-as-judge for evaluation. We evaluate how well the metrics align with human judgements in Appendix D.2.

Metric	BA-Calendar	BA-Text
Clarity	0.96	0.80
Completeness	0.90	0.96
Consistency	0.86	0.76
Feasibility	-	0.76

Table 3: Accuracy for V-Agent model-based test results with human annotated ground truths.



(a) BA-Text: Constrainedness (bar) and GPT-4o average pass all (line).

(b) BA-Calendar: Constrainedness (bar) and GPT-4o average pass all (line).

Figure 2: Comparison of constrainedness metrics for BA-Text and BA-Calendar.

Further, BenchAgents allows the benchmark creators to verify, check, and provide feedback on the outputs of each agent to ensure high quality and align the benchmark to their requirements. Appendix G reports the extent of edits developers provided in addition for the two benchmarks.

5.3 Are Generated Instances Difficult?

While we did not explicitly optimize for difficulty, any good benchmark should not be trivial to solve. To ensure there is a range of difficulty over our task instances, we borrow insight from prior work (Abdin et al., 2023; Yuksekgonul et al., 2023) and aim to assess if the respective constrainedness metrics (refer § 4.1 and § 4.2) from the V-Agent act as a reliable proxy for difficulty. We do so by comparing the average "pass all" from evaluating GPT-4o on both datasets.

In Fig. 2a, we bucketize task instances by their constrainedness measures and report both the average "pass all" and number of task instances in each bucket. We observe a monotonic decrease in "pass all" as constrainedness increases for buckets with more than 10 task instances across both datasets. This suggests that adding constraints indeed increases difficulty.

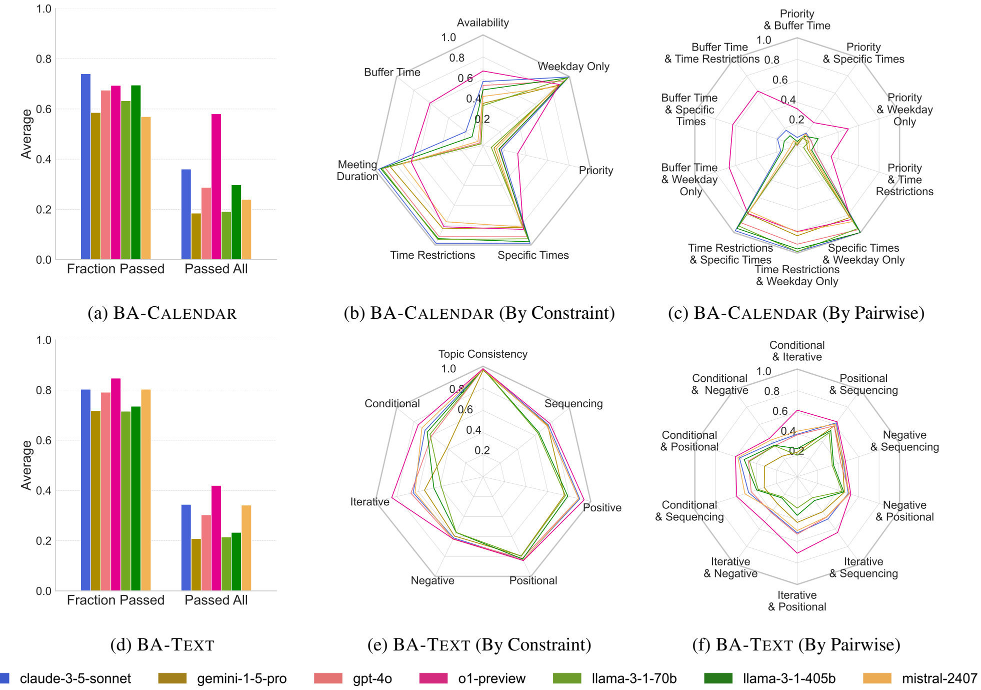
6 Model Analysis

We evaluated the generated benchmarks on OpenAI o1-preview (OpenAI, 2024b), GPT-4o (OpenAI, 2024a), Claude 3.5 (Anthropic, 2024), Gemini 1.5 Pro (Reid et al., 2024), Llama 3.1 70B and 405B (Dubey et al., 2024), and Mistral 2407 (MistralAI, 2024).[3] For both datasets, we report (i) *fraction passed*: the fraction of constraints passed per instance and solution, and (ii) *pass all*: whether the solution satisfies all constraints. We discuss salient findings below. Note that such findings are only possible due to the diverse set of parameters and constraints found and supported by BenchAgents.

6.1 Model Performance

Insight 1. *Models struggle to satisfy multiple constraints simultaneously.* Fig. 3a and Fig. 3d show that the "fraction passed" rate across all models is always substantially higher than the "passed all" rate. Most models see a drop of nearly 50% in performance when comparing the two metrics, showing that while they can satisfy *some* constraints in the query, reliably satisfying *all* of them is

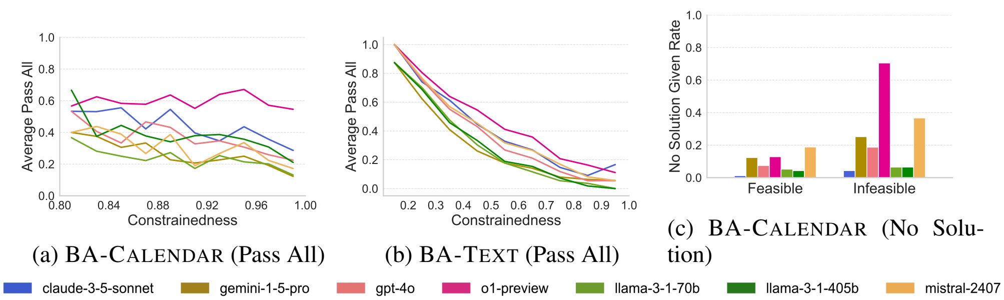
[3]Details of configurations are presented in Appendix E.



