SYNTACTIC AND SEMANTIC CONTROL OF LARGE LANGUAGE MODELS VIA SEQUENTIAL MONTE CARLO

Anonymous authors
Paper under double-blind review

ABSTRACT

A wide range of LLM applications require generating text that conforms to syntactic or semantic constraints. Imposing such constraints nontrivially alters the distribution over sequences, usually making exact sampling intractable. In this work, building on the Language Model Probabilistic Programming framework of Lew et al. (2023), we develop an approach to approximate inference for controlled LLM generation based on sequential Monte Carlo (SMC). Our SMC framework allows us to flexibly incorporate domain- and problem-specific constraints at inference time, and efficiently reallocate computation in light of new information during the course of generation. We demonstrate that our approach improves downstream performance on four challenging domains—Python code generation for data science, text-to-SQL, goal inference, and molecule synthesis. We compare to a number of alternative and ablated approaches, showing that our accuracy improvements are driven by better approximation to the full Bayesian posterior.

1 INTRODUCTION

The goal of *controlled generation* from language models (LMs) is to produce text guided by a set of syntactic or semantic constraints. One prominent setting for controlled generation is semantic parsing, or code generation, which involves generating text in a programming (or other formal) language. Much prior work in this space has focused on ensuring that the LM outputs adhere to the formal language using regular expressions or context-free grammars (Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024). But in practice, we may wish to use diverse signals beyond grammaticality to guide generation. For example:

- Checking (partial) code statically (type-checking, linting, partial evaluation);
- Running (partial) code on a test case and checking if it raises an error or returns the wrong answer;
- Simulating environments (e.g. in robotics or chemistry) and assigning a score to the resulting state;
- Rolling out possible completions of partial code and computing their max, min, or average score;
- Asking another language model to critique the code generated so far.

Such signals vary along several important dimensions: some are cheap to compute (linting), others are more costly (simulations); some can be evaluated incrementally with each sampled token (language model critique), others provide sparser guidance (running code); some enforce binary hard constraints (type-checking), others yield soft continuous scores (scoring).

One way to represent such signals uniformly is as *potential functions* $\phi : \mathcal{A}^* \to \mathbb{R}_{\geq 0}$, assigning non-negative scores to sequences of tokens from an alphabet $\mathcal{A}$. Given a set Φ of such potentials, we can then frame the problem of controlled generation probabilistically: we wish to sample from the *global product of experts* distribution on complete sequences $\boldsymbol{x}$. We define

$$p_{\text{global}}(\boldsymbol{x}) = \frac{1}{Z}\, p_{\text{LM}}(\boldsymbol{x}) \prod_{\phi \in \Phi} \phi(\boldsymbol{x}),$$

where p_{LM} is the LM's distribution on complete token sequences, and Z is the normalizing constant. Even when each ϕ can be evaluated relatively cheaply, sampling exactly from this distribution is

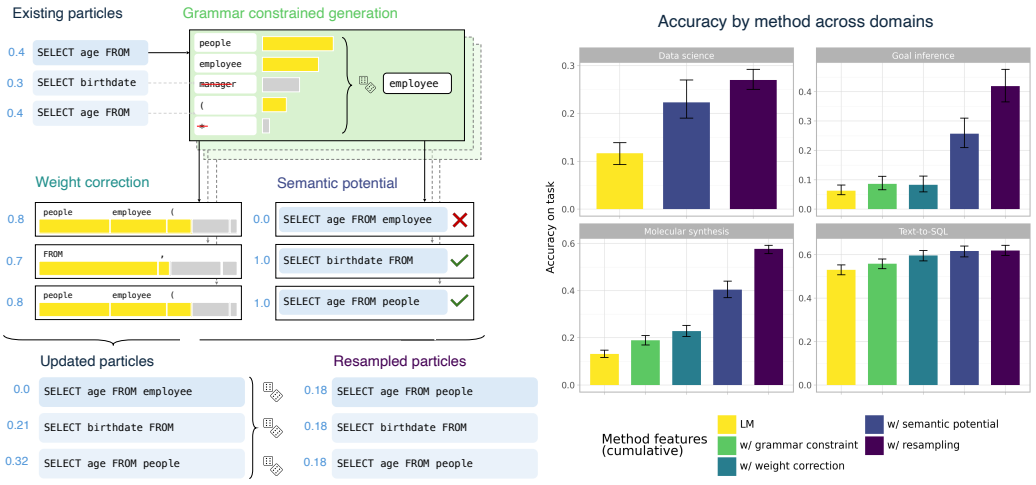

Figure 1: **Controlled generation from LLMs via sequential Monte Carlo.** *Left:* We use sequential Monte Carlo to sample from high-quality approximations to Bayesian posteriors over LLM outputs. Partial sequences are repeatedly extended via grammar-constrained generation. We then apply weight corrections to mitigate the greediness of locally constrained decoding, as well as semantic potentials to encode rich information that cannot be included in logit masks. Finally, resampling focuses computation on promising particles. *Right:* Cumulative accuracy gains from these innovations, on challenging data science, text-to-SQL, goal inference, and molecule synthesis benchmarks.

generally intractable. Popular approaches based on per-token logit biasing or masking avoid the intractable global normalizing constant, but introduce greedy approximations that can badly distort the distribution (Lew et al., 2023; Park et al., 2024).

Unnormalized densities over token sequences also arise in other difficult language modeling problems, such as infilling, prompt engineering, and prompt intersection, where *Sequential Monte Carlo* has been proposed as an effective means of approximating these intractable distributions (Lew et al., 2023; Zhao et al., 2024). In this paper, we use sequential Monte Carlo to tackle a number of challenging semantic parsing problems, using incremental static and dynamic analyses to inform both proposal distributions and twist functions—algorithm components that in previous work were learned via a costly contrastive fine-tuning procedure (Zhao et al., 2024). Our approach enables us to flexibly integrate heterogeneous signals that capture both syntactic and semantic aspects of each domain.

Our paper makes the following contributions:

- *SMC for Constrained Semantic Parsing.* We develop a variant of sequential Monte Carlo specialized for semantic parsing and code generation under diverse syntactic and semantic constraints (§2). Unlike many previous frameworks for constrained decoding, our algorithm can integrate constraints that cannot be incrementally evaluated to compute masks on the entire token vocabulary, as well as constraints that can only be evaluated at irregular intervals during generation.
- *Empirical Evaluation of Performance in Diverse Domains.* We implement our approach, and five baselines, in four challenging problem domains: Python code generation for data science, text-to-SQL, goal inference, and molecule synthesis (§3.1). We find that sequential Monte Carlo significantly improves performance across domains (§3.2).
- *Empirical Evaluation of Algorithm Components.* We run ablation experiments, and find that improved performance can be attributed to three algorithmic components: *weight correction*, which mitigates the greediness of popular locally constrained decoding methods; *semantic potentials*, which incorporate useful signals that some baseline methods cannot integrate; and *adaptive resampling*, which adaptively focuses computation on partial sequences that look more promising.
- *Empirical Validation of the Probabilistic Perspective.* We derive estimators of the KL divergence from each method's output distribution to the global product of experts (Appendix D). We find that generation quality is correlated with how well each method approximates the global product of experts distribution, and that the global product of experts distribution is calibrated, in that higher-probability generations are more likely to perform well in each domain (§3.3).

2 Monte Carlo Inference for Constrained Semantic Parsing

Notation. Let $\mathcal{A}$ be a vocabulary of tokens, including a distinguished token EOS marking the end of a complete sequence. We write $\mathcal{A}^*_{\text{EOS}}$ for the set of EOS-terminated token sequences, and $\mathcal{A}^*$ for the set of prefixes of strings in $\mathcal{A}^*_{\text{EOS}}$. We use $\boldsymbol{x}$ to refer to a sequence of tokens, with x_i being the i^{th} token in the sequence. We use the notation $\boldsymbol{x}_{<t}$ to refer to the sequence $x_1 \cdots x_{t-1}$, and $\boldsymbol{x}_{\leq t}$ to refer to the sequence $\boldsymbol{x}_{<t}x_t$. Here, juxtaposition indicates sequence concatenation.

We consider a set Φ of domain- or task-specific potential functions that encode relevant constraints. Each potential function $\phi : \mathcal{A}^* \to \mathbb{R}_{\geq 0}$ returns a non-negative real $\phi(\boldsymbol{x})$ when evaluated on some sequence $\boldsymbol{x}$, freely using any structure in the sequence so far. We assume that all potentials satisfy $\phi(\boldsymbol{x}) = 0 \implies \phi(\boldsymbol{xy}) = 0$, for all $\boldsymbol{x}, \boldsymbol{y}$ such that $\boldsymbol{xy} \in \mathcal{A}^*_{\text{EOS}}$. Intuitively, this means it is safe to cancel a partial generation if $\phi(\boldsymbol{x}) = 0$, because there is no completion that can make ϕ positive.

A *language model* p_{LM} is a probability distribution on $\mathcal{A}^*_{\text{EOS}}$. Note that any such distribution factors autoregressively into a product of conditional next-token distributions:

$$p_{\text{LM}}(\boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{x}|} p_{\text{LM}}(x_t \mid \boldsymbol{x}_{<t}). \tag{1}$$

Target distribution. Our goal in controlled generation is to sample the *global product of experts* distribution induced by the potentials Φ, which we write as $[p_{\text{LM}} \otimes \Phi]$:

$$p_{\text{global}}(\boldsymbol{x}) = [p_{\text{LM}} \otimes \Phi](\boldsymbol{x}) = \frac{p_{\text{LM}}(\boldsymbol{x}) \prod_{\phi \in \Phi} \phi(\boldsymbol{x})}{\sum_{\boldsymbol{y} \in \mathcal{A}^*_{\text{EOS}}} p_{\text{LM}}(\boldsymbol{y}) \prod_{\phi \in \Phi} \phi(\boldsymbol{y})} = \frac{1}{Z} p_{\text{LM}}(\boldsymbol{x}) \prod_{\phi \in \Phi} \phi(\boldsymbol{x}). \tag{2}$$

The distribution is normalized globally with respect to all possible complete sequences of tokens. When each potential function takes values in $[0, 1]$, p_{global} can be understood intuitively as the *rejection sampling* distribution that arises by repeatedly generating $\boldsymbol{x} \sim p_{\text{LM}}$ until a sample $\boldsymbol{x}$ is *accepted*, where each sample $\boldsymbol{x}$ is accepted with probability $\prod_{\phi \in \Phi} \phi(\boldsymbol{x})$. Depending on how frequently raw samples from p_{LM} satisfy all the constraints, rejection sampling can be extremely expensive. Our work aims to accurately approximate p_{global} with much less computation.

