Branch-Solve-Merge Improves Large Language Model Evaluation and Generation

Swarnadeep Saha[*]
UNC Chapel Hill

Omer Levy
Meta

Asli Celikyilmaz
Meta

Mohit Bansal
UNC Chapel Hill

Jason Weston
Meta

Xian Li
Meta

Abstract

Large Language Models (LLMs) are frequently used for multi-faceted language generation and evaluation tasks that involve satisfying intricate user constraints or taking into account multiple aspects and criteria. However, their performance can fall short, due to the model's lack of coherence and inability to plan and decompose the problem. We propose BRANCH-SOLVE-MERGE (BSM), a Large Language Model program (Schlag et al., 2023) for tackling such challenging natural language tasks. It consists of *branch*, *solve*, and *merge* modules that are parameterized with specific prompts to the base LLM. These three modules plan a decomposition of the task into multiple parallel sub-tasks, independently solve them, and fuse the solutions to the sub-tasks. We apply our method to the tasks of LLM response evaluation and constrained text generation and evaluate its effectiveness with multiple LLMs, including Vicuna, LLaMA-2-chat, and GPT-4. BSM improves the evaluation correctness and consistency for each LLM by enhancing human-LLM agreement by up to 26%, reducing length and pairwise position biases by up to 50%, and allowing LLaMA-2-chat to match or outperform GPT-4 on most domains. On a constraint story generation task, BSM improves the coherence of stories while also improving constraint satisfaction by 12%.

1 Introduction

Large Language Models (LLMs) are widely used for various text generation tasks (Radford et al., 2019; Brown et al., 2020; OpenAI, 2023b; Chowdhery et al., 2022; Touvron et al., 2023). It has also become common to employ them as evaluators of such LLM generations in order to assess, critique and improve the outputs (Zheng et al., 2023; Bai et al., 2022b). However, LLMs still struggle with tasks that have intricate requirements like satisfying a set of constraints or meeting objectives that

are, in general, multi-dimensional (e.g., evaluating the quality of generated text against certain diverse criteria). This appears to primarily stem from the model's lack of self-consistency and inability to plan (Yao et al., 2023b; Bubeck et al., 2023). Recent research has tried to mitigate these limitations by developing iterative methods that involve eliciting reasoning, planning, and refinement, but so far they are still considered as open problems (Bai et al., 2022b; Madaan et al., 2023; Ganguli et al., 2023; Yao et al., 2023c; Chen et al., 2023; Li et al., 2023; Huang et al., 2023).

In this work, we propose BRANCH-SOLVE-MERGE (BSM), a decomposition method for solving such multi-faceted natural language tasks. Our approach is an instance of a *Large Language Model program* (Schlag et al., 2023; Dohan et al., 2022) and consists of three modules: *branch*, *solve*, and *merge* that are parameterized with specific prompts to an underlying LLM. Given an arbitrary user task, the 'branch' module generates a solution plan by decomposing the task into multiple *parallel* sub-tasks, where each sub-task is represented by a unique branch, representing different components required to solve the overall problem. The 'solve' module then solves each of these independent sub-problems. Finally, the 'merge' module fuses the solutions to these sub-problems to generate the overall solution. We apply our method to two challenging tasks where LLMs are commonly utilized but their performance still lags behind humans:

- **Evaluation of LLM Outputs (Zheng et al., 2023).** LLMs are now regularly used to perform automatic evaluation of model responses, e.g., to user queries (Dubois et al., 2023). Evaluating LLMs holistically is challenging because of their ability to generate long-form answers to arbitrary user questions (Zheng et al., 2023), the lack of reliability originating from many biases (Zheng et al., 2023; Wu and Aji, 2023; Wang et al., 2023b), and reliance on hand-designed evalua-
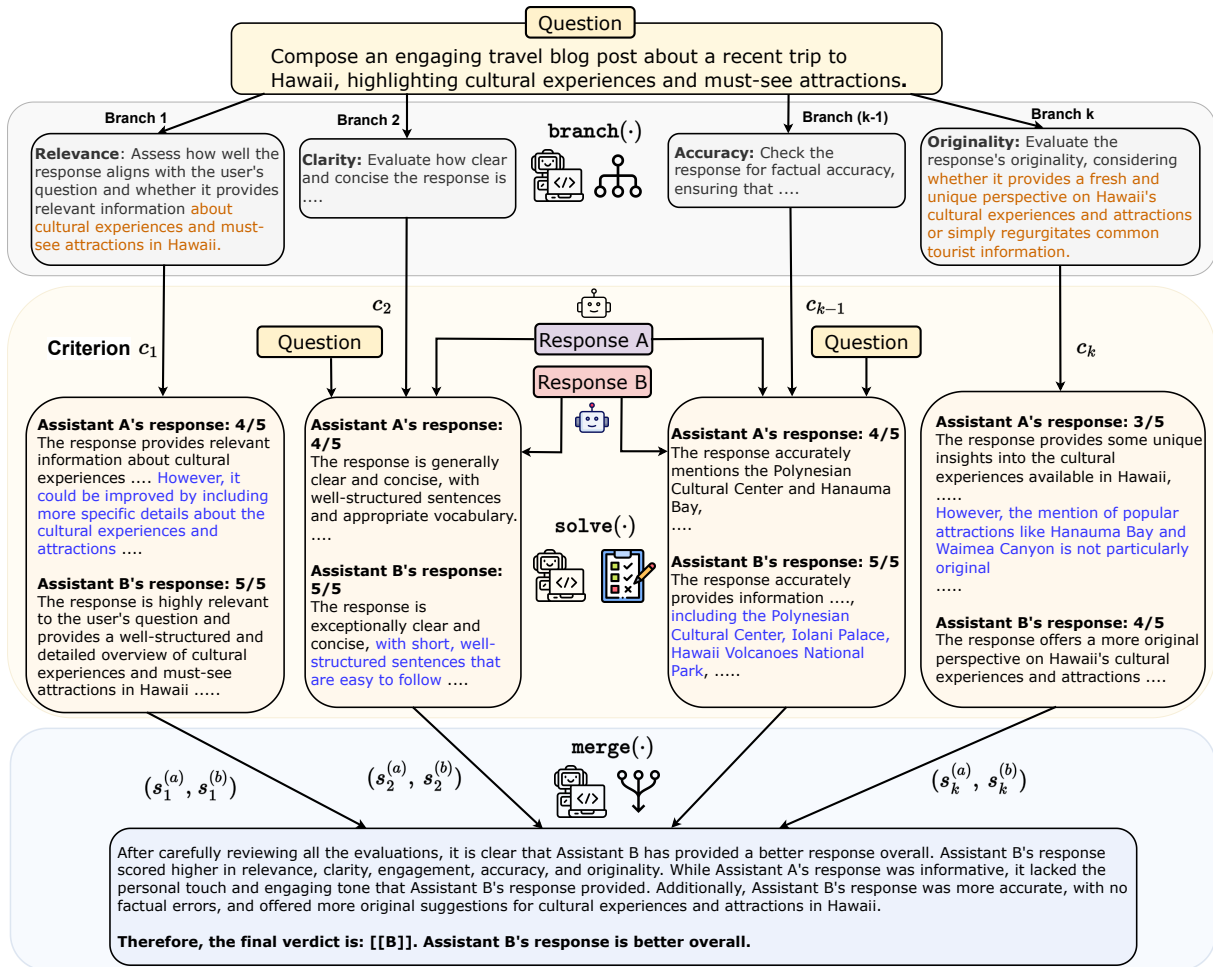

8352

Figure 1: An illustration of BRANCH-SOLVE-MERGE with LLaMA-2-70B-chat for pairwise evaluation of LLM responses. Given a question and two LLM responses A and B, BSM generates a preference judgment. The Branch module conditions on the question to generate a question-specific evaluation plan which in this case consists of different criteria like 'Relevance' to the Hawaii trip topic, 'Clarity', etc. The 'Solve' module evaluates the response pairs for each criteria (branch) independently and the 'Merge' module combines the individual judgments to generate the final verdict, in this case that B is the better response.

tion plans that impact the method's generalization, introducing unintended human biases (Liu et al., 2023; Wu and Aji, 2023). BSM can be applied to this task by each branch assessing different aspects and criteria that require evaluation.[*]

- **Constrained Text Generation.** State-of-the-art LLMs struggle with constrained text generation tasks, e.g., the constraint of writing a story that should include several concepts. Models commonly either violate constraints, or else generate text that is incoherent in order to satisfy these constraints (Bubeck et al., 2023; Yao et al., 2023a). BSM can be applied to this task by each branch writing part of the story satisfying only some of

the constraints, followed by a final merge.