(a) BA-CALENDAR (b) BA-CALENDAR (By Constraint) (c) BA-CALENDAR (By Pairwise)

(d) BA-TEXT (e) BA-TEXT (By Constraint) (f) BA-TEXT (By Pairwise)

Figure 3: Model performance across different metrics: (a) and (b) show fraction passed and pass all for all task instances; (c), (d), (e), and (f) show pass rate for a given constraint or combination of constraints for task instances where constraints are applied.



(a) BA-CALENDAR (Pass All) (b) BA-TEXT (Pass All) (c) BA-CALENDAR (No Solution)

Figure 4: (a) and (b) Average pass all vs. constrainedness for BA-CALENDAR and BA-TEXT; (c) No Solution outcome for feasible and infeasible instances in BA-CALENDAR. If the instance is feasible, a low rate of no solution outcomes is preferred. Otherwise, if the instance is infeasible, a high rate is preferred.

a challenge. Of all models evaluated, o1-preview shows the *smallest gap* between the two metrics for BA-CALENDAR showing progress in this space with improved focus on reasoning and planning. However, o1-preview also struggles with satisfying all constraints in BA-TEXT.

Insight 2. *Models' prioritisation of constraint satisfaction varies.* For BA-CALENDAR, in Fig. 3b, we observe that in contrast to other strong models, o1-preview, Gemini-1-5-Pro and Mistral-2407 have lower performance with respect to simple constraints such as meeting duration or time restrictions. On further inspection, we see that these models are the most likely to respond with "no solution exists" for both feasible and infeasible instances (Fig. 4c). We observe that for challenging problems, these models choose to not give a solution while other models provide a solution that meets some constraints. In these cases, other models might prioritise simpler constraints like meeting duration compared to availability.

Insight 3. *Bigger models are better at joint constraint satisfaction for tasks with strict criteria.* For BA-CALENDAR and models with known size, in Fig. 3a, we observe "pass all" increases w.r.t. model size for Llama 3-1-70b, Mistral-2407 and Llama 3-1-405b, as expected. However, this trend is not consistent with BA-TEXT in Fig. 3d where we observe Mistral-2407 outperforming Llama 3-1-70b and Llama 3-1-405b, and matching performance of GPT-4o. We posit this difference to the more ambiguous and open-ended nature of BA-TEXT; BA-CALENDAR has well defined and stricter satisfaction criteria, allowing large models to plan and execute the task well.

Insight 4. *As the complexity in a query increases, individual and joint constraint satisfaction performance decreases.* This is observed in Fig. 4a and Fig. 4b for joint constraint satisfaction and Fig. 12 (in Appendix F) for individual constraint satisfaction. All models have very similar trends in pass rate at different constrainedness levels for BA-TEXT. Here, even strong reasoning models like o1-preview struggle at high levels of constrainedness. The trend while downward is more varied for BA-CALENDAR with o1-preview performing more similarly at different constrainedness levels. These results on performance w.r.t. constrainedness are consistent with findings in the literature; Yao et al. (2023); Abdin et al. (2023); Yuksekgonul et al. (2023) also find, when text generation tasks incorporate more constraints, GPT-4 only partially satisfies them. Similarly, in Fig. 13 (Appendix F), model performance monotonically decreases with the number of days and the number of participants. This finding is consistent with prior work (Zheng et al., 2024) showing the challenge with increasing complexity in search space.

6.2 MODEL FAILURES

Insight 5. *Numerical and logical reasoning requiring state tracking is challenging for most models.* In Fig. 3b and Fig. 3e respectively, we observe lower performance for constraints that involve logical reasoning such as availability, buffer time and priority for BA-CALENDAR and conditional, iterative and sequencing constraints for BA-TEXT. While o1-preview's performance exceeds other models with respect to these constraints (highlighting the importance of inference-time compute scaling (Snell et al., 2024)), we see that its performance on these constraints is still low indicating further room for improvement.

Insight 6. *Most models struggle with recognizing infeasible problems.* In Fig. 4c we see that most models have less than 40% accuracy in correctly identifying infeasible problems and responding with a correct "no solution" response. Here, o1-preview improves over prior models significantly pushing the rate to 70%. This shows that other models are strongly inclined to always respond with a solution even when no solution exists.

Insight 7. *All models struggle with specific combination of constraints.* Comparing Fig. 3e and Fig. 3f, we see that while model performance on positional and sequencing constraints individually is higher than others, all models struggle with the combination of the two where the best model performance is less than 60%. Similarly, models also jointly struggle with negative and positional constraints, especially smaller models like Mistral 2407. In BA-CALENDAR, for all tasks involving priority, performance is consistently low, potentially since priority itself is a challenging constraint. We pose that the performance gap for certain constraint combinations is due to ambiguity in how these constraints should be applied together.

7 LIMITATIONS

LLMs for Planning: P-AGENT uses an LLM to interpret user requirements and generate plans. To this end, the LLM may misinterpret ambiguous developer instructions. Further, the quality of the plan depends on the training data of the LLM, which may not have sufficient knowledge about the task to generate high-quality plans. Thus, the plan may not fully capture the developer's intended application. However, we highlight that in BENCHAGENTS the develop reviews and edits all plans before they are passed to the other agents, ensuring that the plans meet the desired use case.

LLMs for Initialisation: G-AGENT, V-AGENT and E-AGENT all use LLMs to initialise the respective data generation, verification, and evaluation processes with code and prompts for future LLM calls. This relies on the LLM to have coding capabilities and to understand user intent from the plan passed to it from the P-AGENT in order to write high-quality prompts. Again, we emphasise that the DIL reviews and edits these code and prompts before execution for quality control.