Locally constrained decoding. A popular approach to enforcing constraints at decode-time is to apply constraints before sampling each token, as a *mask* or *bias* applied to the logits computed by the LM (see, e.g., Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Ugare et al., 2024). In this approach, at each time step t, the current sequence $\boldsymbol{x}_{<t}$ is extended with a new token x_t, drawn proportionally to $p_{\text{LM}}(x_t \mid \boldsymbol{x}_{<t}) \cdot \prod_{\phi \in \Phi} \frac{\phi(\boldsymbol{x}_{<t}x_t)}{\phi(\boldsymbol{x}_{<t})}$. Samples drawn in this way are distributed according to a *local product of experts* distribution, which we write $[p_{\text{LM}} \odot \Phi]$:

$$p_{\text{local}}(\boldsymbol{x}) = [p_{\text{LM}} \odot \Phi](\boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{x}|} \frac{p_{\text{LM}}(x_t \mid \boldsymbol{x}_{<t}) \prod_{\phi \in \Phi_{\text{eff}}} \frac{\phi(\boldsymbol{x}_{<t}x_t)}{\phi(\boldsymbol{x}_{<t})}}{\sum_{x'_t \in \mathcal{A}} p_{\text{LM}}(x'_t \mid \boldsymbol{x}_{<t}) \prod_{\phi \in \Phi_{\text{eff}}} \frac{\phi(\boldsymbol{x}_{<t}x'_t)}{\phi(\boldsymbol{x}_{<t})}}. \tag{3}$$

Note that unlike in Eqn. 2, the normalization is now performed locally, once per token.

Despite its popularity, locally constrained decoding has two important shortcomings. First, the local product of experts can only be sampled efficiently when it is possible to cheaply evaluate the potentials $\phi \in \Phi$ on all possible one-token continuations $\boldsymbol{x}_{<t}x'_t$ of the current sequence. For some constraints (e.g., checking membership in the language of a regular expression or context-free grammar), there do exist algorithms for efficient parallel evaluation across tens of thousands of possible continuations. However, for many of the applications of interest in the present paper (including those listed in Tab. 1, e.g., error-checking with test-cases), this is not feasible. Second, even when potential functions can be incrementally and cheaply computed, the local and global product of experts do not define the same distribution (Lew et al., 2023; Park et al., 2024). In particular, the local product enforces constraints greedily, which can lead to myopic sampling down dead-end paths. Continuations with high probability in the short run may have zero mass under a potential just a few steps later.

Importance sampling. Both these problems can be addressed with *importance sampling*, a standard Monte Carlo technique for approximating intractable distributions. We describe a particular application of the technique specialized to this setting. First, from the overall set of potentials Φ, we identify a subset of *efficient potentials* Φ_{eff}, which admit efficient computation of local normalizing constants. This means that the local product of experts $[p_{\text{LM}} \odot \Phi_{\text{eff}}]$ with respect to just Φ_{eff} can be tractably sampled; we use it as a *proposal distribution*. The importance sampling algorithm repeatedly generates multiple *particles* $\boldsymbol{x}^{(i)} \sim [p_{\text{LM}} \odot \Phi_{\text{eff}}]$, then computes corresponding *importance weights*

$$w(\boldsymbol{x}^{(i)}) = \frac{Z \cdot [p_{\text{LM}} \otimes \Phi](\boldsymbol{x}^{(i)})}{[p_{\text{LM}} \odot \Phi_{\text{eff}}](\boldsymbol{x}^{(i)})} = \left(\prod_{t=1}^{|\boldsymbol{x}^{(i)}|} \sum_{x'_t \in \mathcal{A}} p_{\text{LM}}(x'_t \mid \boldsymbol{x}^{(i)}_{<t}) \prod_{\phi \in \Phi_{\text{eff}}} \frac{\phi(\boldsymbol{x}^{(i)}_{<t} x'_t)}{\phi(\boldsymbol{x}^{(i)}_{<t})} \right) \cdot \prod_{\phi \in \Phi \setminus \Phi_{\text{eff}}} \phi(\boldsymbol{x}^{(i)}). \tag{4}$$

The first term in the weight is a product of local normalizing constants that are already computed as a byproduct of sampling the local product of experts. The second term can be computed by running each of the less efficient potentials once, on the final generated sequence. The first term corrects for the greediness of locally constrained decoding, penalizing particles that wound up in "dead ends" where *all* possible continuations $x'_t \in \mathcal{A}$ scored poorly under some potential. The second term multiplies in the potentials that were not directly accounted for during the sample generation process.

Given samples $\boldsymbol{x}^{(1)}, \cdots \boldsymbol{x}^{(N)}$, importance sampling approximates the posterior using the distribution $\sum_{i=1}^{N} \frac{w(\boldsymbol{x}^{(i)})}{\sum_{j=1}^{N} w(\boldsymbol{x}^{(j)})} \cdot \delta_{\boldsymbol{x}^{(i)}}$. Under mild conditions, this approximation converges to the posterior as N grows — however, the number of particles required to obtain a good approximation of the target distribution is exponential in the KL divergence between proposal and target (Chatterjee & Diaconis, 2018). This means that, in practice, if the local product of experts is too far from the target distribution, sampling and weighting a modest number of particles will not help.

Sequential Monte Carlo. One avenue for improvement comes from noticing that importance sampling waits until a particle has been completely generated before computing its weight. This is despite the fact that some terms in the weight can, in principle, be computed early, given just a partial sequence: at each step, we can compute a new local normalizing constant for the local product of experts, and even the less efficient potentials may be able to provide some useful signal before the entire sequence is generated. *Sequential Monte Carlo* (SMC; Chopin et al., 2020), is a natural generalization of importance sampling that samples from a *sequence* of related target distributions $p_1, \ldots, p_T$ on state spaces of increasing dimension. In our case, we consider intermediate targets p_t defined on the growing sequence of spaces $\mathcal{A}^*_{\leq t} = \{\boldsymbol{x} \in \mathcal{A}^* \mid \exists \boldsymbol{y} \in \mathcal{A}^*_{\texttt{EOS}}.\ \boldsymbol{x} = \boldsymbol{y}_{\leq t}\}$. The targets are defined as partial-sequence versions of the global product of experts. For $\boldsymbol{x} \in \mathcal{A}^*_{\leq t}$, we have

$$p_t(\boldsymbol{x}) = \frac{\left(\prod_{i=1}^{|\boldsymbol{x}|} p_{\text{LM}}(x_i \mid \boldsymbol{x}_{<i})\right) \prod_{\phi \in \Phi} \phi(\boldsymbol{x})}{\sum_{\boldsymbol{y} \in \mathcal{A}^*_{\leq t}} \left(\prod_{i=1}^{|\boldsymbol{y}|} p_{\text{LM}}(y_i \mid \boldsymbol{y}_{<i})\right) \prod_{\phi \in \Phi} \phi(\boldsymbol{y})}. \tag{5}$$

The key difference between the global product of experts and p_t is that p_t is defined on, and normalized with respect to, possibly incomplete token sequences of at most length t, rather than complete sequences of arbitrary length. As such, unlike both p_{global} and p_{local}, p_t depends on the behavior of the potentials $\phi \in \Phi \setminus \Phi_{\text{eff}}$ when applied to partial sequences. But no matter how the partial potentials are defined, as t approaches ∞, $p_t(\boldsymbol{x})$ approaches $p_{\text{global}}(\boldsymbol{x})$ for all complete sequences $\boldsymbol{x} \in \mathcal{A}^*_{\texttt{EOS}}$.

The sequential Monte Carlo algorithm generates approximations to each intermediate target in turn. We begin with a collection of N weighted particles $(\boldsymbol{x}^i, w^i_0) = (\epsilon, 1)$, where ϵ is the empty sequence of tokens. Then, starting at $t = 1$, we repeat the following three steps until all particles are EOS-terminated:

1. *Extend.* For each $i \in 1, \ldots, N$, if $\boldsymbol{x}^{(i)} \notin \mathcal{A}^*_{\texttt{EOS}}$, propose $x^{(i)}_t \sim [p_{\text{LM}} \odot \Phi_{\text{eff}}](x^{(i)}_t \mid \boldsymbol{x}_{<t})$.