We apply BSM to both these problems, see Fig. 1 and Fig. 3, and evaluate its effectiveness with multiple open-source and black-box LLMs of varying sizes and strengths including LLaMA-2-7B-chat (Touvron et al., 2023), Vicuna-33B (Chiang et al., 2023), LLaMA-2-70B-chat, and GPT-4 (OpenAI, 2023b). BSM significantly improves both tasks, addressing the aforementioned limitations of LLM evaluation and generation:

- BSM improves *correctness* of LLM evaluation. In particular, on the MT-Bench benchmark (Zheng et al., 2023), BSM improves LLM-human agreement for evaluating multi-turn questions belonging to different domains including writing, coding, reasoning, and mathematics. For example, compared to zero-shot prompting and

---

[*]Subsequently, we will refer to the task as 'LLM Evaluation'. In the scope of our study, this will involve the pairwise evaluation of the response quality of two LLM outputs.

self-consistency (Wang et al., 2022) baselines, BSM with LLaMA-2-70B-chat improves LLM-human agreement by up to absolute 26% and even matches or outperforms GPT-4 on many domains. BSM with GPT-4 improves agreement by a further 3% over GPT-4. Overall, these findings point to BSM's ability to evaluate LLM responses to arbitrary user questions from diverse domains and to improve any base LLM as an evaluator.

- BSM also improves the *consistency* of LLM evaluation. It significantly reduces position, length, and self-enhancement biases of LLM-based evaluators. For instance, BSM with LLaMA-2-70B-chat reduces position bias by up to absolute 50%. Importantly, BSM with GPT-4 also improves GPT-4's reliability as an evaluator when evaluating its own responses.

- For the constrained story generation task, BSM generates more coherent stories, which are preferred by a GPT-4 judge a substantial 93% of the time compared to a zero-shot baseline. It also improves constraint satisfaction by 12%.

Overall, BSM provides a framework for planning and task decomposition for addressing challenging multi-faceted language generation and evaluation tasks. As the approach is framed as a generic LLM program, it can be applied to any underlying LM and potentially a wide range of tasks.

2 Related Work

LLM Programs and Decomposing Complex Tasks. LLM programs such as BSM solve complex problems with an algorithm that breaks the problem down into multiple steps and each step is then parameterized with a different prompt to an underlying LLM (Schlag et al., 2023; Dohan et al., 2022; Creswell and Shanahan, 2022). Complex tasks, in general, require task decomposition (Khot et al., 2022) and planning (Yao et al., 2022; Huang et al., 2022; Yao et al., 2023b; Ning et al., 2023). This has motivated a lot of recent work on advanced prompting methods (Khot et al., 2022; Zhou et al., 2022; Wang et al., 2023a; Dua et al., 2022; Saha et al., 2022, 2023; Khot et al., 2021; Gupta and Kembhavi, 2023; Cho et al., 2023). However, most of these works typically focus on reasoning problems (like commonsense, symbolic, or mathematical) that benefit from *sequential* decompositions. We, however, study tasks that benefit from branching into *parallel* decompositions, in particular LLM evaluation and constrained text generation. As well

as being an LLM program, BSM is also an instance of Graph-of-Thoughts (GoT) prompting (Lei et al., 2023; Besta et al., 2023) because the execution trace takes the shape of a graph. GoT defines a wide array of LLM programs, including refining, backtracking and skipping graph nodes, which we do not consider here. Our work develops a specific fixed program, and applies it to the challenging tasks of evaluating or improving language models.

LLM Evaluation. A fundamental challenge with the rapid progress of LLMs is evaluating their capabilities holistically (Chang et al., 2023; Liang et al., 2022). Human evaluation is difficult and expensive (Smith et al., 2022). On the other hand, LLMs, by being trained with RLHF, are shown to exhibit alignment with humans (Ouyang et al., 2022; Bai et al., 2022a). Hence, a standard procedure for comparing and evaluating LLM generations is by utilizing a strong LLM like GPT-4 (Bubeck et al., 2023; OpenAI, 2023a; Dubois et al., 2023; Zhou et al., 2023; Chiang and Lee, 2023; Wang et al., 2023c; Hada et al., 2023; Liu et al., 2023) on different benchmarks (Zhong et al., 2023; Köpf et al., 2023; Zheng et al., 2023). LLM-based evaluators are not fair evaluators (Wang et al., 2023b; Wu and Aji, 2023) and there have been proposals of using multi-agent debate (Chan et al., 2023) or developing wider and deeper LLMs (Zhang et al., 2023). In contrast, BSM improves LLM evaluation through an intuitive and general decomposition-based approach that can be applied on top of any LLM to evaluate responses for a wide range of tasks.

Constrained Text Generation. Recent works evaluate LLMs for their capabilities in the more difficult setting of controllable and constrained text generation (Keskar et al., 2019; Dathathri et al., 2019; Lu et al., 2021, 2022; Lin et al., 2020; Li et al., 2022) and show that even GPT-4 struggles with such planning-based tasks (Bubeck et al., 2023; Madaan et al., 2023; Yao et al., 2023a). We experiment with such a constrained story generation task and show the promise of BSM.

3 BRANCH-SOLVE-MERGE

We first introduce some notation to formally describe BSM. Let p_θ denote an LLM with parameters θ. We also denote $x = x_{1,\cdots,n}$ as a sequence of n tokens, such that $p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i|x_{1,\cdots,i-1})$. BSM is an LLM program that aims to solve complex planning-based tasks with three neural mod-

ules: branch, solve, and merge. Each module is parameterized with unique prompts to the LLM p_θ. The LLM program further defines an algorithm on top of these modules, acting as a controller and invoking a module at each step of the algorithm.

3.1 Components of BRANCH-SOLVE-MERGE

LLM Program. For a given task, BSM defines a controller as an algorithm that lays out the transition logic between the modules. Let us denote the three modules with their functional forms: branch($\cdot$), solve($\cdot$), and merge($\cdot$). Then the program is defined as Prog : $(x, \text{branch}(\cdot), \text{solve}(\cdot), \text{merge}(\cdot)) \to y$, taking as input a task instance x, along with the module implementations and generating an output y.

Branch Module. Given a task, the branch module generates multiple sub-tasks where each sub-task is represented by a unique branch. Branching into sub-problems allows task decomposition such that each part can be solved independently in parallel, at which point the partial solutions are combined. Formally, given a task input x, we define a 'branch' prompt $\text{prompt}_\text{branch}(x)$ that can be wrapped around x with branching instructions and some demonstrations (if available). Conditioning on the prompt, the LLM p_θ generates a set of k sub-problems $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(k)}\}$, where k is referred to as the branching factor. The sub-problems are generated auto-regressively as a sequence of tokens: $X \sim p_\theta(X|\text{prompt}_\text{branch}(x))$. Importantly, the flexibility of our method comes from the fact that for a given problem, the LLM itself decides (generates) the sub-problems and the corresponding branching factor.

Solve Module. The solve module solves the task at hand by generating an output $y^{(i)}$ for a branch task input $x^{(i)}$. Similar to the branch prompt, we define a 'solve' prompt $\text{prompt}_\text{solve}(x^{(i)})$, conditioning on which the LLM generates a solution $y^{(i)} \sim p_\theta(y^{(i)}|\text{prompt}_\text{solve}(x^{(i)}))$ for each branch.

Merge Module. The merge module fuses the solutions to the sub-problems to generate a global solution to the main problem. This is done through a 'merge' prompt $\text{prompt}_\text{merge}(Y)$ that generates a merged solution $y \sim p_\theta(y|\text{prompt}_\text{merge}(Y))$, conditioning on a set of sub-solutions $Y = \{y^{(1)}, y^{(2)}, \cdots, y^{(k)}\}$. Conceptually, the merge module learns an aggregator function that could aggregate a set of values (using an aggregation operator) or fuse pieces of text, depending on the task.

Next, we motivate and conduct case studies of BSM with two challenging NLP tasks: LLM evaluation and constrained generation.