LLMs for Instance Generation: While executing the data generation processes, G-AGENT makes LLM calls, for example, in flexible template creation to generate task prompts from sets of parameters and constraints. Here, there is risk of hallucination and other inconsistencies between the desired parameters and constraints and the content of the prompt. This compounds the need for a verification process to ensure completeness and consistency between the prompt and the parameters and constraints. Further, the LLM may produce prompts that are unclear or infeasible, again highlighting the need for quality checks for clarity and feasibility. Finally, there is a risk that using LLMs for instance generation will result in instances that are not challenging enough since we use LLMs to create the data that we then evaluate them on. However, it is important to note the compositional nature of the data generation where BENCHAGENTS uses both LLM calls and code execution, with DIL for the initialisation stage. Our results show that both datasets we create are in fact challenging across all models. Further, since V-AGENT computes a metric for complexity, BENCHAGENTS supports filtering by complexity to ensure instances in the benchmark are sufficiently challenging.

LLMs for Model-based Quality Checks and Evaluation Metrics: BENCHAGENTS supports the use of model-based quality checks via the V-AGENT and evaluation metrics via the E-AGENT. There is a growing body of literature on the effectiveness of using LLM-as-Judge Zheng et al. (2023); Verga et al. (2024). There is a risk that LLMs provide inaccurate and/or biased responses resulting in low quality instances in our datasets and/or inaccurate evaluation metrics. However, we highlight that we conduct a human assessment of both the V-AGENT model-based quality checks and the E-AGENT evaluation metrics for both of our generated datasets (see Appendix D) and any further innovations to improve the reliability of model-based evaluations can be directly integrated into our framework.

Computational Costs: Running multiple LLM agents in BENCHAGENTS can be computationally expensive. As a result, the overall computational cost is higher than synthetic data generation frameworks that rely solely on code execution or single inference calls. This could limit the accessibility to developers with limited computational resources. Research on eliciting multi-agent capabilities with smaller models would improve the cost and efficiency of the entire framework.

Single LLM Used in Agents: For the generation of BA-CALENDAR and BA-TEXT, we use GPT-4o as the model in all our LLM agents. However, BENCHAGENTS is flexible and the compositional nature enables the use of different models for different agents. Again, we highlight that this is not a limitation of the framework but an implementation decision given the nature of the two generated benchmarks and resource availability. Future work can explore the benefit of using different LLMs for different agents.

7.1 ETHICAL CONSIDERATIONS

The aims of BENCHAGENTS are to assist in the automation and scalability of benchmark creation. However, since LLMs can hallucinate or misinterpret instructions, caution should be taken not to use model outputs without verification. BENCHAGENTS incorporates DIL feedback such that developers check the intermediate steps of the benchmark creation process and ensure the resulting benchmark meets the desired user requirements.

8 CONCLUSION

We introduce BENCHAGENTS, a framework for automatic benchmark creation, which employs multiple LLM agents to interact with each other and developers to create high-quality, diverse and challenging NLP benchmarks. BENCHAGENTS reduces developer effort while maintaining an appropriate level of human oversight. Further, the hybrid LLM and code execution approach enables flexibility to adapt to the generation of new complex and generative datasets in an efficient and controllable way. In addition, BENCHAGENTS enables the computation of dis-aggregated evaluation metrics. We highlight the advantage of this by producing two new benchmarks to evaluate planning and constraint satisfaction capabilities of models and present insights of the common failure modes across seven SOTA models.

REFERENCES

Marah I Abdin, Suriya Gunasekar, Varun Chandrasekaran, Jerry Li, Mert Yuksekgonul, Rahee Ghosh Peshawaria, Ranjita Naik, and Besmira Nushi. Kitab: Evaluating llms on constraint satisfaction for information retrieval. *arXiv preprint arXiv:2310.15511*, 2023.

Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024. Accessed: 2024-08-13.

Negar Arabzadeh, Siqing Huo, Nikhil Mehta, Qinqyun Wu, Chi Wang, Ahmed Awadallah, Charles L. A. Clarke, and Julia Kiseleva. Assessing and verifying task utility in llm-powered applications, 2024. URL https://arxiv.org/abs/2405.02178.

Vidhisha Balachandran, Jingya Chen, Neel Joshi, Besmira Nushi, Hamid Palangi, Eduardo Salinas, Vibhav Vineet, James Woffinden-Luey, and Safoora Yousefi. Eureka: Evaluating and understanding large foundation models. *arXiv preprint arXiv:2409.10566*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Xiang Lisa Li, Evan Zheran Liu, Percy Liang, and Tatsunori Hashimoto. Autobencher: Creating salient, novel, difficult datasets for language models, 2024. URL https://arxiv.org/abs/2407.08351.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report, 2023a. URL https://arxiv.org/abs/2309.05463.

Zhuoyan Li, Hangxiao Zhu, Zhuoran Lu, and Ming Yin. Synthetic data generation with large language models for text classification: Potential and limitations, 2023b. URL https://arxiv.org/abs/2310.07849.

MistralAI. Mistral large 2. https://mistral.ai/news/mistral-large-2407/, 2024. Accessed: 2024-08-13.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. Agentinstruct: Toward generative teaching with agentic flows, 2024. URL https://arxiv.org/abs/2407.03502.

OpenAI. Gpt-4o system card. https://openai.com/index/gpt-4o-system-card/, 2024a. Accessed: 2024-10-14.

OpenAI. Openai o1 system card. https://openai.com/index/openai-o1-system-card/, 2024b. Accessed: 2024-10-14.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of lrm o1, 2024. URL https://arxiv.org/abs/2410.02162.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating llm generations with a panel of diverse models, 2024. URL https://arxiv.org/abs/2404.18796.

Siyuan Wang, Zhuohan Long, Zhihao Fan, Zhongyu Wei, and Xuanjing Huang. Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation, 2024. URL https://arxiv.org/abs/2402.11443.

Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. Top leaderboard ranking = top coding proficiency, always? evoeval: Evolving coding benchmarks via llm, 2024a. URL https://arxiv.org/abs/2403.19114.

Congying Xia, Chen Xing, Jiangshu Du, Xinyi Yang, Yihao Feng, Ran Xu, Wenpeng Yin, and Caiming Xiong. Fofo: A benchmark to evaluate llms' format-following capability, 2024b. URL https://arxiv.org/abs/2402.18667.