2. *Reweight.* For each extended particle $\boldsymbol{x}^{(i)}$, update the weight $w^{(i)}$ according to the formula

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{Z_t p_t(\boldsymbol{x}^{(i)})}{Z_{t-1} p_{t-1}(\boldsymbol{x}_{<t}^{(i)})} \frac{1}{[p_{\text{LM}} \odot \Phi_{\text{eff}}](x_t^{(i)} \mid \boldsymbol{x}_{<t}^{(i)})} \tag{6}$$

$$= w_{t-1}^{(i)} \cdot \left(\sum_{x_t' \in \mathcal{A}} p_{\text{LM}}(x_t' \mid \boldsymbol{x}_{<t}^{(i)}) \prod_{\phi \in \Phi_{\text{eff}}} \frac{\phi(\boldsymbol{x}_{<t}^{(i)} x_t')}{\phi(\boldsymbol{x}_{<t}^{(i)})} \right) \cdot \prod_{\phi \in \Phi \setminus \Phi_{\text{eff}}} \frac{\phi(\boldsymbol{x}^{(i)})}{\phi(\boldsymbol{x}_{<t}^{(i)})}. \tag{7}$$

3. *Resample.* If the *effective sample size* $\widehat{N}_{\text{ess}} := \left(\sum_{i=1}^{N} (w^{(i)})^2 \right)^{-1}$ is under a chosen threshold (e.g., $\frac{N}{3}$), perform a *resampling step*: generate N independent ancestor indices

$$a^{(i)} \sim \text{Categorical}\left(\frac{w_t^{(1)}}{\sum_{j=1}^{N} w_t^{(j)}}, \ldots, \frac{w_t^{(N)}}{\sum_{j=1}^{N} w_t^{(j)}} \right),$$

then simultaneously reassign all particles $(\boldsymbol{x}^{(i)}, w_t^{(i)}) \leftarrow (\boldsymbol{x}^{(a^{(i)})}, \frac{1}{N} \sum_{j=1}^{N} w_t^{(j)})$.

The extension step generates next tokens from the local product of experts proposal, just as in importance sampling. The reweighting step incrementally computes the importance weight, with a new factor at each token. The resampling step exploits any early signal available in the incremental weights to reallocate computation to promising particles (which are likely to be chosen as ancestors multiple times) and away from unpromising particles (which are unlikely to be chosen as ancestors, and will thus be culled before they can be completed). This reallocation of computation can lead to dramatic improvements in inference quality.

Further extensions. We further extend SMC in two ways. First, potentials in Φ_{eff} may still be modestly expensive to evaluate on the entire vocabulary. In these cases, we can develop cheap stochastic approximations to the full token-masking distributions $[p_{\text{LM}} \odot \Phi_{\text{eff}}](x_t \mid \boldsymbol{x}_{<t})$, and use these as proposals during the *Extend* step. The incremental weight computation must also be corrected to account for these approximations; we derive stochastic unbiased estimators of the incremental weights that can be soundly used within SMC (see Appendix C). Second, the intermediate targets p_t needn't advance generation token-by-token; in some domains it may be beneficial to consider more semantically meaningful increments. For example, the intermediate target p_t may be defined over the space of all partial Python programs containing t or fewer lines of code (rather than tokens); the *Extend* step would then sample a different number of tokens per particle, waiting in each partial sequence until a new full line has been generated. Such strategies can lead to better *particle alignment* (Lundén et al., 2018), making resampling more effective. We exploit this in one of our experiments by implementing SMC steps over Python statements (§3.1).

3 Experiments

We study the performance of our proposed sampling methods on four challenging semantic parsing domains: Goal Inference (STRIPS), Data Science (Python), Text-to-SQL (SQL), and Molecular Synthesis (SMILES). We compare seven approaches to constrained generation:

1. *Language model (p_{LM}).* As a baseline, we report the performance of the base language model (see §3.1 for details on the language models used).
2. *Language model with grammar constraint (Locally-constrained decoding; target: $[p_{\text{LM}} \odot \phi_{\text{CFG}}]$).* This is the approach used by much prior work (Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Moskal et al., 2024; Ugare et al., 2024). In each of our domains, we formulate a context-free grammar (CFG) encoding a notion of syntactic well-formedness appropriate for the domain (see §3.1). Writing ϕ_{CFG} for the binary function that determines whether its input is a prefix of some valid string in the grammar's language, this baseline directly samples the local product-of-experts $[p_{\text{LM}} \odot \phi_{\text{CFG}}]$—i.e., it uses per-token logit masking to greedily enforce the CFG constraint.
3. *Language model with grammar constraint and weight correction (Grammar-only IS; target: $[p_{\text{LM}} \otimes \phi_{\text{CFG}}]$).* This method generates particles from $[p_{\text{LM}} \odot \phi_{\text{CFG}}]$, then computes importance

Table 1: Tasks, languages, their corresponding constraint-encoding potential functions, and an example output.

Task	Language	ϕ_{sem}	Example Output
Goal inference	STRIPS	Plan simulation	`(:goal (and (arm-empty) (on-table b1) (on b2 b1) ... (clear b5))`
Data Science	Python	Error-checking with test cases	`idx_list = ((x == a) & (y == b))` `result = idx_list.nonzero()[0]`
Text-to-SQL	SQL	Dynamic alias and table-column checking	`SELECT born_state FROM head GROUP BY born_state HAVING count(*) >= 3`
Molecular Synthesis	SMILES	Check valences, kekulize aromatic systems	`CC1=CC2(OC=N)C(=O)NC3CC23C(C1)=NO`

weights to correct toward the target $[p_{\text{LM}} \otimes \phi_{\text{CFG}}]$. These weights mitigate some of the greediness of local-product-of-experts sampling, but do not yet integrate any potentials beyond ϕ_{CFG}.

4. *Language model with grammar constraint, weight correction and resampling (Grammar-only SMC; target: $[p_{\text{LM}} \otimes \phi_{\text{CFG}}]$).* This method is a straightforward application of Lew et al. (2023) to locally-constrained decoding; and is similar to Park et al. (2024), which also attempts to correct for the greediness of locally-constrained decoding. As in the previous method, it targets $[p_{\text{LM}} \otimes \phi_{\text{CFG}}]$, but uses resampling to reallocate computation to promising particles.
5. *Language model with grammar constraint and semantic potential (Sample-Rerank; target: $[[p_{\text{LM}} \odot \phi_{\text{CFG}}] \otimes \phi_{\text{sem}}]$).* Sample-Rerank is a common approach for incorporating an external signal into an LM's generations post-hoc, for instance a reward model (e.g. (Nakano et al., 2021)), or a boolean encoding validity (e.g. (Olausson et al., 2023)). In each domain, we formulate an additional potential ϕ_{sem} that encodes task-specific signals of sequence quality (see §3.1). This baseline generates grammar-constrained sequences from the local product of experts $[p_{\text{LM}} \odot \phi_{\text{CFG}}]$, then for each sequence $\boldsymbol{x}$, computes the weight $w(\boldsymbol{x}) = \phi_{\text{sem}}(\boldsymbol{x})$.
6. *Language model with grammar constraint, weight correction, and semantic potential (Full IS; target: $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$).* This is the full importance sampling method described in §2, with $\Phi = \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}$ and $\Phi_{\text{eff}} = \{\phi_{\text{CFG}}\}$. Unlike in the previous method, the importance weights now include correction terms that mitigate the greediness of local sampling. We include this method primarily as an ablation of our next method (SMC), modified not to include incremental resampling.
7. *Language model with grammar constraint, weight correction, semantic potential, and resampling (Full SMC; target: $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$).* This method includes all of the algorithmic contributions of our approach. It is the full sequential Monte Carlo algorithm, with $\Phi_{\text{eff}} = \{\phi_{\text{CFG}}\}$ and $\Phi = \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}$. It targets the same global posterior as the previous method but uses resampling to reallocate computation to promising particles.

We report results using $N = 10$ particles; see App. A.2 for downstream accuracy results for a varying number of particles. We ran experiments on GCP instances with 1 A100 GPU and 12 vCPUs (our CFG parser is implemented for CPU and is parallelized across particles), with the exception of the Data Science domain, for which we used 4 H100 GPUs and 64 vCPUs.

3.1 Domains

We briefly describe our problem domains, summarized in Table 1; see Appendix E for further details.

- **Goal inference (Planetarium).** *Task:* Produce a formal description of an agent's goal in the STRIPS subset of the PDDL planning language (Fikes & Nilsson, 1971; Aeronautiques et al., 1998), given a natural-language description of the goal, as well as PDDL code describing the agent's initial conditions and plan for achieving the goal. *Data:* Blocksworld tasks with up to 10 objects, from the Planetarium benchmark (Zuo et al., 2024). *Metric:* Accuracy with respect to ground-truth PDDL goal. *Base LM:* Llama 3.1 8B. *Grammar:* STRIPS syntax for goals within Planetarium Blocksworld's domain definition. *Semantic Potential:* Evaluation (using the VAL plan validator (Howey et al., 2004)) of whether the known plan, when executed in the known initial conditions, can be proven *not* to satisfy the (partial) goal.

Table 2: Comparison of different methods with their scores on different domains. Errors are bootstrapped 95% confidence intervals.