3.2 BSM: Case Study with LLM Evaluation

Task Description. We consider the task of evaluating LLM-based chat assistants. Formally, given an open-ended question and a pair of responses from two LLM agents, the task requires producing a preference judgement of which response is better or if it is a tie (see Fig. 1). Evaluating LLM responses is challenging for many reasons:

1. **Long-form answers to arbitrary questions.** With the goal of providing a general-purpose assistant, the user asks arbitrary questions from any domain, and the LLM responds with long-form answers (Zheng et al., 2023). Based on the initial model response, the user can ask follow-up questions. Depending on the type of question, the evaluation process must consider the intent of the question, what is expected from an ideal response, and what criteria to evaluate on.

2. **LLM evaluators are prone to biases.** LLM-based evaluators are not reliable and are prone to different biases including (a) *Position Bias*: evaluation changes based on the encoding order of the responses, (b) *Length Bias*: tendency to favor longer responses, (c) *Self-enhancement Bias*: the LLM-evaluator favoring its own responses (Zheng et al., 2023).

3. **GPT-4 as evaluator is expensive.** While API-based models like GPT-4 are fairly good evaluators (Zheng et al., 2023), these models are proprietary and charge users per token generated. Current open-source alternatives correlate less well with humans and are much more susceptible to the aforementioned biases.

4. **Hand-designing evaluation plans is not scalable.** A robust evaluator should generalize well, capable of evaluating responses to arbitrary questions and hence, hand-designing the evaluation plan for every task is not desirable (Liu et al., 2023). For example, see Fig. 1, where evaluating responses to a 'writing' question requires considering factors like 'Relevance', 'Clarity', etc whereas if the question is a 'coding' question (see Fig. 2 in the Appendix), one should evaluate for 'Code Correctness', 'Code Readability', etc.

Hence, given the multi-faceted nature of this evaluation task, we develop a version of BSM, as

described below. For this study, we focus on evaluating two-turn conversational questions although our method is generally applicable for any number of turns. Let us denote the first question as q_1 and the follow-up question as q_2. Let the responses from the two LLMs A and B be $r_1^{(A)}$ and $r_1^{(B)}$ for q_1, and $r_2^{(A)}$ and $r_2^{(B)}$ for q_2.

Branch Module for LLM Evaluation. It generates an evaluation plan i.e., a set of evaluation criteria that the response will be evaluated against. The branch module only conditions on the input question and for turn-1 questions, is defined as as $\mathsf{branch}(q_1)$, while for turn-2 questions, it conditions on both turn-1 and turn-2 questions, represented as $\mathsf{branch}(q_1, q_2)$. The output is a set of evaluation criteria, $\mathsf{branch}(q) \rightarrow \{c_i\}_{i=1}^k$, where each c_i is the title of the criterion (e.g., 'Relevance') and a short description of how to evaluate for it (e.g., 'Assess how well the response aligns with the user's question ... and must-see attractions in Hawaii.'). See Fig. 1 and 2 for examples of generated branches for different questions.

Solve Module for LLM Evaluation. It compares and evaluates the responses based on a specific criterion. The output of the evaluation is a pair of scores (within a specified range, according to the solving instruction, e.g., 1-5) for each of the responses. For example, given an evaluation criterion c, we denote the solve module for a question q as: $\mathsf{solve}(q, r_1^{(A)}, r_1^{(B)}, c) \rightarrow (s^{(A)}, s^{(B)})$, where $s^{(A)}$ and $s^{(B)}$ are the evaluation scores assigned to the two assistant responses. Note that the solve module is not symmetric i.e., the encoding order of the two responses is important (and we address this below in our LLM program). The module additionally generates explanations along with the scores. Fig. 1 shows example generations from the solve module with a LLaMA-2-70B-chat model.

Merge Module for LLM Evaluation. We develop two variants of the merge module. A simple non-neural variant sums up the scores across all branches. We also develop a neural LLM variant that conditions on the individual evaluations and generates the final verdict with a model-decided aggregation strategy, denoted as $\mathsf{merge}(q, \{c_i\}_{i=1}^k, \{s_i^{(A)}\}_{i=1}^k, \{s_i^{(B)}\}_{i=1}^k) \rightarrow y$, where the evaluation criteria $\{c_i\}_{i=1}^k$ are the outputs of the branch module and $s_i^{(A)}$ and $s_i^{(B)}$ are the criterion-wise evaluations (scores and explanations) of the two assistant responses generated from the solve module. The final verdict is $y \in \{A, B, tie\}$.

LLM Program for LLM Evaluation. The overall LLM program pseudocode is given in Algorithm 1. To account for position bias, the program executes two independent runs of BSM by swapping the encoding order of the responses in the 'solve' module. The final judgment is either 'A' or 'B' if and only if the judgement is consistent for both orders, otherwise it is a 'tie'.

3.3 BSM: Case Study with Constrained Gen

Task Description. Our next case study shows the general applicability of BSM by applying it to a completely different task, that of LLM generation. We consider a constrained story generation task – given a set of concepts l, the task is to generate a coherent story y by including all concepts in it (see Fig. 3 in the Appendix). When the number of concepts is large, LLMs tend to either leave out some concepts or generate text that is incoherent. The task requires composition incorporating the various constraints.

Branch Module for Constrained Generation. The branch module $\mathsf{branch}(l) \rightarrow (l_1, l_2, t)$ proposes a story generation plan, consisting of (1) two subsets of concepts l_1 and l_2 and (2) a story topic t. The two subsets represent sub-problems of the original task with a smaller number of concepts. The story topic ensures that all sub-stories generated as part of BSM belong to the same topic.

Solve Module for Constrained Generation. The solve module $\mathsf{solve}(l_i, t) \rightarrow y_i$ conditions on a subset of concepts l_i and the story topic t to generate a story y_i on that topic, while also including all concepts in l_i. Intuitively, 'solving' the constrained generation task is easier with a smaller number of concepts.

Merge Module for Constrained Generation. The merge module $\mathsf{merge}(y_1, y_2) \rightarrow y$ conditions on two intermediate stories and fuses them together to generate the final story y. Since both intermediate stories belong to the same high-level topic, the fusion can lead to a final coherent story. Overall, BSM ensures better constraint satisfaction by solving sub-problems and maintains coherency through a top-level plan that includes a story topic.

4 Experiments

4.1 Large Language Model Evaluation

4.1.1 Experimental Setup

Dataset. We experiment with the MT-Bench dataset, that evaluates LLMs as judges of other LLM's responses when acting as helpful AI assistants in multi-turn conversations (Zheng et al., 2023). It consists of instructions from 8 diverse domains e.g., writing, reasoning, math, coding, etc.

Evaluation Metrics. We evaluate BSM (and baselines) using the following four metrics.

- **LLM-Human Agreement (Ag).** Following past work (Zheng et al., 2023), we report LLM-human agreement $\in [0, 1]$ individually for turn-1 and turn-2 questions, and their combination.
- **Position Bias (PB).** To evaluate whether BSM helps reduce the consistency problem with LLM-based evaluators, we report PB, which is the fraction of samples where the judgment changes based on the encoding order of the responses.
- **Length Bias (LB).** We measure LB as the fraction of samples where humans prefer the shorter response but the evaluator model does not. In other words, we compute how often an evaluator chooses the longer response when according to human preference, it should not.
- **Self-enhancement Bias (SB).** SB refers to an evaluator model preferring its own responses. Evaluating this bias in isolation is challenging because knowing when the model is choosing its own response because of this bias and not for another reason is an interpretability question. However, the question we are interested in studying here is the following: *When an LLM is evaluating its own responses (which is a common phenomenon when using LLMs as evaluators), does BSM lead to better and more reliable evaluation?* We measure this by considering the following setting. We use GPT-4 as the base judge model and consider the subset of samples from the MT-Bench benchmark where one of the responses is *also* generated by GPT-4. If BSM with GPT-4 improves human-agreement for this subset of samples, it suggests that *even* in scenarios where model A is judging its own outputs, BSM (with model A) leads to a better evaluator. While this does not necessarily compute whether an evaluator has less SB, it does verify whether the evaluator model correlates better with humans even when it is evaluating its own responses.

While multiple past works have highlighted the importance of these biases (Zheng et al., 2023; Wu and Aji, 2023), we measure all of them with concrete metrics within the same evaluation framework. Conceptually, 'Ag' evaluates *correctness* while 'PB' for example evaluates *consistency* of LLM-based evaluators. These are complementary aspects and an ideal evaluator should perform well in all metrics for it to be reliably used.