Shunyu Yao, Howard Chen, Austin W. Hanjie, Runzhe Yang, and Karthik Narasimhan. Collie: Systematic construction of constrained text generation tasks, 2023. URL https://arxiv.org/abs/2307.08689.

Xiaohan Yuan, Jinfeng Li, Dongxia Wang, Yuefeng Chen, Xiaofeng Mao, Longtao Huang, Hui Xue, Wenhai Wang, Kui Ren, and Jingyi Wang. S-eval: Automatic and adaptive test generation for benchmarking safety evaluation of large language models, 2024. URL https://arxiv.org/abs/2405.14191.

Mert Yuksekgonul, Varun Chandrasekaran, Erik Jones, Suriya Gunasekar, Ranjita Naik, Hamid Palangi, Ece Kamar, and Besmira Nushi. Attention satisfies: A constraint-satisfaction lens on factual errors of language models. *arXiv preprint arXiv:2309.15098*, 2023.

Andy K Zhang, Kevin Klyman, Yifan Mai, Yoav Levine, Yian Zhang, Rishi Bommasani, and Percy Liang. Language model developers should report train-test overlap, 2024a. URL https://arxiv.org/abs/2410.08385.

Jieyu Zhang, Weikai Huang, Zixian Ma, Oscar Michel, Dong He, Tanmay Gupta, Wei-Chiu Ma, Ali Farhadi, Aniruddha Kembhavi, and Ranjay Krishna. Task me anything, 2024b. URL https://arxiv.org/abs/2406.11775.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. Natural plan: Benchmarking llms on natural language planning, 2024. URL https://arxiv.org/abs/2406.04520.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

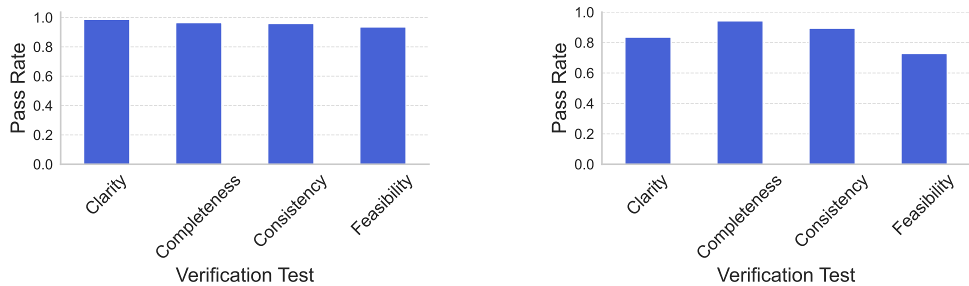
Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. Dyval: Dynamic evaluation of large language models for reasoning tasks, 2024. URL https://arxiv.org/abs/2309.17167.

APPENDIX

A V-AGENT'S PASS RATES

Fig. 5a and Fig. 5b show the pass rate for each verification test over all instances generated by G-AGENT for BA-CALENDAR and BA-TEXT respectively.

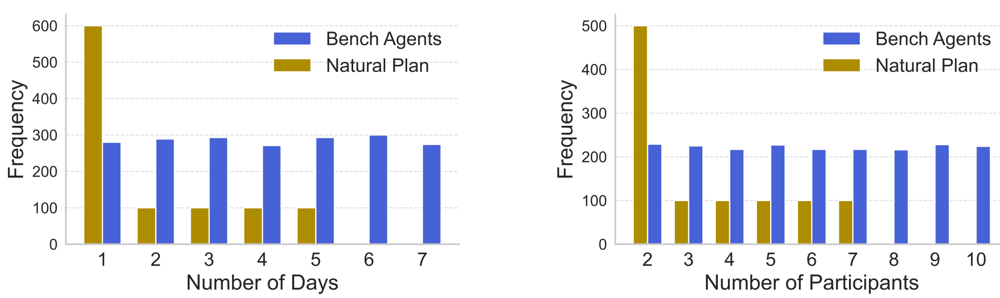


(a) BA-CALENDAR

(b) BA-TEXT

Figure 5: Pass rate for verification checks.

B BA-CALENDAR'S PARAMETER COVERAGE & COMPARISON TO NATURALPLAN



(a) Coverage of Number of Days

(b) Coverage of Number of Participants

Figure 6: Coverage Metrics for Calendar Scheduling Parameters.

For BA-CALENDAR, we may compare to an existing dataset: NATURALPLAN Zheng et al. (2024). NATURALPLAN only includes parameters as metadata, and not constraints. Thus, for the following analysis we extract a set of constraints in each NATURALPLAN prompt using GPT-4o. Figures 6a and 6b demonstrate that BENCHAGENTS has increased coverage compared to NATURALPLAN with respect to the number of days and number of participants parameters. In Figure 7, we can see that both BENCHAGENTS and NATURALPLAN share similar constraints, however, the buffer time constraint is a novel addition by our P-AGENT.

C BA-TEXT'S PARAMETER COVERAGE

In Figure 8, we observe a relatively uniform distribution across constraints. The increased frequency of the positive constraint may be explained by the sampling function of the G-AGENT, whereas, the reduced frequency of positional and sequencing constraints may be explained by the increased likelihood of these two constraints conflicting and so being filtered out by the V-AGENT.

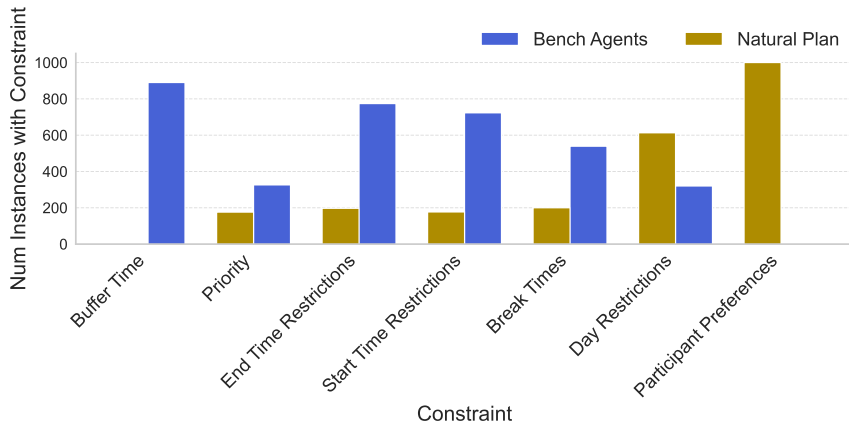


Figure 7: Coverage of Calendar Scheduling Constraints.