Method	Score			
	Goal inference	Molecular synthesis	Data science	Text-to-SQL
LM	0.063 (0.05, 0.08)	0.132 (0.12, 0.15)	0.213 (0.19, 0.24)	0.531 (0.51, 0.55)
w/ grammar constraint (Locally-constrained Decoding)	0.086 (0.07, 0.11)	0.189 (0.17, 0.21)	-	0.559 (0.54, 0.58)
w/ grammar constraint, weight correction (Grammar-only IS)	0.083 (0.06, 0.11)	0.228 (0.21, 0.25)	-	0.597 (0.57, 0.62)
w/ grammar constraint, potential (Sample-Rerank)	0.289 (0.24, 0.34)	0.392 (0.36, 0.42)	-	0.581 (0.56, 0.60)
w/ grammar constraint, correction, and resampling (Grammar-only SMC)	0.401 (0.34, 0.46)	0.205 (0.18, 0.23)	-	0.596 (0.57, 0.62)
w/ grammar constraint, potential, and correction (Full IS)	0.257 (0.21, 0.31)	0.404 (0.37, 0.44)	0.346 (0.31, 0.39)	**0.618** (0.59, 0.64)
w/ grammar constraint, potential, correction, and resampling (Full SMC)	**0.419** (0.37, 0.48)	**0.577** (0.56, 0.59)	**0.407** (0.36, 0.45)	**0.620** (0.60, 0.64)

- **Python for data science (DS-1000).** *Task:* Generate Python code that uses standard data science libraries (NumPy, PyTorch, Pandas, etc.) to solve a task specified in natural language and via an (executable) test case. *Data:* The DS-1000 benchmark (Lai et al., 2023). *Metric:* Accuracy of the generated program with respect to the provided test cases. *Base LM:* Llama 3 70B. *Grammar:* We use a trivial potential $\phi_{\text{CFG}}(\boldsymbol{x}) = 1$, as we find that the unconstrained LM reliably generates grammatical Python (that may nonetheless induce runtime errors). *Semantic Potential:* Given a partial program $\boldsymbol{x}$, ϕ_{sem} truncates $\boldsymbol{x}$ to the last-generated newline not ending in `:`, and executes the resulting (partial) program on the provided test case, checking for runtime errors.
- **Text-to-SQL (Spider).** *Task:* Given a relational database schema and a natural-language question, generate a SQL query that answers the question. *Data:* The development split of Spider (Yu et al., 2018), a large-scale text-to-SQL dataset. *Metric:* Execution accuracy (whether the generated SQL query, when run against a test database, produces the same results as the ground-truth SQL query). *Base LM:* Llama 3.1 8B-Instruct. *Grammar:* We use the SQL context-free grammars released by Roy et al. (2024), which enforce valid SQL syntax and also schema-specific constraints that limit table and column names to those present in the given schema. *Semantic Potential:* Check whether column names in the generated (partial) query actually belong to the queried tables, modulo aliasing. (The grammar ensures only that the column names exist in *some* table.)
- **Molecular synthesis (GDB-17).** *Task:* Generate drug-like molecules in the SMILES format (Weininger, 1988). *Data:* Few-shot prompts constructed by repeatedly choosing 20 random examples from the GDB-17 dataset (Ruddigkeit et al., 2012). *Metric:* Quantitative Estimate of Drug-likeness (QED; Bickerton et al., 2012), a standard molecular fitness function implemented in the Python RDKit library (Landrum et al., 2024); additional molecular properties of interest are reported in App. E.2. *Base LM:* Llama 3.1 8B. *Grammar:* SMILES syntax for molecules. *Semantic Potential:* A SMILES prefix validator implemented in the Python *partialsmiles* library (O'Boyle, 2024), which checks syntax, atom valences, and the ability to kekulize aromatic systems.

3.2 Evaluation of Downstream Performance

We begin by investigating whether our approach leads to significant performance gains. Table 2 reports posterior-weighted accuracy for our approach and ablations of its components: grammar constraints, weight corrections, semantic potentials, and resampling. We summarize four key results:

Grammar constraints. In line with previous literature (e.g., Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Wang et al., 2024), we find that the addition of a grammar constraint ϕ_{CFG} improves downstream accuracy relative to the base LM across all domains in which it is used, even without the use of weight corrections.

Semantic potentials. Furthermore, we observe that integrating the semantic potential ϕ_{sem} improves accuracy in models with and without any weight corrections. In the latter case, the improvement in the goal inference, data science and molecular synthesis domains is large; in the text-to-SQL domain, it is smaller but statistically significant (paired permutation test, $p < 0.01$). This suggests that making use of information that cannot be efficiently encoded in logit masks can greatly improve performance, even without the use of weight corrections.

Weight corrections. Although the use of ϕ_{CFG} and ϕ_{sem} alone lead to significant gains in downstream accuracy, these gains can be amplified with addition of weight corrections. In cases without the semantic potential, weight corrections provide significant albeit relatively small gains in accuracy across three domains; in goal inference it does not significantly affect performance. In the presence

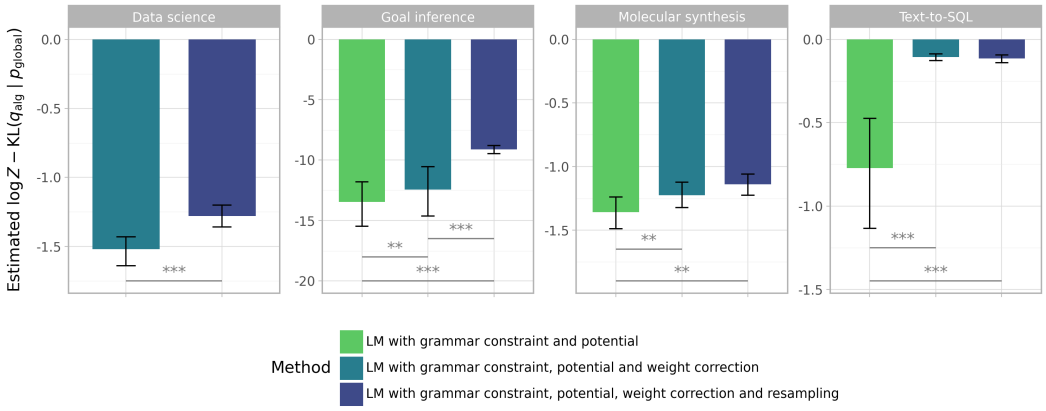

Figure 2: Estimated KL between the algorithm and the global product of experts, for a representative problem instance in each domain. Values closer to 0 indicate that the algorithm is better at approximating the global product of experts $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$. Significant differences are indicated with ** for $p < 0.01$ and *** for $p < 0.001$ (t-test). Algorithms are run with $N = 10$ particles.

of the semantic potential, adding weight corrections improves accuracy for text-to-SQL, and has no effect in goal inference and molecular synthesis. Overall, these results indicate that debiasing samples from a local product of experts to correctly target the global product of experts can significantly improve downstream accuracy. That said, the accuracy gains attributable to weight corrections are modest compared to other components of the algorithm, which suggests that the bias from locally constrained decoding may be less severe in these semantic parsing domains than has been observed in other domains (e.g. constrained generation of natural language (Lew et al., 2023)).

Resampling. Finally, we observe that the addition of resampling steps improves downstream accuracy in all domains except text-to-SQL, in which they neither significantly improve nor hurt performance. These results motivate adaptively focusing computation on promising partial sequences.

3.3 Validation of the Probabilistic Perspective

The best-performing methods from the previous section were designed to approximate the global product of experts distribution $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$. In this section, we investigate how closely each of these methods approximates this global distribution, and whether the quality of the probabilistic inference correlates with the downstream performance results from the previous section. In particular, we investigate several questions:

How closely do different methods approximate the global product of experts? We consider the distribution over sequences $q^{\text{r}}_{\text{alg}}(\boldsymbol{x})$ defined by each algorithm (see Appendix D for details and derivations).[1] For each $q^{\text{r}}_{\text{alg}}$, we computed an estimate of $\log Z - D_{\text{KL}}(q^{\text{r}}_{\text{alg}} \parallel [p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}])$. We refer to this quantity as the *approximation quality*. Since the term $\log Z$ is algorithm-independent, we can directly compare the estimated approximation quality across algorithms to determine which ones have lower KL divergence relative to the global product of experts. However, because $\log Z$ is instance-specific, these comparisons can only be made at the instance level. Accordingly, for each domain, we select the instance with the median unique accuracy as a representative example. Figure 2 visualizes estimated approximation quality on these examples across the three methods which include ϕ_{sem}: (i) Sample-Rerank; (ii) Full IS (iii) Full SMC.[2] This set of methods can be viewed as the cumulative addition of weight corrections and resampling steps to the language model with grammar constraint and semantic potential. Estimates were computed across 100 runs of each algorithm.

[1] More precisely, $q^{\text{r}}_{\text{alg}}$ is the marginal distribution of a single particle chosen proportionally to the particle weights after the algorithm alg is run. If all weights are zero, $q^{\text{r}}_{\text{alg}}$ reruns the algorithm until at least one weight is non-zero, and selects from the resulting particle set.

[2] Note that we do not compute Sample-Rerank for DS1000 since this method is equivalent to Full IS when $\phi_{\text{CFG}}(\boldsymbol{x}) = 1$.

Table 3: Pearson correlation between relative particle weights and accuracy scores for all weighted methods. Greater correlation indicates that relative weights are better associated with downstream performance (closer to 1 is better).

Method	Correlation between relative weight and score			
	Goal inference	Molecular synthesis	Data science	Text-to-SQL
LM with grammar constraints and correction (Grammar-Only IS)	0.138 (0.10, 0.18)	0.218 (0.16, 0.28)	0.217 (0.18, 0.26)	0.810 (0.79, 0.83)
LM with grammar constraints, potential, and correction (Full IS)	0.677 (0.64, 0.71)	0.570 (0.53, 0.61)	0.289 (0.25, 0.33)	0.796 (0.78, 0.81)
LM with grammar constraints, potential, correction, and resampling (Full SMC)	0.793 (0.76, 0.82)	0.826 (0.81, 0.84)	0.370 (0.31, 0.42)	0.810 (0.79, 0.83)

We summarize two key observations. First, we find that sampling from the local distribution can hurt approximation quality. In each domain, sampling from the local product of experts $[p_{\text{LM}} \odot \phi_{\text{CFG}}] \otimes \phi_{\text{sem}}$ without weight correction leads to significantly lower approximation quality relative to the methods which sample from the global product $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$. Second, we find that resampling steps can improve approximation quality. The addition of resampling steps significantly improves approximation quality in the data science and goal inference domains, and does not significantly improve nor hurt quality in the molecular synthesis and text-to-SQL domains.

Does approximation quality predict downstream accuracy? These trends in approximation quality are consistent with those observed in our evaluation of downstream accuracy. For example, we find that text-to-SQL is the domain in which weight corrections led to the most significant improvement in approximations of the global posterior, as well as the domain in which weight corrections most improve downstream performance. This suggests that framing constrained semantic parsing as a probabilistic inference problem (of sampling from the global products of experts distribution) is reasonable even when we are only interested in task-specific performance metrics that the probabilistic framing does not explicitly reflect. Furthermore, in some domains, these benefits extend beyond our main performance metric; for instance, resampling during molecular generation yields simultaneous improvements along a number of additional dimensions of interest, including de-novo similarity and diversity (Fig. 3).