Implementation Details. We develop BSM on top of multiple LLMs of varying scales and capabilities: LLaMA-2-7B-chat, Vicuna-33B, LLaMA-2-70B-chat, and GPT-4. We implement all modules zero-shot, providing only module-specific instructions and assuming no access to demonstrations of how to branch, solve, or merge.

Baselines. We compare our method, BSM, to (1) **two variants of zero-shot prompting with the same LLM**: a relative evaluator, that directly generates a preference judgment and an absolute evaluator, that generates two scores for the two responses and then the final preference is determined based on the higher score, (2) **plan&solve prompting** (Wang et al., 2023a), which plans (i.e., generates evaluation criteria) but instead of solving them independently, solves all branches together in one LLM call, (3) **self-consistency** (Wang et al., 2022), which samples multiple evaluations from the prompted LLM (with temperature 0.7) and chooses the majority vote as the final judgment. For fair comparison, self-consistency samples the same number of generations as the branching factor in BSM. We also note that self-consistency is a simple special case of BSM, where the branch module spawns multiple instances of the *same* underlying problem (instead of sub-problems), solves them by sampling different solutions, and the merging operator is a majority vote. Refer to Appendix A for more details about the dataset, implementation, and baselines.

4.1.2 Main Results

BSM improves LLM-human agreement and reduces biases. Table 1 evaluates the efficacy of BSM with LLaMA-2-70B-chat as the base LLM, specifically focusing on the 'writing' category of questions from the MT-Bench benchmark. We report our main findings below.

- **Overall agreement.** We find that BSM improves LLM-human agreement for both turn-1 and turn-2 questions, compared to all baselines. In partic-

Method	Overall				Turn-1			Turn-2	
	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓
Zero-shot (Relative)	0.43	51.66	54.88	0.53	42.66	50.00	0.34	60.66	59.42
Zero-shot (Absolute)	0.45	30.00	48.87	0.56	19.33	43.75	0.34	40.66	53.62
Plan&Solve	0.43	43.00	54.13	0.43	42.00	51.56	0.43	44.00	56.52
Self-Consistency	0.52	35.66	48.12	0.57	32.00	45.31	0.47	39.33	50.72
BSM	**0.55**	**17.33**	**39.09**	**0.60**	**14.66**	**39.46**	**0.50**	**20.00**	**39.13**

Table 1: Comparison of zero-shot LLM evaluators (Relative and Absolute), Plan&Solve, Self-Consistency, and BSM on the 'writing' questions in the MT-Bench dataset. All methods use LLaMA-2-70B-chat as the base LLM. We report LLM-Human Agreement (Ag), Position Bias (PB), and Length Bias (LB) for turn-1 and turn-2 questions overall, and individually. BSM improves agreement scores, and reduces position and length biases.

	Ag↑	PB↓	LB↓
Zero-shot (w/ GPT-4)	0.51	**6.33**	36.36
BSM (w/ GPT-4)	**0.54**	7.33	**34.54**

Table 2: BSM leads to less self-enhancement bias. BSM obtains better agreement for the fraction of samples where one of the responses is also generated by GPT-4.

Method	Ag↑	PB↓	LB↓
Zero-shot (w/ LLaMA-2-7B-chat)	0.39	62.33	54.88
BSM (w/ LLaMA-2-7B-chat)	**0.41**	**48.33**	**53.38**
Zero-shot (w/ Vicuna-33B)	0.51	30.66	48.12
BSM (w/ Vicuna-33B)	**0.56**	**20.00**	**42.85**
Zero-shot (w/ LLaMA-2-70B-chat)	0.43	51.66	54.88
BSM (w/ LLaMA-2-70B-chat)	**0.55**	**17.33**	**39.09**
Zero-shot (w/ GPT-4)	0.59	17.33	39.09
BSM (w/ GPT-4)	**0.62**	**17.00**	**36.84**

Table 3: Comparison of zero-shot evaluation and BSM on 'writing' questions with different base LLM evaluators. BSM improves agreement for all models and reduces biases for all models except GPT-4.

ular, it obtains up to 12% absolute improvement over Plan&Solve, which specifically shows the utility of branching into and solving *independent* sub-problems. BSM also outperforms Self-Consistency. As noted earlier, Self-Consistency is a special case of BSM. This result is noteworthy because both approaches leverage similar amounts of compute in generating multiple solutions – but branching and solving the differing sub-problems provides superior results to solving the same problem multiple times.

- **Turn-1 versus Turn-2 questions.** Evaluating turn-2 questions is harder because it requires additional contextualization of the responses for the turn-1 question. This is also reflected in all baseline methods (except for plan&solve) exhibiting lower turn-2 agreement scores (e.g., zero-shot results drop from 0.53 in turn-1 to 0.34 in turn-2). BSM shows that a decomposition approach that generates an evaluation plan is particularly helpful for evaluating long context questions, resulting in more improvements for turn-2 questions (e.g., up to 16% improvement). An illustration is shown in Fig. 2, in which for the turn-2 question, the model generates 'Adherence to Instructions' as the first criterion to evaluate.

- **Position and Length Bias Reduction.** On top of improving LLM-human agreement, BSM helps reduce critical biases with LLM-based evaluators (e.g., up to 34% reduction in PB). This is a direct consequence of BSM's task decomposition that helps reduce inconsistencies in evaluation.

BSM's reduction in LB could be attributed to the following: when an evaluator branches into different criteria, if 'length' is indeed one of the criteria that the responses should be evaluated against, it only counts as a single branch (i.e., one sub-problem) of the overall evaluation and hence, branching allows the model to explicitly evaluate for other criteria, beyond just length.

- **Self-enhancement Bias Reduction.** Table 2 evaluates self-enhancement bias by comparing BSM (with zero-shot GPT-4) for the samples where one of the responses is also generated by GPT-4. We observe a 3% better correlation with humans, suggesting that BSM improves evaluation even when the LLM judges its own outputs.

BSM not only leads to an improvement in overall LLM-human agreement (as per the 'Ag' metric) but also on the fraction of samples where one response is generated by the same evaluator LLM (as per the 'SB' metric), thus pointing to its robustness as an evaluation method. In summary, BSM improves both *correctness* and *consistency* of LLM-based evaluators.

BSM improves upon all zero-shot base LLMs. We demonstrate the generalizability of BSM as an LLM program by implementing it on top of four dif-

Method	Coding			Reasoning			Math		
	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓
Zero-shot (w/ LLaMA-2-70B-c)	0.47	52.33	51.32	0.47	38.00	48.75	0.52	45.66	50.56
BSM (w/ LLaMA-2-70B-c)	**0.61**	**25.66**	**42.47**	**0.57**	**20.33**	**46.25**	**0.64**	**17.66**	**34.83**
GPT-4	0.61	19.66	38.93	0.64	22.66	53.75	0.62	19.00	39.32

Table 4: Reference-based LLM evaluation for 'Coding', 'Reasoning', and 'Math' question categories of MT-Bench. BSM improves reference-based evaluations and for math, outperforms GPT-4.

Domain	Method	Overall			Turn-1			Turn-2		
		Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓
Roleplay	Zero-shot (w/ LLaMA-2-70B-c)	0.55	29.66	51.67	0.61	30.00	48.14	0.50	29.33	55.88
	BSM (w/ LLaMA-2-70B-c)	**0.61**	**11.00**	**40.26**	**0.66**	**10.66**	**38.27**	**0.56**	**11.33**	**42.64**
	GPT-4	0.64	13.66	43.62	0.65	16.00	45.67	0.63	11.33	41.17
Extraction	Zero-shot (w/ LLaMA-2-70B-c)	0.40	70.66	51.82	0.46	61.33	51.47	0.33	80.00	52.08
	BSM (w/ LLaMA-2-70B-c)	**0.55**	**31.33**	**40.24**	**0.55**	**32.00**	**45.58**	**0.44**	**30.66**	**36.45**
	GPT-4	0.71	15.00	33.53	0.68	13.33	35.29	0.75	16.66	32.29
Stem	Zero-shot (w/ LLaMA-2-70B-c)	0.46	59.33	55.31	0.50	52.66	51.19	0.43	66.00	61.40
	BSM (w/ LLaMA-2-70B-c)	**0.72**	**10.33**	**44.68**	**0.70**	**10.66**	**40.47**	**0.73**	**10.00**	**50.87**
	GPT-4	0.72	13.66	46.80	0.68	16.66	44.04	0.75	10.66	50.87
Humanities	Zero-shot (w/ LLaMA-2-70B-c)	0.46	59.00	45.69	0.51	52.00	49.18	0.41	66.00	43.33
	BSM (w/ LLaMA-2-70B-c)	**0.67**	**18.00**	**36.42**	**0.63**	**18.00**	**39.34**	**0.71**	**18.00**	**34.44**
	GPT-4	0.73	14.00	37.08	0.70	19.33	42.62	0.76	8.66	33.33

Table 5: LLM evaluation for 'Roleplay', 'Extraction', 'Stem', and 'Humanities' question categories of MT-Bench. We compare LLaMA-2-70B-chat BSM with the baseline zero-shot method, and also report GPT-4 results. BSM obtains significant improvements over the LLaMA baseline, and matches or is close to GPT-4 agreement in three of the four domains, while sometimes outperforming GPT-4 in reducing biases.

ferent base LLMs, ranging from LLaMA-2-7B to GPT-4. As shown in Table 3, BSM improves agreement with humans for all base LLMs, compared to a zero-shot baseline. Even though zero-shot GPT-4 is the state-of-the-art LLM-based evaluator, applying BSM obtains a further improvement of 3%. Moreover, applying BSM to LLaMA-2-70B-chat makes it competitive with GPT-4 for turn-1 questions. BSM also significantly reduces position and length biases for all models except for GPT-4.