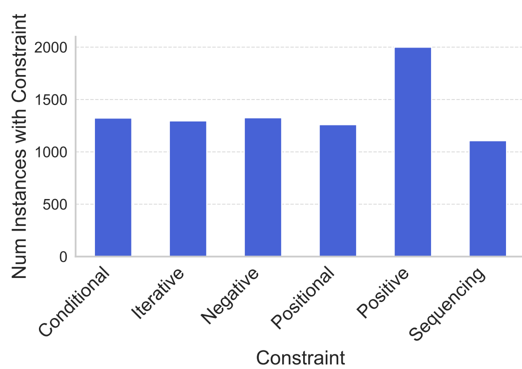


Figure 8: Coverage of Text Generation Constraints.

D MORE DETAILS ON HUMAN ANNOTATION EXPERIMENT FROM § 5.2

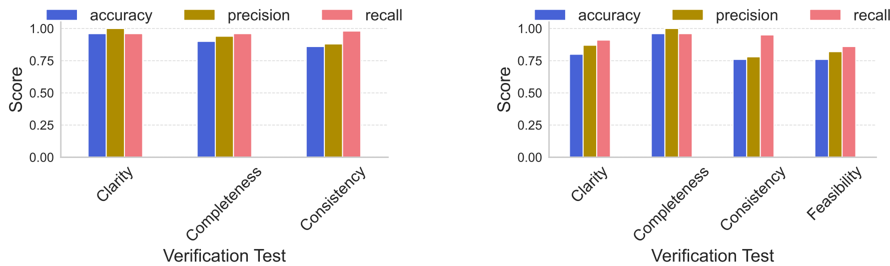
In both the human assessment of the V-AGENT and E-AGENT, annotators were given instructions to annotate each model-based quality check or evaluation metric as either True or False based on the same criteria as output by the P-AGENT. For the human assessment of the V-AGENT, these were clarity, completeness, consistency (and feasibility for BA-TEXT). For the human assessment of the E-AGENT, definitions of each constraint were given to the annotators in line with the agent configuration and P-AGENT outputs.

D.1 HUMAN ASSESSMENT OF V-AGENT

Fig. 9a and 9b report accuracy, precision and recall for V-AGENT and human annotated ground truths.

D.2 HUMAN ASSESSMENT OF E-AGENT

We perform a human assessment of the quality of the model-based evaluations for BA-TEXT. In this study, we take 20 task instances from the G-AGENT after filtering by the V-AGENT, leaving only high quality task instances. For each task instance, we acquire two human annotations for each evaluation test. For the ground truth, we take the worst of both annotations again and our predicted



(a) BA-CALENDAR

(b) BA-TEXT

Figure 9: Human annotator and V-AGENT scores.

values are obtained by executing the model-based evaluation tests from the E-AGENT on GPT-4o solutions. We report accuracy in Table 10 and precision/recall are reported in Fig. 11. Finally, we note that the use of an evaluation agent for generative tasks is consistent with prior work Arabzadeh et al. (2024).

	BA-TEXT
Topic	1.00
Positive	0.85
Negative	0.80
Positional	0.90
Sequencing	0.80
Conditional	0.75
Iterative	0.70

Figure 10: Accuracy for E-AGENT model-based checks.

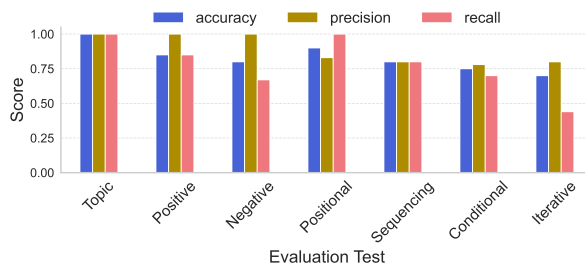


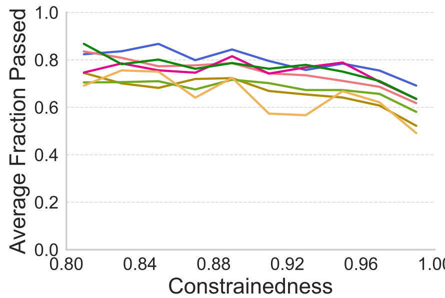
Figure 11: Human annotator and E-AGENT scores for BA-TEXT.

E EXPERIMENTAL SETTINGS

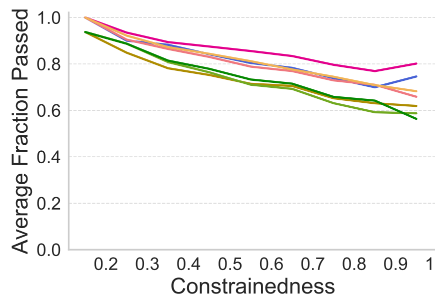
To evaluate each test instance, we perform zero-shot inference with the task prompt, with temperature 0, `top_p` 0.95 and `max_tokens` 2000. The purported solution is then evaluated under the evaluation criteria provided by the E-AGENT. For each test instance, we obtain a set of evaluation test results for constraint satisfaction.

F ADDITIONAL RESULTS ON MODEL ANALYSIS (§ 6)

Here, we present some additional results from our model analysis in § 6. Fig.12 reports the average fraction passed with increasing constrainedness for BA-CALENDAR and BA-TEXT. Further, Fig.13 shows the average pass all with increasing parameter ranges for the number of days and number of participants parameters in BA-CALENDAR.