Are higher-probability semantic parses (under $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$) more likely to perform well? In each of our experiments, we group output particles by *semantic equivalence*, and estimate the probability of each equivalence class under the method's approximation to the global product of experts, by summing the normalized weights of the members of each equivalence class (this is similar to the postprocessing performed in Shi et al. (2022)). We then measure the correlation between estimated probability of a result and its score on the task-specific metric.

Table 3 shows sequential Monte Carlo overall exhibits high correlation between (approximate) posterior probabilities and downstream performance, and that the differences in correlation between methods closely track the differences in performance in §3.2: in the goal inference, molecular synthesis, and data science domains, where semantic potentials and resampling greatly increase performance, we find that the same features also result in higher correlation between result probability and performance, whereas in text-to-SQL, where the performance gains are slimmer, we find that all methods correlate weight and score equally well. Together, these results validate the probabilistic approach, suggesting that the global posterior captures semantically meaningful uncertainty.

4 Related Work

Our contributions are primarily situated among two bodies of work. First, there is a large body of work leveraging LMs for semantic parsing or code generation tasks, while forcing adherence to a grammar or other constraints (Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Shin & Van Durme, 2022; Geng et al., 2023; Zheng et al., 2023a; Moskal et al., 2024; Wang et al., 2024; Ugare et al., 2024). Closely related is a series of algorithmic advances that enable the efficient construction and application of grammar constraints for sequential inference problems via compilation to automata (Deutsch et al., 2019; Willard & Louf, 2023; Kuchnik et al., 2023; Koo et al., 2024). Second, there is a large body of work which aims to generate from LMs subject to hard or soft constraints, including approaches based in reinforcement learning (RL) (Ziegler et al., 2019; Stiennon et al., 2020; Bai et al., 2022; Ouyang et al., 2022), classifier-guided control (Cheng et al., 2024), efficient probabilistic inference through tractable proxy models (Zhang et al., 2023), and locally-applied *logit biasing* or *masking* based on domain-specific potential functions (Pascual et al., 2021; Huang et al., 2024).

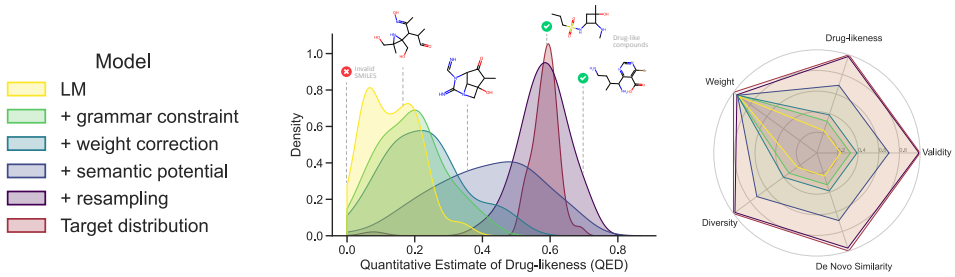

Figure 3: Visualization of posterior distributions for key molecular property indexes. **Middle:** Qualitative Estimate of Drug-likeness (QED), used as the main evaluation metric in Tab. 2. **Right:** Additional metrics of interest *Validity* (proportion of valid SMILES), *Weight* (exact molecular weight), *De Novo Similarity* (average pairwise Tanimoto similarity to the target distribution, excluding exact duplicates), and *Diversity* (inverse average pairwise Tanimoto similarity among compounds generated by a particular method). Of the generation methods evaluated, Full SMC achieves near-100% validity and most closely matches the target data distribution (GDB-17).

In this work, our algorithms aim to unify and improve on each of these bodies of preceding work, tackling semantic parsing and code generation tasks with a combination of grammar constraints (ϕ_{CFG}), arbitrary semantic potentials (ϕ_{sem}), and asymptotically correct inference (via SMC).

In the space of conditional generation via approximate posterior inference through sampling, there are a few neighboring efforts. Lin & Eisner (2018) trained an SMC proposal distribution for a globally normalized neural tagger. Lew et al. (2023) proposed SMC steering of LMs, enabling provably accurate posterior sampling from conditional densities while only ever computing local constraints. This work forms the basis of the SMC algorithms designed and implemented in this paper. Shortly thereafter, Zhao et al. (2024) independently developed a framework for steering LMs via SMC, including the use of learned twist functions that act as intermediate targets. In contrast to their work, we leverage incremental static and dynamic analysis to guide compute to promising partial sequences during generation, as opposed to undergoing a costly contrastive fine-tuning procedure. Concurrent with our work, Park et al. (2024) have highlighted the distinction between *locally-constrained* vs *globally-aligned* decoding from LMs subject to grammar constraints. The authors presented an iterative inference algorithm with a slow-converging approximation to the true posterior. This approach was also optimized for CFG constraints, as opposed to an arbitrary broader collection of multiple intersecting potentials.

An extended and thorough discussion of these bodies of work can be found in Appendix F.

5 Discussion

The experiments in this paper show that it is possible to obtain significant improvements in controlled generation quality by wrapping language model generations in global probabilistic inference algorithms that account for varied constraints. Furthermore, the experiments demonstrate that generation quality is correlated with the probability of generation under our methods, and methods that better approximate the global product of experts also have better downstream performance.

Our approach offers a modular way to improve performance across a broad class of controlled generation problems, especially where task-specific structure can be captured in semantic potentials. One possible source of potentials in many domains is the static analyses implemented efficiently in *language servers*, which IDEs use to incrementally detect problems in code as users type.

This paper has not extensively explored the auxiliary benefits of taking a principled probabilistic approach to constrained generation. As one example, the weights that are maintained and updated by our SMC algorithm are unbiased estimates of the global product of experts' normalizing constant, which can be interpreted as the probability that the *unconstrained* language model would have happened to satisfy all the constraints. This could be a proxy for whether a problem instance is in- or out-of-distribution for a model. Using such probabilistic quantities (also including, e.g., posterior uncertainty over parses), we may be able to build more rational systems, that ask clarifying questions or defer to larger models when there is reason to believe it may help.

REFERENCES

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl | the planning domain definition language. 1998.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Afra Amini, Tim Vieira, and Ryan Cotterell. Variational best-of-n alignment. *arXiv preprint arXiv:2407.06057*, 2024.

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

Martin Berglund, Willeke Martens, and Brink van der Merwe. Constructing a BPE tokenization DFA. In *International Conference on Implementation and Application of Automata*, pp. 66–78. Springer, 2024.

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012.

Kris Cao and Laura Rimell. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2104–2114, 2021.

Sourav Chatterjee and Persi Diaconis. The sample size required in importance sampling. *The Annals of Applied Probability*, 28(2), April 2018. doi: 10.1214/17-aap1326.

Emily Cheng, Marco Baroni, and Carmen Amo Alonso. Linearly controlled language generation with performative guarantees. *arXiv preprint arXiv:2405.15454*, 2024.

Nadezhda Chirkova, Germán Kruszewski, Jos Rozen, and Marc Dymetman. Should you marginalize over possible tokenizations? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–12, 2023.

Nicolas Chopin, Omiros Papaspiliopoulos, et al. *An introduction to sequential Monte Carlo*, volume 4. Springer, 2020.

Daniel Deutsch, Shyam Upadhyay, and Dan Roth. A general-purpose algorithm for constrained sequential inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 482–492, 2019.

Jay Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie Mellon University, 1968.

Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Daniel Flam-Shepherd, Kevin Zhu, and Alán Aspuru-Guzik. Language models can learn complex molecular distributions. *Nature Communications*, 13(1):3293, 2022.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

R. Howey, D. Long, and M. Fox. Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 294–301, 2004. doi: 10.1109/ICTAI.2004.120.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. Grounded decoding: Guiding text generation with grounded models for embodied agents. *Advances in Neural Information Processing Systems*, 36, 2024.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. In *First Conference on Language Modeling*, 2024.

Tomasz Korbak, Ethan Perez, and Christopher Buckley. RL with KL penalties is better viewed as Bayesian inference. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1083–1091, 2022.

Michael Kuchnik, Virginia Smith, and George Amvrosiadis. Validating large language models with relm. *Proceedings of Machine Learning and Systems*, 5:457–476, 2023.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Greg Landrum et al. RDKit: Open-source cheminformatics. https://github.com/rdkit/rdkit, 2024.

Alexander K Lew, Marco Cusumano-Towner, and Vikash K Mansinghka. Recursive Monte Carlo and variational inference with auxiliary variables. In *Uncertainty in Artificial Intelligence*, pp. 1096–1106. PMLR, 2022.

Alexander K Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash Mansinghka. Sequential Monte Carlo steering of large language models using probabilistic programs. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*, 2023.

Chu-Cheng Lin and Jason Eisner. Neural particle smoothing for sampling from conditional sequence models. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 929–941, New Orleans, June 2018. doi: 10.18653/v1/N18-1085. URL https://aclanthology.org/N18-1085/.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Daniel Lundén, David Broman, Fredrik Ronquist, and Lawrence M Murray. Automatic alignment of sequential monte carlo inference in higher-order probabilistic programs. *arXiv preprint arXiv:1812.07439*, 2018.

Michal Moskal, Madan Musuvathi, and Emre Kıcıman. AI Controller Interface. https://github.com/microsoft/aici/, 2024.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

version 1.0. Available from O'Boyle, N.M. partialsmiles. partialsmiles. N.M.O'Boyle.partialsmiles,version1.0.Availablefromhttps://github.com/baoilleach/partialsmiles, 2024.

Theo X Olausson, Alex Gu, Benjamin Lipkin, Cedegao E Zhang, Armando Solar-Lezama, Joshua B Tenenbaum, and Roger Levy. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. *arXiv preprint arXiv:2310.15164*, 2023.