BSM generalizes to reference-based evaluations. We find that BSM also excels at *reference-based* evaluations for complex tasks like in math, reasoning, and coding (Cobbe et al., 2021; Wei et al., 2022). Following past work (Zheng et al., 2023), we evaluate responses for these categories by first generating an answer using GPT-4 and then appending it to the evaluation prompt, which is our baseline in this experiment. For BSM, we then follow a similar recipe by conditioning the 'solve' module on the GPT-4 generated answers. The key assumption here is that these answers are curated once and have limited variations unlike answers for open-ended questions. Table 4 shows that BSM significantly outperforms zero-shot baseline in all

categories (by up to 14% better agreement scores and 27% better position bias in coding questions). On Math, it even outperforms the state-of-the-art GPT-4 evaluator, outperforming on all metrics.

BSM generalizes across further domains. Table 5 shows that BSM is capable of evaluating generations for questions in other categories like 'Roleplay', 'Extraction', 'Stem', and 'Humanities', with similar findings. See Appendix A.2 for details.

Scalability of BSM's branching. One of the core strengths of BSM is its scalability – it uses the same branch prompt (in Fig. 4) for all evaluation domains (e.g., writing, code, reasoning, etc). The prompt only specifies the meaning of a branch for a given task and the LLM is capable of generating its own branches for different domains without any human intervention. We observe almost no overlap between the branch names of coding questions and writing questions. For example, the most occurring 'writing branches' are *Clarity, Relevance, Creativity, Accuracy, Engagement, Coherence, Originality, Completeness, Grammar and Readability, etc* whereas the most occurring 'coding branches' are *Efficiency, Completeness, Accuracy, Correct-*

ness, Code Readability, User Experience, Time Ef-ficiency. Branches for questions belonging to the same domain exhibit more overlap, as measured by the names of the branches. For example, 'Correct-ness' is a branch that is generated for evaluating almost all coding problems; however, their descrip-tions are different and problem-specific (e.g., see Fig. 2 for an example).

4.2 Constrained Text Generation

4.2.1 Experimental Setup

Dataset. Our constrained story generation task is a more challenging variant of a generative com-monsense reasoning task, CommonGen (Lin et al., 2020). While the original task requires generating a single coherent sentence from 3 or 4 concepts, we increase the complexity of the task by having the model generate a concise story consisting of 10 concepts (Madaan et al., 2023)[†]. We experiment with 100 samples for the purpose of this study.

Evaluation Metrics. We evaluate the generated stories along two axes: **constraints satisfaction** and **overall story quality**. For constraints satisfac-tion, we report two metrics: (a) **All Present (AP):** fraction of samples where all constraints are sat-isfied i.e., there are no missing concepts, and (b) **Missing Concepts (MC):** average percentage of missing concepts. Higher 'all present' and lower 'missing concepts' are preferable. We identify a concept as missing if it does not appear in the story in any word form. For evaluating overall story qual-ity, we conduct a pairwise evaluation with GPT-4. The evaluation prompt is provided in Fig. 7. To ac-count for position bias in this pairwise comparison, we follow our findings from the LLM Evaluation task and conduct each evaluation twice, by swap-ping the order of the stories and preferring one story over the other only if the evaluations are consistent.

Implementation Details. We evaluate BSM us-ing LLaMA-2-7B-chat and LLaMA-2-70B-chat. All modules generate text using greedy decoding. For the branch module, the LLM is prompted to divide the concepts into two groups.

Baselines. We compare BSM to (1) **zero-shot prompting with the same LLM:** given a set of concepts, directly generates the story, (2) **plan&solve prompting**, that first proposes a story topic (as a plan) and then generates a story on that

Method	LLaMA-2-7B-chat		LLaMA-2-70B-chat	
	AP↑	MC↓	AP↑	MC↓
Zero-shot	15.0	17.3	22.0	27.2
Plan&Solve	13.0	18.0	21.0	26.6
Self-Consis	19.0	16.6	24.0	20.1
BSM	**23.0**	**15.5**	**28.0**	**14.7**

Table 6: Constrained story generation evaluation results. BSM (with both LLaMA-2 models) improves constraint satisfaction in the generated stories.

topic, (3) **self-consistency**, where we first sample multiple stories and then prompt the LLM again to select one of the sampled stories that has more constraints satisfied.

4.2.2 Results and Analysis

Constraint Satisfaction. Our main results are given in Table 6. They show that BSM with both model variants outperforms all baselines on our constraint satisfaction metrics. We also note that this is still a challenging task even for a stronger LLaMA-2-70B-chat model and the scale of the model has little impact on constraint satisfaction. For example, even BSM with LLaMA-2-70B-chat omits at least one concept for 72% of the sam-ples, echoing the findings from prior work that constrained text generation is hard even for state-of-the-art LLMs (Yao et al., 2023a). We provide an analysis of missing concepts in BSM in Ap-pendix B.

Overall Story Quality. BSM not only satisfies more constraints but almost always generates a more coherent story. We find that in a head-to-head comparison with the zero-shot prompting baseline (with LLaMA-2-70B-chat), stories generated by BSM are preferred a substantial 93% of the time by GPT-4. This can be attributed to two aspects of BSM. First, in each of the branches, the model conditions on a lesser number of concepts and thus generates intermediate stories that by themselves are more coherent. Second, in the merging step, the model is able to condition on these two interme-diate stories and generate a final story that further improves the coherence.

5 Conclusion

We presented BSM, an LLM program for improv-ing LLM evaluation and generation. We conducted two case studies with different implementations of branch, solve, and merge modules, showcasing the effectiveness and generalizability of BSM.

Limitations

We list the limitations of our work below.

1. Evaluating for safety, toxicity, and bias in LLM generations is also critical for a holistic evaluation of LLMs, however, we do not address this topic in our paper.

2. While BSM obtains improvements in length bias, we note that measuring length bias in isolation is challenging because knowing whether the model prefers the longer response because of its length (and not for another reason) is an interpretability question and humans also tend to prefer longer responses, especially for open-ended questions.

3. Recursive or multi-level BSM, in which an LLM recursively branches into parallel sub-tasks is an interesting avenue for future work but we do not explore this in this work due to the increased computation cost.

4. Decomposition into parallel sub-tasks should also help improve efficiency (e.g., compared to sequential decompositions) (Ning et al., 2023) but in this work, we instead focused on improving the task performance.

Acknowledgments

The authors thank the reviewers for their helpful comments and suggestions.

References

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022b. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, et al. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2023. A survey on evaluation of large language models. *arXiv preprint arXiv:2307.03109*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Cheng-Han Chiang and Hung-yi Lee. 2023. Can large language models be an alternative to human evaluations? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15607–15631, Toronto, Canada. Association for Computational Linguistics.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Jaemin Cho, Abhay Zala, and Mohit Bansal. 2023. Visual programming for text-to-image generation and evaluation. In *NeurIPS*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Antonia Creswell and Murray Shanahan. 2022. Faithful reasoning using large language models. *arXiv preprint arXiv:2208.14271*.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2019. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*.

David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A Saurous, Jascha Sohl-Dickstein, et al. 2022. Language model cascades. *arXiv preprint arXiv:2207.10342*.

Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. 2022. Successive prompting for decomposing complex questions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1251–1265.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*.

Deep Ganguli, Amanda Askell, Nicholas Schiefer, Thomas Liao, Kamilė Lukošiūtė, Anna Chen, Anna Goldie, Azalia Mirhoseini, Catherine Olsson, Danny Hernandez, et al. 2023. The capacity for moral self-correction in large language models. *arXiv preprint arXiv:2302.07459*.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Rishav Hada, Varun Gumma, Adrian de Wynter, Harshita Diddee, Mohamed Ahmed, Monojit Choudhury, Kalika Bali, and Sunayana Sitaram. 2023. Are large language model-based evaluators the solution to scaling up multilingual evaluation? *arXiv preprint arXiv:2309.07462*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2021. Text modular networks: Learning to decompose tasks in the language of existing models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1264–1279.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. 2023. OpenAssistant conversations–democratizing large language model alignment. *arXiv preprint arXiv:2304.07327*.

Bin Lei, Chunhua Liao, Caiwen Ding, et al. 2023. Boosting logical reasoning in large language models through a new framework: The graph of thought. *arXiv preprint arXiv:2308.08614*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259*.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. 2022. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2020. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1823–1840.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, et al. 2022. Neurologic a* esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 780–799.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Neurologic decoding:(un) supervised neural text generation

with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4299.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Huazhong Yang, and Yu Wang. 2023. Skeleton-of-thought: Large language models can do parallel decoding. *arXiv preprint arXiv:2307.15337*.

OpenAI. 2023a. Evals is a framework for evaluating llms and llm systems, and an open-source registry of benchmarks.

OpenAI. 2023b. Gpt-4 technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Swarnadeep Saha, Xinyan Yu, Mohit Bansal, Ramakanth Pasunuru, and Asli Celikyilmaz. 2023. MURMUR: Modular multi-step reasoning for semistructured data-to-text generation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11069–11090, Toronto, Canada. Association for Computational Linguistics.

Swarnadeep Saha, Shiyue Zhang, Peter Hase, and Mohit Bansal. 2022. Summarization programs: Interpretable abstractive summarization with neural modular trees. In *The Eleventh International Conference on Learning Representations*.

Imanol Schlag, Sainbayar Sukhbaatar, Asli Celikyilmaz, Wen-tau Yih, Jason Weston, Jürgen Schmidhuber, and Xian Li. 2023. Large language model programs. *arXiv preprint arXiv:2305.05364*.

Eric Smith, Orion Hsu, Rebecca Qian, Stephen Roller, Y-Lan Boureau, and Jason Weston. 2022. Human evaluation of conversations is an open problem: comparing the sensitivity of various methods for evaluating dialogue agents. In *Proceedings of the 4th Workshop on NLP for Conversational AI*, pages 77–97.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023b. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. 2023c. How far can camels go? exploring the state of instruction tuning on open resources. *arXiv preprint arXiv:2306.04751*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Minghao Wu and Alham Fikri Aji. 2023. Style over substance: Evaluation biases for large language models. *arXiv preprint arXiv:2307.03025*.

Shunyu Yao, Howard Chen, Austin W Hanjie, Runzhe Yang, and Karthik Narasimhan. 2023a. COLLIE: Systematic construction of constrained text generation tasks. *arXiv preprint arXiv:2307.08689*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023b. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yao Yao, Zuchao Li, and Hai Zhao. 2023c. Beyond chain-of-thought, effective graph-of-thought reasoning in large language models. *arXiv preprint arXiv:2305.16582*.

Xinghua Zhang, Bowen Yu, Haiyang Yu, Yangyu Lv, Tingwen Liu, Fei Huang, Hongbo Xu, and Yongbin Li. 2023. Wider and deeper llm networks are fairer llm evaluators. *arXiv preprint arXiv:2308.01862*.

	Ag↑	PB↓	LB↓
Zero-shot (w/ LLaMA-2-70B-c)	0.46	34.00	45.00
BSM (w/ LLaMA-2-70B-c)	**0.48**	**23.77**	**40.25**

Table 7: Results of reference-free evaluation of 'Reasoning' questions. BSM outperforms the zero-shot baseline in evaluating 'Reasoning' questions, even when reference answers are not used (on a random subset of 100 samples).

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Additional Experiments: LLM Evaluation

A.1 Experimental Setup

Dataset. We experiment with the MT-Bench dataset, that evaluates LLMs as judges of other LLM's responses when acting as helpful AI assistants in multi-turn conversations (Zheng et al., 2023). It consists of 2400 LLM responses and 3000 expert human judgements. LLM outputs are responses to 80 representative instructions from 8 diverse domains: writing, roleplay, extraction, reasoning, math, coding, knowledge I (STEM), and knowledge II (humanities/social science). Each question is a conversational question, consisting of two turns, in which the turn-2 question is a follow-up to the turn-1 question. For each question, the dataset consists of responses from 6 different LLMs (Alpaca-13B, Vicuna-13b, LLaMA-13B, Claude-v1, GPT-3.5-turbo, and GPT-4), resulting in 15 possible response pairs. Thus, the entire evaluation set consists of 300 response-pair samples per category.

Implementation Details. Algorithm 1 shows the LLM program. For better reproducibility, all modules in BSM generate text using greedy decoding. For the branch module, the LLM is prompted to generate a plan consisting of a maximum of five evaluation criteria (which we found it adheres to in experiments). For the merge module, we find that the non-neural merge of summing up the criterion-wise evaluations is simple and works well in practice, hence all our experimental results are reported with that method. The prompts are shown in Figures 4 and 5. All experiments are run on an AWS cluster of 8 A100 GPUs.

Baselines. All methods, including BSM, account for position bias in the same manner, generating a verdict for both encoding orders and choosing the final verdict based on the individual verdicts (assigning a tie if the two encoding orders disagree). In particular, Self-Consistency computes majority vote independently for each encoding order.

A.2 Results and Analysis

BSM generalizes well across domains. In Table 5, we evaluate BSM's ability to evaluate generations for questions in the categories of 'Roleplay', 'Extraction', 'Stem', and 'Humanities'. We find that BSM is robust and performs well across domains in terms of improvement over the LLaMa-2-70B-chat baseline, and approaches GPT-4 performance on several of the domains. In particular, on the Stem domain, it is able to improve agreement scores over the baseline by up to 26% (absolute), match GPT-4, and even outperform it in terms of position and length biases. Table 7 shows that BSM outperforms the zero-shot baseline for 'Reasoning' questions, even in reference-free evaluations (i.e., GPT-4 generated answers are not used either in the baseline or in BSM).

Combining BSM and SC reduces position bias further. BSM generates a single solution for each sub-problem (each branch). A possible enhancement is combining BSM with self-consistency i.e., sampling multiple solutions for each sub-problem. In particular, we implement BSM+SC by sampling five evaluations per branch (with temperature 0.7) and then the score for each sub-evaluation in that branch is given by the average score. We compare BSM with BSM+SC in Table 8. While agreement scores do not improve further, we observe a 2%

	Overall			Turn-1			Turn-2		
	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓
BSM	0.55	17.33	39.09	0.60	14.66	39.46	0.50	20.00	39.13
BSM + SC	0.55	15.33	39.09	0.61	10.66	39.06	0.49	20.00	39.13

Table 8: Effect of using Self-Consistency within each branch of BRANCH-SOLVE-MERGE (BSM+SC). Results are with the LLaMA-2-70B-chat model. While the overall agreement scores do not improve further, we obtain a further 2% reduction in position bias.

	Overall	Turn 1	Turn 2
Vicuna-33B	0.52	0.53	0.51
BSM (w/ Vicuna-33B)	**0.55**	**0.56**	**0.54**
LLaMA-2-70B	0.48	0.58	0.37
BSM (w/ LLaMA-2-70B)	**0.53**	**0.59**	**0.47**
GPT-4	0.61	0.59	**0.63**
BSM (w/ GPT-4)	**0.63**	**0.63**	0.62

Table 9: LLM-Human agreement scores for the 'writing' category questions (overall and individually for turn-1 and turn-2). Here agreement is computed using majority voting (instead of treating each human vote per sample independently).

reduction in position bias. This points to two conclusions. First, BSM, through its decomposition approach, already constructs sub-problems that are granular enough and hence, the variance reduction that one obtains through self-consistency within each sub-problem is limited. However, the moderate reduction in position bias still reflects its usefulness, which is a direct effect of making evaluations more consistent.