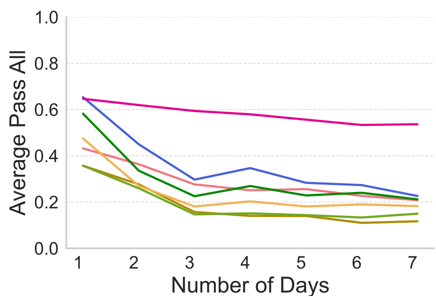


(a) BA-CALENDAR (Fraction Passed)

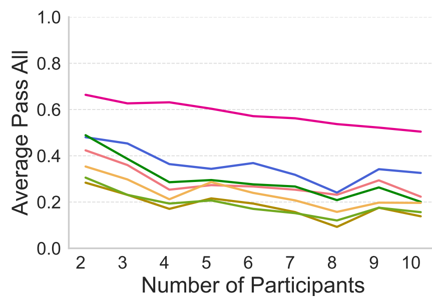


(b) BA-TEXT (Fraction Passed)

Figure 12: Average Fraction Passed with Increasing Constrainedness.



(a) BA-CALENDAR



(b) BA-CALENDAR

Figure 13: Model performance with increasing availability schedule complexity.

G QUANTIFYING DEVELOPER FEEDBACK

Dataset	G-AGENT		V-AGENT		E-AGENT	
	Plan	Code	Plan	Code	Plan	Code
BA-CALENDAR	0.34 (827)	0.14 (11,491)	0.41 (911)	0.34 (11,399)	0.0 (608)	0.25 (17,449)
BA-TEXT	0.02 (3,093)	0.15 (6,145)	0.15 (687)	0.05 (6,276)	0.25 (700)	0.08 (7,687)

Table 4: Normalized Levenshtein distance (max number of characters) between model generations pre- and post- DIL feedback.

From § 3, recall that DIL feedback can utilize the outputs of different agents to refine generation. To quantify the amount of developer-in-the-loop feedback, we calculate the Levenshtein distance on the natural language plans generated by P-AGENT nd the code generated by the respective G-AGENT V-AGENT and E-AGENT In each case, we calculate the distance between the agent-generated plan or code and the plan or code *after* developer-in-the-loop edits and normalize with the maximum number of characters.

Observations: Table 4 reports the normalized distance and the maximum number of characters for each dataset. We observe that BA-CALENDAR requires substantially more characters overall and proportionally more developer-in-the-loop feedback compared to BA-TEXT. We pose that this is because more intervention is needed to ensure the correct parsing of logical conditions verified and evaluated using programmatic tests; constraints in BA-TEXT are not as challenging to parse. This in turn results in better data generated for BA-CALENDAR (§ 5.1).

H BA-CALENDAR

H.1 AGENT CONFIGURATION

Figure 14 displays the task description for BA-CALENDAR from the agent configuration. For the full agent configuration used for BA-CALENDAR, see https://anonymous.4open.science/r/BenchAgents-752D/configs/task/calendar.yaml.

```
Questions will include an availability schedule for participants and additional
constraints to increase difficulty.
The goal is to find one day and common time slot when all participants are available.
Schedules will be provided as a dictionary with participants as keys and availability
schedules as values.
Each schedule will be a dictionary with days of the week as keys and a list of
time blocks as values.
For example:
availability = {
  "p1": {
"Monday": [9:00-12:00, 14:00-17:00],
"Tuesday": [10:00-15:00]
  },
  "p2": {
"Monday": [15:00-18:00],
"Tuesday": [09:00-12:00, 14:00-17:00]
  },
  "p3": {
"Monday": [10:00-11:00, 13:00-16:00, 18:00-19:00],
"Tuesday": [11:00-14:00]
  }
}
A time block will be a string in the format "start_time-end_time" where a participant
is available. A time slot refers to a single time block of length meeting duration.
Granularity refers to the start and end times we consider for blocks. For example,
if granularity is 30 minutes, we consider 9:00, 9:30, 10:00, etc.
Scheduling parameters refers to the parameters used to generate availability schedules
such as Minium length of time block etc.
Constraints refer to additional conditions that must be met for any correct solution
slot such as meeting duration, etc.
Constraints will always have a default value of None, False or 0.
For evaluation, solutions will be in the format:
"[day] [start_time]-[end_time]" or "No common time slot available".
```

Figure 14: BA-CALENDAR task description from its agent config

H.2 DATA GENERATION WITH G-AGENT

As described in § 3.1, P-AGENT takes the agent configuration (see Appendix H.1) as input and generates a plan for data generation (see Appendix H.4). Note, that the DIL may update the plan. The plan includes a list of parameters and constraints, along with the range of values each may take. Tables 5 and 6 contains the full list of parameters and constraints from the plan along with whether each was provided by the agent configuration, LLM or DIL.

The G-AGENT takes the agent configuration and data generation plan as input and writes the code to initialise the data generation procedure. The data generation procedure is initialised as follows. First, the G-AGENT writes functions, `sample_parameters()` and

Parameter	Description	Provided By
Min Length Time Blocks	Minimum length of time blocks per participant per day	Agent Config
Number of Days	Number of days in availability schedule	Model
Number of Participants	Number of participants in availability schedule	Model
Max Length Time Blocks	Maximum length of time blocks per participant per day	Model
Earliest start time	Earliest time in availability schedule	Model
Latest end time	Latest time in availability schedule	Model
Min Number Time Blocks	Minimum number of time blocks per participant per day	DIL
Max Number Time Blocks	Maximum number of time blocks per participant per day	DIL

Table 5: Parameters for BA-CALENDAR.

Constraint	Description	Provided By
Meeting Duration	Duration of solution meeting	Agent Config
Availability	Participants must be available for solution meeting	Agent Config
Buffer Time	Buffer time before and after solution meeting	Agent Config
Weekday Only	Solution meeting must be on a weekday	Model
Morning Time Restriction	Solution must be after a given start	Model
Evening Time Restriction	Solution must be before a given end	Model
Priority	Solution must be the first available slot	DIL
Specific Times	Solution must fall outside of specific time slot	DIL

Table 6: Constraints for BA-CALENDAR.