João CA Oliveira, Johanna Frey, Shuo-Qing Zhang, Li-Cheng Xu, Xin Li, Shu-Wen Li, Xin Hong, and Lutz Ackermann. When machine learning meets molecular synthesis. *Trends in Chemistry*, 4 (10):863–885, 2022.

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

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. *arXiv preprint arXiv:2405.21047*, 2024.

Damian Pascual, Beni Egressy, Clara Meister, Ryan Cotterell, and Roger Wattenhofer. A plug-and-play method for controlled text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3973–3997, 2021.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022.

Subhro Roy, Samuel Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. Benchclamp: A benchmark for evaluating language models on syntactic and semantic parsing. *Advances in Neural Information Processing Systems*, 36, 2024.

Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, 2021.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I Wang. Natural language to code translation with execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3533–3546, 2022.

Richard Shin and Benjamin Van Durme. Few-shot semantic parsing with language models trained on code. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5417–5425, 2022.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen Jr, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7699–7715, 2021.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2), 1995.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of LLMs should leverage suboptimal, on-policy data. In *Forty-first International Conference on Machine Learning*, 2024.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *arXiv preprint arXiv:2408.02442*, 2024.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. Improving LLM code generation with grammar augmentation. *arXiv preprint arXiv:2403.01632*, 2024.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. Grammar prompting for domain-specific language generation with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Brandon T Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv e-prints*, pp. arXiv–2307, 2023.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. *arXiv preprint arXiv:2312.08566*, 2023.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. In *Forty-first International Conference on Machine Learning*, 2024.

Lance Ying, Katherine M Collins, Megan Wei, Cedegao E Zhang, Tan Zhi-Xuan, Adrian Weller, Joshua B Tenenbaum, and Lionel Wong. The neuro-symbolic inverse planning engine (nipe): Modeling probabilistic social inferences from linguistic inputs. In *First Workshop on Theory of Mind in Communicating Agents*, 2023.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425.

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. Tractable control for autoregressive language generation. In *International Conference on Machine Learning*, pp. 40932–40945. PMLR, 2023.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. Proc2pddl: Open-domain planning representations from texts. *arXiv preprint arXiv:2403.00092*, 2024.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Forty-first International Conference on Machine Learning*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs, 2023a.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, et al. Secrets of rlhf in large language models part i: Ppo. *arXiv preprint arXiv:2307.04964*, 2023b.

Tan Zhi-Xuan, Lance Ying, Vikash Mansinghka, and Joshua B Tenenbaum. Pragmatic instruction following and goal assistance via cooperative language-guided inverse planning. *arXiv preprint arXiv:2402.17930*, 2024.

Banghua Zhu, Michael Jordan, and Jiantao Jiao. Principled reinforcement learning with human feedback from pairwise or k-wise comparisons. In *International Conference on Machine Learning*, pp. 43037–43067. PMLR, 2023.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L Littman, and Stephen H Bach. Planetarium: A rigorous benchmark for translating text to structured planning languages. *arXiv preprint arXiv:2407.03321*, 2024.

Table 4: Downstream accuracy of different method using a smaller base language model (Llama 3.1 8B in Data science and Llama 3.2 1B in all other domains). Errors are bootstrapped 95% confidence intervals. Instruct model is used for Text-to-SQL.

Method	Score			
	Goal inference	Molecular synthesis	Data science	Text-to-SQL
LM	0.012 (0.01, 0.02)	0.032 (0.02, 0.04)	0.114 (0.09, 0.14)	0.224 (0.207, 0.241)
w/ grammar constraint (Locally-constrained Decoding)	0.046 (0.03, 0.06)	0.031 (0.02, 0.04)	-	0.250 (0.232, 0.270)
w/ grammar constraint, weight correction (Grammar-only IS)	0.037 (0.02, 0.06)	0.041 (0.03, 0.05)	-	0.301 (0.281, 0.323)
w/ grammar constraint, potential (Sample-Rerank)	0.087 (0.06, 0.12)	0.119 (0.09, 0.16)	-	0.299 (0.278, 0.321)
w/ grammar constraint, correction, and resampling (Grammar-only SMC)	0.052 (0.03, 0.08)	0.050 (0.04, 0.06)	-	0.302 (0.281, 0.324)
w/ grammar constraint, potential, and correction (IS)	0.079 (0.05, 0.11)	0.122 (0.09, 0.16)	0.225 (0.19, 0.26)	**0.348** (0.326, 0.372)
w/ grammar constraint, potential, correction, and resampling (SMC)	**0.125** (0.09, 0.16)	**0.517** (0.48, 0.55)	**0.285** (0.24, 0.34)	**0.348** (0.325, 0.374)

A Additional experiments

A.1 Smaller Base LMs

This section evaluates downstream accuracy across methods using smaller base language models. For the Text-to-SQL, Molecular Synthesis and Goal Inference domains, which in the §3.2 experiments used Llama 3.1 (8B), we substitute Llama 3.2 (1B). In the Data Science domain, which used Llama 3 (70B) in the §3.2 experiments, we substitute Llama 3.1 (8B). All experiments were run with $N = 10$ particles, and the instruct version of Llama 3.2 (1B) was used in the text-to-SQL domain to remain consistent with the model variants used in the main paper.

We report posterior weighted accuracy using the smaller LMs across all methods and domains in Tab. 4. Although accuracy is significantly lower compared to the larger LMs, we find that weight corrections, semantic potentials and resampling steps still improve model performance. Most interestingly, we also find that, in general, the relative gains in accuracy provided by our method are more pronounced for smaller language models. With the exception of Text-to-SQL, we observe that our approach with the smaller LM outperforms the locally-constrained decoding baseline (LM w/ grammar constraint) using the larger LM (see Tab. 2). In the Data Science domain, our Full SMC approach with the smaller LM outperforms the larger base LM. These results suggest that our approach can dramatically improve the performance of smaller LMs.

A.2 Accuracy by number of particles

This section investigates how performance improvements vary with the number of particles. Tab. 5 reports downstream accuracy for $N = 5$, $N = 10$ and $N = 50$ particles using the Llama 3.1 (8B) models. Note that we only include methods in which samples are generated from an approximate posterior that is constructed from a set of importance-weighted particles. For the base LM and locally constrained decoding baselines, samples are generated through direct ancestral sampling. As a result, the number of particles does not influence accuracy in these cases (though additional particles can provide a better estimate of the true model accuracy), so we omit these methods from the analysis.

We observe two patterns of results. In the Text-to-SQL and Molecular synthesis domains, increasing the number of particles has a marginal impact on downstream accuracy. However, in Goal inference and Data Science, we observe that a greater number of particles can lead to significantly better downstream accuracy (though only when increasing from 5 to 10 particles in the Data Science domain). These results indicate that the effect of the number of particles on downstream accuracy is dependent on the task. Future work will seek to characterize how properties of the task setup (e.g., the relationship between the (incremental) target distributions and the evaluation metric) affect the number of particles needed to achieve reasonable accuracy.

A.3 Resampling without replacement (Lew et al., 2023)

This section evaluates our approach using the without-replacement resampling method introduced in Lew et al. (2023). Specifically, we replace multinomial resampling steps with Lew et al. (2023)'s without replacement scheme in the full SMC algorithm with semantic potential (LM w/ grammar constraint, potential, correction, and resampling). For comparison, we ran the SMC steer baseline with $N = 5$ particles and a beam size of 3, alongside our approach using multinomial resampling

Table 5: Accuracy by number of particles across methods. Errors are bootstrapped 95% confidence intervals. Llama 3.1 8B is used as the base LM for all domains. Instruct model is used for Text-to-SQL.

Method	Score			
	Goal inference	Molecular synthesis	Data science	Text-to-SQL
5 Particles				
LM w/ grammar constraint, correction (Grammar-only IS)	0.106 (0.08, 0.14)	0.239 (0.21, 0.27)	-	0.587 (0.56, 0.61)
LM w/ grammar constraint, potential (Sample-Rerank)	0.214 (0.17, 0.26)	0.407 (0.36, 0.45)	-	0.578 (0.55, 0.60)
LM w/ grammar constraint, correction, and resampling (Grammar-only SMC)	0.310 (0.26, 0.37)	0.209 (0.18, 0.24)	-	0.599 (0.57, 0.62)
LM w/ grammar constraint, potential, and correction (Full IS)	0.216 (0.17, 0.27)	0.411 (0.37, 0.45)	0.204 (0.16, 0.25)	**0.611** (0.59, 0.63)
LM w/ grammar constraint, potential, correction, and resampling (Full SMC)	**0.319** (0.27, 0.37)	**0.552** (0.52, 0.58)	**0.224** (0.18, 0.27)	**0.620** (0.59, 0.64)
10 Particles				
LM w/ grammar constraint, weight correction (Grammar-only IS)	0.083 (0.06, 0.11)	0.228 (0.21, 0.25)	-	0.597 (0.57, 0.62)
LM w/ grammar constraint, potential (Sample-Rerank)	0.289 (0.24, 0.34)	0.392 (0.36, 0.42)	-	0.581 (0.56, 0.60)
LM w/ grammar constraint, correction, and resampling (Grammar-only SMC)	0.401 (0.34, 0.46)	0.205 (0.18, 0.23)	-	0.596 (0.57, 0.62)
LM w/ grammar constraint, potential, and correction (Full IS)	0.257 (0.21, 0.31)	0.404 (0.37, 0.44)	0.223 (0.19, 0.27)	**0.618** (0.59, 0.64)
LM w/ grammar constraint, potential, correction, and resampling (Full SMC)	**0.419** (0.37, 0.48)	**0.577** (0.56, 0.59)	**0.285** (0.26, 0.32)	**0.620** (0.60, 0.64)
50 Particles				
LM w/ grammar constraint, correction (Grammar-only IS)	0.069 (0.05, 0.09)	0.211 (0.20, 0.22)	-	0.603 (0.58, 0.63)
LM w/ grammar constraint, potential (Sample-Rerank)	0.416 (0.36, 0.47)	0.382 (0.37, 0.40)	-	0.585 (0.56, 0.61)
LM w/ grammar constraint, correction, and resampling (Grammar-only SMC)	0.595 (0.54, 0.65)	0.212 (0.20, 0.23)	-	0.599 (0.58, 0.62)
LM w/ grammar constraint, potential, and correction (Full IS)	0.393 (0.35, 0.45)	0.389 (0.38, 0.40)	0.218 (0.19, 0.25)	**0.626** (0.60, 0.66)
LM w/ grammar constraint, potential, correction, and resampling (Full SMC)	**0.611** (0.56, 0.66)	**0.569** (0.56, 0.58)	**0.292** (0.25, 0.33)	**0.622** (0.60, 0.65)

Table 6: Downstream accuracy comparison with the SMC Steering method from Lew et al. (2023) in the text-to-SQL domain. Errors are bootstrapped 95% confidence intervals. Both methods include semantic potentials. Our method is run with 10 particles. SMC Steering is run with 5 particles and a beam size of 3. Both methods are run with Llama 3.1 8B Instruct.