Effect of Branching Factor. BSM has the benefit of relying on the underlying LLM for deciding what sub-problems to branch to, while the prompt controls the maximum branching factor (see the phrase 'list of up to five factors' in the branch prompt in Fig. 4). We vary this maximum branching factor from 2 to 5 and study its effect on 100 samples from the 'writing' category of questions. Table 10 reports our findings. We observe highest agreement at a branching factor of 4, after which the result mostly saturates. In general, the optimal branching factor should depend on the specific question under consideration and unlike past work where users specify what factors to evaluate on (Liu et al., 2023; Zheng et al., 2023), BSM generates that plan on its own. Position bias continues to decrease with increasing branching factor, where more branches helps reduce variance in the final judgment.

BSM is robust to evaluation scale. Evaluation tasks, in general, require defining a scale for scoring the responses. In Table 11, we compare the performance of BSM by varying this evaluation scale, specified in the 'solve' prompt (see Fig 5), either scoring 1-5 (used in the main experiments) or 1-10. We observe that BSM is fairly robust to such variations, obtaining comparable agreement scores. The position bias, however, increases slightly with a larger scale.

B Additional Experiments: Constrained Text Generation

Analysis of Missing Concepts in BSM. The source of missing concepts in BSM can be attributed to one of the following two categories: (a) the 'solve' module, i.e., the model omits concepts even when generating an intermediate story in a branch subproblem with a lesser number of concepts; or (b) the 'merge' module, i.e., the intermediate stories include their respective concepts but the fusion process omits some of these. We observe that out of 72% of the BSM stories (with LLaMA-2-70B-chat) where at least one concept is missing, a significant 60% of these belong to the first category (i.e., concept omission in the 'solve' module) versus only 12% belong to the second category (i.e., concept omission during 'merging'). This suggests that constraint satisfaction can be further improved via a 'recursive' BSM method involving iterative branching to even more granular sub-problems. However, recursive BSM would be significantly more expensive because of many more calls to the base LLM. We leave this exploration as part of future work.

BF	Overall			Turn-1			Turn-2		
	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓
2	0.50	22.00	49.20	0.49	24.00	45.00	0.50	22.00	56.52
3	0.52	19.00	38.09	0.53	14.00	35.00	0.51	24.00	43.47
4	**0.53**	19.00	38.09	0.51	16.00	35.00	0.55	22.00	43.47
5	0.52	**12.00**	**34.92**	0.51	12.00	35.00	0.54	12.00	34.78

Table 10: Impact of maximum Branching Factor (BF) on 100 samples of the 'writing' category in LLM response evaluation with BSM on LLaMA-2-70B-chat.

Solving Technique	Overall			Turn-1			Turn-2		
	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓	Ag↑	PB↓	LB↓
Eval Scale (1-5)	0.52	12.00	34.92	0.51	12.00	35.00	0.54	12.00	34.78
Eval Scale (1-10)	0.50	18.00	36.50	0.51	18.00	40.00	0.50	18.00	30.43

Table 11: Analysis of evaluation scale for LLM response evaluation, with 100 samples of 'writing' category. BSM (with LLaMA-2-70B-chat) is fairly robust to such variations.
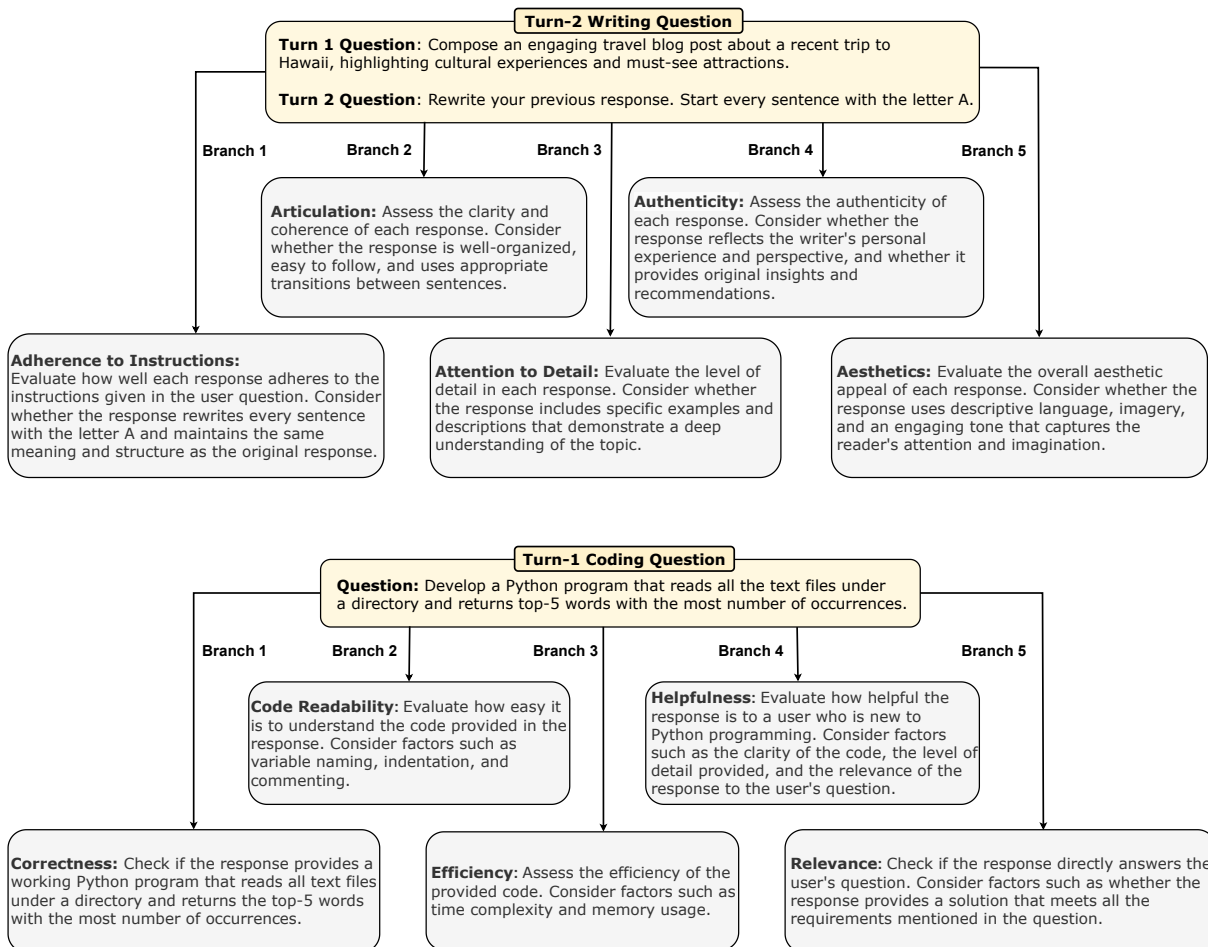

Figure 2: Examples of LLM Evaluation branch generation. We show different branches (evaluation plans) generated by BSM with a LLaMA-2-70B-chat model for different kinds of questions: (top) a turn-2 writing question and (bottom) a coding question.
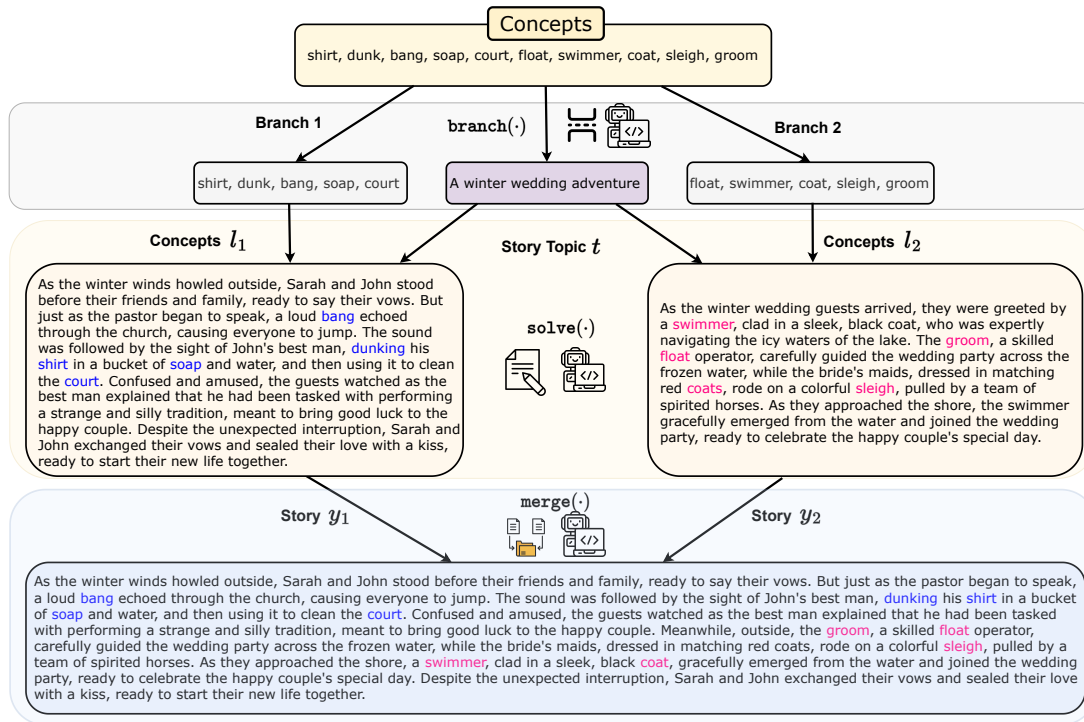
Figure 3: An illustration of BSM with LLaMA-2-70B-chat for constrained story generation. Given a set of random concepts, the 'Branch' module first divides them into two sets and generates a story topic. The 'Solve' module conditions on the concepts and the topic to generate an intermediate story for each branch. The 'Merge' module merges the intermediate stories to generate a final story ensuring that all concepts are still present.