`sample_constraints(parameters)`, for sampling parameters and constraints. Then, the G-AGENT writes a function `sample_answer(constraints)` for sampling a candidate answer given the constraints. Next, the G-AGENT writes a function `sample_availability(parameters, candidate_answer)` for sampling availability schedules given the parameters and candidate answer.

During data generation, G-AGENT first runs the code for sampling parameters and constraints. Then, G-AGENT sequentially runs the code for sampling a candidate answer and availability. Finally, the prompt is generated by G-AGENT with constraints and availability schedules in-context. The instance is a triple (`prompt, parameters, constraints`). This procedure is repeated for every instance in the dataset. An example prompt generated for BA-CALENDAR by G-AGENT is given in Fig. 15.

```
Given the following availability schedules for participants, find a common time
slot for a meeting that lasts 60 minutes. Additionally,
ensure there is a buffer time of 5 minutes before and after
the meeting.

Availability:
p1:
Monday: 07:00-08:30, 09:30-12:30, 13:15-14:00, 15:00-17:15, 17:45-18:00
Tuesday: 07:00-11:45, 12:15-16:00, 16:45-18:00
Wednesday: 07:00-09:15, 09:45-11:45, 12:30-18:00
p2:
Monday: 07:00-07:15, 08:15-12:30, 13:15-15:00, 16:00-18:00
Tuesday: 07:00-10:15, 10:45-16:00, 17:00-18:00
Wednesday: 07:00-08:00, 09:00-12:30, 13:30-18:00

What is the common time slot for the meeting?
```

Figure 15: BA-CALENDAR example prompt from generated instance

H.3 BA-CALENDAR: QUALITY CHECKS & EVALUATION METRICS

Tables 7 and 8 contains the full list of quality checks and evaluation metrics from the P-AGENT along with whether each was provided by the agent configuration, LLM or DIL.

Quality Check	Type	Provided By
Clarity	Model-based	DIL
Consistency	Model-based	Model
Constrainedness	Programmatic	Agent Config
Completeness	Model-based	Agent Config
Feasibility	Programmatic	Agent Config

Table 7: Quality Checks for BA-CALENDAR.

Evaluation Metric	Type	Provided By
Availability	Programmatic	Agent Config
Meeting Duration	Programmatic	Model
Buffer Time	Programmatic	Model
Weekdays Only	Programmatic	Model
Time Restrictions	Programmatic	Model
Priority	Programmatic	Model
Specific Times	Programmatic	Model

Table 8: Evaluation Metrics for BA-CALENDAR.

H.4 P-AGENT PLAN

It can be found in Figure 16.

```
Parameters:
- Minimum length of time block: 15, 30, 45, 60 minutes
- Maximum length of time block: 60, 90, 120, 180, 240 minutes
- Number of participants: 2-10
- Number of days in the schedule: 1-7
- Maximum number of time blocks per day: 1-5
- Minimum number of time blocks per day: 1-5
- Earliest start time: 6:00, 7:00, 8:00, 9:00
- Latest end time: 17:00, 18:00, 19:00, 20:00

Constraints:
- Meeting duration: 15, 30, 45, 60, 90, 120 minutes
- Buffer time before and after meeting: None, 5, 10, 15, 30 minutes
- No meetings on weekends: True, False
- No meetings before: None, 8:00, 9:00, 10:00
- No meetings after: None, 17:00, 18:00, 19:00
- High priority meeting (must be scheduled in first available slot): True, False
- No meetings during specific times: None, 12:00-13:00, 16:00-17:00
```

Figure 16: P-AGENT Plan post DIL feedback for BA-CALENDAR.

I BA-TEXT

I.1 AGENT CONFIGURATION

Figure 17 shows the task description for BA-TEXT from the agent configuration. For the full agent configuration used for BA-TEXT, see https://anonymous.4open.science/r/BenchAgents-752D/configs/task/text.yaml.

```
The task is open-ended text generation under constraints.
The goal is to generate text that meets the constraints.

Here are some settings in which we will base our task instances.

Chef
Roles: amateur, professional
Task: recipe

Teacher
Roles: primary school, high school
Task: lesson plan

Student
Roles: high school, university
Task: essay

Researcher
Roles: machine learning, biology
Task: scientific article

Engineer
Roles: mechanical, structural
Task: technical report

We will add constraints to each settings grouped into categories.
Here are the definitions of each category:
Positive: apply a constraint to include something in the text
generation
Negative: apply a constraint to exclude something in the text generation
Positional: apply a constraint at an absolute or relative position in the
text generation
Sequencing: apply multiple constraints in a specific order
Iterative: apply a constraint multiple times for items in a list
Conditional: apply a constraint if the text generation meets a condition,
else apply another constraint or do nothing. we can have multiple conditions
and constraints such as if condition then apply constraint, else if condition
apply another constraint, else apply another constraint. do not make conditional
constraints that read if condition apply constraint or another constraint
as this is ambiguous.

All constraints should be conditioned on the model context only and not
outside knowledge.
No constraints should use outside knowledge.
Constraints should only ask for text based outputs and not ask for figures
or other output types.
Constraints should be based on the model's context and the setting.
```

Figure 17: BA-TEXT Task Description from Agent Config.

I.2 Data Generation with G-Agent

P-Agent takes the agent configuration (see Appendix I.1) as input and generates a plan for data generation (see Appendix I.4). For each *user* parameter and constraint group defined in the task description from the agent configuration, the plan includes an example constraint. The DIL has the option to update these to align the constraint generation with developer preferences, however, we can see from Appendix G Table 4, the normalized edit distance for BA-Text data generation plan is 0.02 and so minimal changes were made. Definitions of parameters (all provided by the agent configuration) may be found in Table 9.

Parameter	Description	Provided By
User	User performing task	Agent Config
Role	Role of user	Agent Config
Task	Text generation task	Agent Config
Number of Constraints	The number of constraints applied in each constraint group	Agent Config

Table 9: Parameters for BA-Text.