Method	Score
Full SMC	**0.620** (0.60, 0.64)
SMC Steering (Lew et al., 2023)	0.607 (0.58, 0.63)

with $N = 10$ particles (and an ESS threshold of 0.9). These settings effectively give the SMC steer method a particle count of $N = 15$, giving it an advantage in the comparison.

Tab. 6 reports posterior-weighted accuracy for these methods in the text-to-SQL domain (we restricted this analysis to a single domain because of limitations in computational resources). We observe that without replacement resampling steps slightly hurt performance compared to multinomial resampling.

A.4 Computational Cost

Though we have shown that practitioners can improve over locally-constrained decoding by using our proposed SMC method, in practice there is additional computational cost stemming from two sources: resampling and computing semantic potentials ϕ_{sem}. The cost of resampling is negligible, consisting only of simple sum, softmax, and categorical sampling operations at every token. The cost of computing semantic potentials, on the other hand, is more significant and varies across domains. Table 7 shows the average per token cost of computing semantic potentials for all of our domains,: we see that it rarely goes above about 30ms.

In general, the computational cost of semantic potentials is lessened by two factors: 1) semantic potentials often change not at every token, but only at larger, semantically meaningful units (for instance the end of a SQL clause or a python statement)—caching can therefore significantly lessen computational cost, 2) semantic potentials are often CPU rather than GPU computations (and so the cost of computation is much cheaper).

B Parser

We implement prefix parsing for Context-Free Grammars (CFGs) in order to enforce syntactic well-formedness in generated output sequences. Letting $\boldsymbol{y}$ refer to a sequence of terminal symbols from our CFG, writing $\boldsymbol{y}\circ$ to mean string $\boldsymbol{y}$ considered as as prefix, the prefix probability is given

Table 7: Average per token cost (in seconds) of computing the semantic potential ϕ_{sem} for each of our domains. Intervals are bootstrapped confidences estimated by selecting 10 SMC generations at random for each domain.

Method	Goal Inference	Molecular Synthesis	Data Science	Text-to-SQL
ϕ_{sem} seconds per token	0.011 (0.007, 0.016)	0.0003 (0.0002, 0.0004)	0.007 (0.0009, 0.023)	0.031 (0.0204, 0.0413)

by $p_{\text{CFG}}(\boldsymbol{y}\circ) = \sum_{\boldsymbol{y}\in\Sigma^*} p_{\text{CFG}}(\boldsymbol{y})\mathbf{1}[\boldsymbol{y} \in \text{prefix}(\boldsymbol{y})]$. Such prefix probabilities can be used to derive incremental (i.e., conditional) next-terminal distributions.

$$p_{\text{CFG}}(y_t \mid \boldsymbol{y}_{<t}) = \frac{p_{\text{CFG}}(\boldsymbol{y}\circ)}{p_{\text{CFG}}(\boldsymbol{y}_{<t}\circ)} \tag{8}$$

We implement an algorithm similar to the probabilistic, prefix-parsing variant of Earley's algorithm (Earley, 1968) given in Stolcke (1995). However, we note that properly achieving the integration between p_{LM} and ϕ_{CFG} is challenging due to the *token-terminal alignment problem*—the vocabulary of terminal symbols which forms the leaves of CFG derivation trees is not the same as the vocabulary of tokens output by the large language model. To address this problem, we extend our prefix parser such that it computes the conditional probability of next-characters rather than next terminals ($p_{\text{CFG}}(c \mid \boldsymbol{y}_{<t})$ where c is a character).

C Proposal distribution

We use the character-level prefix parser described in the previous section to develop a *character-based proposal* algorithm. This algorithm proposes tokens from $\mathcal{A}$ by sampling sequences of characters, and approximately, but asymptotically correctly, samples from the distribution on next tokens given as the product of p_{LM} and ϕ_{CFG} using the weight correction techniques for auxiliary randomness presented in Lew et al. (2022). This section describes this approach.

C.1 Proposal framework

Let p denote the distribution on next tokens targeted in inference. Our character-based proposal is a part of a general framework for proposal distributions which propose a token $x \in \mathcal{A}$ according to the following procedure, a variant of the Horvitz-Thompson estimator (Horvitz & Thompson, 1952) that can be justified for use within SMC using the RAVI framework (Lew et al., 2022):

1. Sample a subset S of the token vocabulary according to a distribution q_S.
2. Compute the *unnormalized target probability* $\tilde{p}(x)$ of each token $x \in S$.
3. Compute the *local weight* $w'(x)$ of each token as $\frac{\tilde{p}(x)}{\Pr(x\in S)}$ where $\Pr(x \in S)$ is the *inclusion probability*—the probability that x ended up in the sampled set under q_S.
4. Renormalize the local weights of the tokens in S and sample one of them.
5. Set the importance weight equal to the sum of the weights $\sum_{x\in S} w'(x)$.

We motivate the weight computation in step 5 with reference to *proper weighting*. Given an unnormalized density $\tilde{p}(x)$ with normalizing constant Z, a sample (x, w) is properly weighted for $\tilde{p}$ if, for any function f,

$$\mathop{\mathbb{E}}_{(x,w)\sim q'}[f(x)\, w] = Z \mathop{\mathbb{E}}_{x\sim p}[f(x)] \tag{9}$$

where q' is a joint proposal over pairs (x, w). In our setting, it is possible to construct a proposal that is a properly weighted sampler for $\tilde{p}$ by introducing a distribution $\widehat{q}(S \mid x)$ that downweights each importance weight—specifically, $w(x, S) := \frac{p(x)\widehat{q}(S|x)}{q(x,S)}$. Under this weighting scheme, our proposal gives rise to a properly weighted sampler for $\tilde{p}$:

$$\mathop{\mathbb{E}}_{(x,w)\sim q'}[wf(x)] = \sum_x \sum_S q(x,S)\frac{\tilde{p}(x)\widehat{q}(S \mid x)}{q(x,S)}f(x) \tag{10}$$

$$= \sum_x \sum_S \tilde{p}(x)\widehat{q}(S \mid x)f(x) \tag{11}$$

$$= \sum_x \tilde{p}(x)f(x)\sum_S \widehat{q}(S \mid x) \tag{12}$$

$$= Z\sum_x p(x)f(x) \tag{13}$$

$$= Z \mathop{\mathbb{E}}_{x\sim p}[f(x)] \tag{14}$$

Note that for the derivation above to hold in general, the step Eq. (10) $\implies$ Eq. (11) requires $\forall x, S\colon p(x)\widehat{q}(S \mid x) > 0 \implies q(x,S) > 0$. See Lew et al. (2022, Appendix C) for more details.

Unfortunately, we cannot tractably compute the exact density for $q(S \mid x') = \frac{q(x'|S)q_S(S)}{\sum_{S'} q(x'|S')q_S(S')}$ since we cannot generally marginalize over S. Thus, in the spirit of Lew et al. (2022), we use a meta-proposal $q_S(S \mid x' \in S)$ for S based on the idea that we sample an S subject to the constraint that it contain x'. Under the assumption that we can tractably compute $\Pr(x' \in S)$, we can compute the exact density for $q(S \mid x' \in S)$ by Bayes' rule $q_S(S \mid x' \in S) = \frac{q_S(S)\mathbf{1}_S(x')}{\Pr(x' \in S)}$. Then the importance weight attached to the returned x' is given as:

$$\frac{p(x')\tilde{q}_{x'}(S)}{q_S(S)q(x' \mid S)} = \tilde{p}(x')\frac{q_S(S)}{\Pr(x' \in S)}\left(q_S(S)\frac{\frac{\tilde{p}(x')}{\Pr(x' \in S))}}{\sum_x \frac{p(x)}{\Pr(x \in S)}}\right)^{-1} \tag{15}$$

$$= \tilde{p}(x')\frac{q_S(S)}{\Pr(x' \in S)}\left(\frac{\frac{\tilde{p}(x')q_S(S)}{\Pr(x' \in S))}}{\sum_x \frac{p(x)}{\Pr(x \in S)}}\right)^{-1} \tag{16}$$

$$= \sum_x \frac{\tilde{p}(x)}{\Pr(x' \in S)} \tag{17}$$

C.2 Character proposal

Our character proposal distribution is an instance of this framework in which q_S samples sets of tokens S by sampling a sequence of characters. We provide the pseudocode for this algorithm in Algorithm 1, and define two key data structures used by this proposal:

Definition 1. *Our **trie data structure** T is a labeled, tree-structured graph that is defined as follows:*