Algorithm 1 BRANCH-SOLVE-MERGE: LLM Program for LLM Evaluation

---

Require: Question q, LLM Responses $\{r^{(A)}, r^{(B)}\}$, Modules $m = \{\texttt{branch}(\cdot), \texttt{solve}(\cdot), \texttt{merge}(\cdot)\}$
  function BSM($q, r^{(A)}, r^{(B)}, m, swap$)
    $C \leftarrow \texttt{branch}(q)$ ▷ Branch to different evaluation criteria.
    for each $c_i \in C$ **do**
      if $swap = False$ **then**
        $s_i^{(A)}, s_i^{(B)} \leftarrow \texttt{solve}(q, r^{(A)}, r^{(B)}, c_i)$ ▷ Solve for each of the criteria.
      else
        $s_i^{(A)}, s_i^{(B)} \leftarrow \texttt{solve}(q, r^{(B)}, r^{(A)}, c_i)$
      end if
    end for
    $y^{(A,B)} \leftarrow \texttt{merge}(q, \{c_i\}_{i=1}^k, \{s_i^{(A)}\}_{i=1}^k, \{s_i^{(B)}\}_{i=1}^k)$ ▷ Merge the individual evaluations.
    return $y^{(A,B)}$
  end function
  function BRANCH-SOLVE-MERGE($q, r^{(A)}, r^{(B)}, m$)
    $y^{(A,B)} = \text{BSM}(q, r^{(A)}, r^{(B)}, m, swap = False)$ ▷ Get verdict by not swapping the response order.
    $y^{(B,A)} = \text{BSM}(q, r^{(A)}, r^{(B)}, m, swap = True)$ ▷ Get verdict by swapping the response order.
    if $y^{(A,B)} = A$ & $y^{(B,A)} = B$ **then**
      $y \leftarrow A$ ▷ Choose a response only if the individual evaluations are consistent.
    else if $y^{(A,B)} = B$ & $y^{(B,A)} = A$ **then**
      $y \leftarrow B$
    else
      $y \leftarrow tie$
    end if
    return y
  end function

Branch Prompt for Turn-1 Question

We want to evaluate the quality of the responses provided by two AI assistants to the user question displayed below. Your task is to propose an evaluation plan that can be executed to compare the two responses. The evaluation plan should consist of a list of up to five factors that one should consider such as helpfulness, relevance, accuracy, etc. In each line, write an evaluation criterion along with a short descrition of how we should evaluate that criterion.
User Question: **{question1}**
Evaluation Plan:

Branch Prompt for Turn-2 Question

We want to evaluate the quality of the responses provided by two AI assistants. Focus specifically on the second user question displayed below. Your task is to propose an evaluation plan that can be executed to compare the two responses. The evaluation plan should consist of a list of up to five factors that one should consider such as helpfulness, relevance, accuracy, etc. In each line, write an evaluation criterion along with a short descrition of how we should evaluate that criterion.
First User Question: **{question1}**
Second User Question: **{question2}**
Evaluation Plan:

Figure 4: Branch prompts for Turn-1 and Turn-2 questions for the task of LLM Response Evaluation. The Turn-1 prompt conditions only on the Turn-1 question while the Turn-2 prompt conditions on both turn questions to generate an evaluation plan.

Solve Prompt for Turn-1 Question

You are given a user question and responses provided by two AI assistants. Your task is to evaluate and score the quality of the responses based on a single evaluation criterion displayed below. Make sure to evaluate only based on the criterion specified and none other. In the first line, provide a score between 1 to 5 for Assistant A's response. In the second line, provide a score between 1 to 5 for Assistant B's response.
[User Question]
{question1}

[The Start of Assistant A's Answer]
{answer_a}
[The End of Assistant A's Answer]

[The Start of Assistant B's Answer]
{answer_b}
[The End of Assistant B's Answer]

[Evaluation Criterion]
{eval_criterion}
[End of Evaluation Criterion]

Evaluation of **{criterion_name}**:

Solve Prompt for Turn-2 Question

You are given two user questions and responses provided by two AI assistants for the second question. Your task is to evaluate and score the quality of the responses based on a single evaluation criterion displayed below. Make sure to evaluate only based on the criterion specified and none other. In the first line, provide a score between 1 to 5 for Assistant A's response. In the second line, provide a score between 1 to 5 for Assistant B's response.
[First User Question]
{question1}

[Second User Question]
{question2}

[The Start of Assistant A's Answer]
{answer_a}
[The End of Assistant A's Answer]

[The Start of Assistant B's Answer]
{answer_b}
[The End of Assistant B's Answer]

[Evaluation Criterion]
{eval_criterion}
[End of Evaluation Criterion]

Evaluation of **{criterion_name}**:

Figure 5: Solve prompts for Turn-1 and Turn-2 questions for the task of LLM Response Evaluation. Each prompt conditions on the question(s), the responses from both LLMs, and a given evaluation criterion that the branch module has generated.

Branch Prompt for Story Generation

Given a set of concepts, we want to write a concise and coherent story consisting of a few sentences using those concepts. In order to do so, you task is to first propose a story topic and then divide the concepts into two groups such that the story generated from each group of concepts can be combined together to form a longer story. Make sure that you do not leave out any concepts.

Concepts: **{concepts}**

Solve Prompt for Story Generation

Write a concise and coherent story on the following topic consisting of a single paragraph. Make sure to include all the following concepts in the story.

Concepts: **{concepts}**
Story Topic: **{story_topic}**

Merge Prompt for Story Generation

Given two groups of concepts and two stories containing those concepts, combine the two stories into a concise and coherent story consisting of a single paragraph. Make sure that the combined story does not miss any concept from the two groups.

Group 1 Concepts: **{concepts1}**
Story 1: **{story1}**
Group 2 Concepts: **{concepts2}**
Story 2: **{story2}**

Figure 6: Branch, Solve, and Merge prompts for the task of Constrained Story Generation. The branch prompt conditions on the concepts. The solve prompt conditions on a subset of concepts and a story topic that the branch module has generated. The merge prompt conditions on the two intermediate stories that the solve module has generated and their respective concept-sets.

Story Quality Evaluation Prompt

Please act as an impartial judge and evaluate the quality of the stories provided by two AI assistants. Your evaluation should consider factors such as grammaticality, coherence, engagement, etc. Begin your evaluation by comparing the two stories and provide a short explanation. Avoid any position biases and ensure that the order in which the stories were presented does not influence your decision. Do not allow the length of the responses to influence your evaluation. Do not favor certain names of the assistants. Be as objective as possible. After providing your explanation, output your final verdict by strictly following this format: \"[[A]]\" if story A is better, \"[[B]]\" if story B is better, and \"[[C]]\" for a tie.

Story A: **{story_a}**

Story B: **{story_b}**

Figure 7: Prompt for evaluating the quality of the stories with GPT-4. It asks the model to perform a pair-wise evaluation between the stories generated by the baseline method and by BSM.