Similarly to BA-Calendar, the G-Agent takes the agent configuration and data generation plan as input and writes the code to initialise the data generation procedure. The data generation procedure is initialised as follows. First, the G-Agent writes a function `sample_parameters` for parameter sampling. Note that the parameters here include a number of constraints for each constraint group. Next, the G-Agent writes a function `generate_topic(parameters)` which prompts the LLM to generate a topic grounded on the instance parameters. Then, the G-Agent writes a function `generate_constraints(parameters, topic)`, which prompts a LLM to sequentially generate constraints with all previous constraints in-context. An example prompt generated by G-Agent for BA-Text is given in Fig. 18.

```
You are tasked with writing a scientific article on the topic of underfitting in machine
learning. The article should include a detailed explanation of underfitting, provide at
least one example of a model that commonly experiences underfitting, and discuss methods
to mitigate underfitting in machine learning models.

Ensure that you do not include any references to specific datasets, avoid mentioning
any proprietary machine learning frameworks, and exclude any discussion of overfitting.

If the article discusses linear regression, include a section on the limitations of
linear models in complex datasets. If neural networks are mentioned, provide an example
of underfitting in a deep learning context. If the article includes
a section on data preprocessing, discuss how insufficient data preprocessing can lead
to underfitting.

Additionally, discuss at least three different techniques to address underfitting and
provide multiple examples of underfitting in various
machine learning algorithms."
```

Figure 18: BA-Text example prompt from generated instance

During data generation, the G-Agent first samples parameters and then generates a topic. Next, the G-Agent sequentially generates constraints given the parameters and topic, keeping all previously generated constraints for the instance in-context. Finally, finally, the prompt is generated by G-Agent with constraints and topic in-context. An example of a constraint for each constraint group may be found in Table 10. Note that often the number of constraints for each constraint group is more than one i.e. G-Agent will generate multiple constraints in the same constraint group. The instance is a triple `(prompt, parameters, constraints)`. This procedure is repeated for every instance in the dataset.

Constraint Group	Example	Provided By
Positive	Include at least one type of fresh herb	Model
Negative	Do not include any nuts	Model
Positional	Add salt and pepper after all other ingredients have been mixed	Model
Sequencing	Start with washing and chopping vegetables, then proceed to cooking	Model
Conditional	If the recipe includes meat, ensure it is marinated for at least 30 minutes	Model
Iterative	For each step, specify the cooking time and temperature	Model

Table 10: Example Constraints for BA-TEXT.

I.3 BA-TEXT: QUALITY CHECKS & EVALUATION METRICS

Tables 11 and 12 contain the full list of quality checks and evaluation metrics from the P-AGENT along with whether each was provided by the agent configuration, LLM, or DIL.

Quality Check	Type	Provided By
Clarity	Model-based	Model
Consistency	Model-based	Agent Config
Constrainedness	Programmatic	Agent Config
Completeness	Model-based	Agent Config
Feasibility	Model-based	Model

Table 11: Quality Checks for BA-TEXT.

Evaluation Metric	Type	Provided By
Topic Consistency	Model-based	Agent Config
Conditional Constraints	Model-based	Agent Config
Positive Constraints	Model-based	Model
Negative Constraints	Model-based	Model
Positional Constraints	Model-based	Model
Sequencing Constraints	Model-based	Model
Iterative Constraints	Model-based	Model

Table 12: Evaluation Metrics for BA-TEXT.

I.4 P-AGENT PLAN

It can be found in Figure 19.

```
### Chef
**Roles: amateur, professional**
**Task: recipe**
- **Positive:** Include at least one type of fresh herb.
- **Negative:** Do not include any nuts.
- **Positional:** Add salt and pepper after all other ingredients have been mixed.
- **Conditional:** If the recipe includes meat, ensure it is marinated for at
least 30 minutes; else, if the recipe is vegetarian, include a protein source
like beans or tofu.
- **Sequencing:** Start with washing and chopping vegetables, then proceed to
cooking.
- **Iterative:** For each step, specify the cooking time and temperature.

### Teacher
**Roles: primary school, high school**
**Task: lesson plan**
- **Positive:** Include a hands-on activity.
- **Negative:** Do not use any digital tools.
- **Positional:** Introduce the main topic within the first 10 minutes of the lesson.
- **Conditional:** If the lesson is for primary school, include a story; else, if the
lesson is for high school, include a case study.
- **Sequencing:** Begin with a warm-up exercise, followed by the main lesson, and end
with a review.
- **Iterative:** For each section of the lesson, provide an estimated duration.

### Student
**Roles: high school, university**
**Task: essay**
- **Positive:** Include at least three references to academic sources.
- **Negative:** Do not use first-person pronouns.
- **Positional:** State the thesis statement in the first paragraph.
- **Conditional:** If the essay is argumentative, include a counterargument; else,
if the essay is descriptive, include detailed descriptions.
- **Sequencing:** Start with an introduction, followed by body paragraphs, and end
with a conclusion.
- **Iterative:** For each body paragraph, include a topic sentence and supporting
evidence.

### Researcher
**Roles: machine learning, biology**
**Task: scientific article**
- **Positive:** Include a section on related work.
- **Negative:** Do not use informal language.
- **Positional:** Place the abstract at the beginning of the article.
- **Conditional:** If the research includes experiments, provide a detailed
methodology; else, if it is a review, include a comprehensive literature survey.
- **Sequencing:** Start with an introduction, followed by methods, results, and
discussion.
- **Iterative:** For each figure or table mentioned, provide a brief description
in the text.

### Engineer
**Roles: mechanical, structural**
**Task: technical report**
- **Positive:** Include a section on safety considerations.
- **Negative:** Do not include any speculative statements.
- **Positional:** Place the executive summary at the beginning of the report.
- **Conditional:** If the report includes calculations, provide detailed steps.
- **Sequencing:** Start with an introduction, followed by methodology, results,
and conclusions.
- **Iterative:** For each section, include a summary at the end.
```

Figure 19: P-AGENT Plan Post DIL Feedback for BA-TEXT.