- *Let V be the LM's vocabulary of tokens (represented as strings of characters ending with a designated end-of-token marker* `EOT`*).*
- *Let P be the prefix closure of the set V: $P \stackrel{\text{def}}{=} \{\boldsymbol{p} \in \Sigma^* \mid \boldsymbol{p} \preceq t, t \in V\}$ where $\boldsymbol{p} \preceq t$ denotes that $\boldsymbol{p}$ is a prefix of t.*
- *Let $T = (N, E)$ be a labeled graph with node P and labeled edges $E = \left\{\boldsymbol{p} \xrightarrow{a} \boldsymbol{p}\,a \;\middle|\; \boldsymbol{p}, (\boldsymbol{p}\,a) \in P\right\}$*

Definition 2. *Let* `mass` *be a mapping $N \to [0,1]$, defined as follows:*

Each leaf node corresponds to an LLM token with `mass` *given by p in a specified context $\boldsymbol{t} \in V^*$:*

$$\texttt{mass}(t') = p(t' \mid \boldsymbol{t}), \qquad \text{for } t' \in V \tag{18}$$

Name	Proposal	Target	Inference alg.
Language model	p_{LM}	p_{LM}	Exact
w/ grammar constraint	$[p_{\text{LM}} \odot \phi_{\text{CFG}}]$	$[p_{\text{LM}} \odot \phi_{\text{CFG}}]$	Exact
w/ grammar constraint, weight correction	$[p_{\text{LM}} \odot \phi_{\text{CFG}}]$	$[p_{\text{LM}} \otimes \phi_{\text{CFG}}]$	IS
w/ grammar constraint, potential	$[p_{\text{LM}} \odot \phi_{\text{CFG}}]$	$[p_{\text{LM}} \odot \phi_{\text{CFG}}] \otimes \phi_{\text{sem}}$	IS
w/ grammar constraint, potential, and correction	$[p_{\text{LM}} \odot \phi_{\text{CFG}}]$	$[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$	IS
w/ grammar constraint, potential, correction, and resampling	$[p_{\text{LM}} \odot \phi_{\text{CFG}}]$	$[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$	SMC

Table 8: Potentials, Models, and Inference Algorithms

$$\texttt{mass}(\boldsymbol{p}) = \sum_{\substack{t' \in V \\ \text{s.t.}\, \boldsymbol{p} \prec t'}} p(t' \mid \boldsymbol{t}), \qquad \textit{for } \boldsymbol{p} \in P - V \tag{19}$$

$$= \sum_{\boldsymbol{p} \xrightarrow{a} \boldsymbol{p}\, a \in E} \texttt{mass}(\boldsymbol{p}\, a) \tag{20}$$

Here $\boldsymbol{p} \prec t$ denotes that $\boldsymbol{p}$ is a strict prefix of t.

Algorithm 1 Character proposal: This procedure returns sampling weights for possible next tokens, i.e., an element of $\mathbb{R}^V_{\geq 0}$.

procedure `character_proposal`($\boldsymbol{t}$)
  `mass` $\leftarrow$ Apply Eq. (20) to $p(\cdot \mid \boldsymbol{t})$
  $\boldsymbol{p} \leftarrow \varepsilon$ ▷ *start at the trie's root node*
  `inclusion_prob` $\leftarrow 1$ ▷ *prefix's path probability*
  `weights` $\leftarrow \{\}$
  while true:
    $p_1 \leftarrow \left\{a : \frac{\texttt{mass}(\boldsymbol{p}\, a)}{\texttt{mass}(\boldsymbol{p})} \text{ for } \boldsymbol{p} \xrightarrow{a} \boldsymbol{p}\, a \in E\right\}$
    $p_2 \leftarrow \{a : p_{\text{CFG}}(a \mid \boldsymbol{t}\, \boldsymbol{p}) \text{ for } a \in \Sigma\}$
    if $\boldsymbol{p} \xrightarrow{\texttt{EOT}} \boldsymbol{p}\,\texttt{EOT} \in E$: ▷ *End-of-token available (i.e., $\boldsymbol{p}\,\texttt{EOT} \in V$*
      `weights`$(\boldsymbol{p}\,\texttt{EOT}) = \frac{\texttt{mass}(\boldsymbol{p}\,\texttt{EOT}) \cdot \texttt{local\_prob}}{\texttt{inclusion\_prob}}$
    $\overline{q} \leftarrow \{a : p_1(a) \cdot p_2(a) \text{ for } a \in \Sigma\}$ ▷ *Note: $\texttt{EOT} \notin \Sigma$.*
    $Q \leftarrow \sum_{a \in \Sigma} \overline{q}(a)$
    if $Q = 0$: ▷ *cannot continue further*
      break
    $a \sim \overline{q}/Q$ ▷ *Sample next character proportional to $\overline{q}$*
    $\boldsymbol{p} \leftarrow \boldsymbol{p}\, a$ ▷ *extend the prefix (i.e., transition to the next node)*
    `local_prob` $\leftarrow$ `local_prob` $\cdot\, p_2(a)$
    `inclusion_prob` $\leftarrow$ `inclusion_prob` $\cdot\, \overline{q}(a)/Q$
  return `weights`

D Estimating inference quality

D.1 IS and SMC

A standard property of importance sampling is that log of the mean importance weight provides a biased estimate of the KL between the algorithm and the target distribution.

Definition 3 (Extended-state space IS)**.** Let $\mathbf{S} := \{\boldsymbol{x}, k, \boldsymbol{x}_{-k}\}$ denote an extended state-space of a resampled particle $\boldsymbol{x}$, its index k in the particle beam, and the $K-1$ non-resampled particles. We interpret an IS algorithm with proposal q targeting σ as a proposal distribution

$$q_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k}) = \frac{w(\boldsymbol{x}^k)}{\sum_{i=1}^K w(\boldsymbol{x}^i)} \prod_{i=1}^K q(\boldsymbol{x}^i)$$

which targets the distribution

$$\sigma_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k}) = \sigma(\boldsymbol{x}_{-k}, k \mid \boldsymbol{x})\sigma(\boldsymbol{x}), \text{ where } \sigma(\boldsymbol{x}_{-k}, k \mid \boldsymbol{x}) := \frac{1}{K}\prod_{i \neq k} q(\boldsymbol{x}^i)$$

Both distributions are defined over $\mathbf{S}$.

We now show that under the formulation, we can use the log of the mean importance weights of our algorithm as an estimate of $\log Z_\sigma - D_{\text{KL}}(q_{\text{IS}} \parallel \sigma)$.

Proposition 1 (Estimating $\log Z_\sigma - D_{\text{KL}}(q_{\text{IS}} \parallel \sigma)$)**.** Let $w(\boldsymbol{x}) = \sigma(\boldsymbol{x})/q(\boldsymbol{x})$ be the importance weight of a particle $\boldsymbol{x}$ under our IS algorithm. Under the extended state-space formulation of IS, $\log \frac{1}{K}\sum_{k=1}^{K}\left(\frac{w(\boldsymbol{x}^k)}{\sum_{i=1}^{K} w(\boldsymbol{x}^i)}\right)$ provides a biased estimate of $\log Z_\sigma - D_{\text{KL}}(q_{\text{IS}} \parallel \sigma)$.

proof

$$\mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{\tilde{\sigma}_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}{q_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}\right] = -\mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{q_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}{\tilde{\sigma}_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}\right] \tag{21}$$

$$= -\mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{q_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}{\sigma_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k}) \cdot Z_\sigma}\right] \tag{22}$$

$$= \log Z_\sigma - \mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{q_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}{\sigma_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}\right] \tag{23}$$

$$= \log Z_\sigma - \mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{q_{\text{IS}}(k, \boldsymbol{x}_{-k} \mid \boldsymbol{x})q_{\text{IS}}(\boldsymbol{x})}{\sigma_{\text{IS}}(k, \boldsymbol{x}_{-k} \mid \boldsymbol{x})\sigma(\boldsymbol{x})}\right] \tag{24}$$

$$= \log Z_\sigma - \mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{q_{\text{IS}}(k, \boldsymbol{x}_{-k} \mid \boldsymbol{x})}{\sigma_{\text{IS}}(k, \boldsymbol{x}_{-k} \mid \boldsymbol{x})}\right] - \mathop{\mathbb{E}}_{q_{\text{IS}}}\left[\log \frac{q_{\text{IS}}(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right] \tag{25}$$

$$= \log Z_\sigma - D_{\text{KL}}(q_{\text{IS}}(k, \boldsymbol{x}_{-k} \parallel \boldsymbol{x}) \mid \sigma_{\text{IS}}(k, \boldsymbol{x}_{-k} \mid \boldsymbol{x})) - D_{\text{KL}}(q_{\text{IS}} \parallel \sigma) \tag{26}$$

where

$$\frac{\tilde{\sigma}_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})}{q_{\text{IS}}(\boldsymbol{x}, k, \boldsymbol{x}_{-k})} = \frac{\frac{1}{K}\tilde{\sigma}(\boldsymbol{x})\prod_{i \neq k} q(\boldsymbol{x}^i)}{q(\boldsymbol{x})\frac{w(\boldsymbol{x}^k)}{\sum_{i=1}^{K} w(\boldsymbol{x}^i)}\left(\prod_{i \neq k} q(\boldsymbol{x}^i)\right)} \tag{27}$$

$$= \frac{1}{K} w(\boldsymbol{x}) \frac{\sum_{i=1}^{K} w(\boldsymbol{x}^i)}{w(\boldsymbol{x})} \tag{28}$$

$$= \frac{1}{K}\sum_{i=1}^{K} w(\boldsymbol{x}^i) \tag{29}$$

Hence, $\log\left(\frac{1}{K}\sum_{i=1}^{K} w(\boldsymbol{x}^i)\right)$ provides a single sample estimate of $\log Z_{\sigma_{\text{IS}}^{\text{gpoe}}} - D_{\text{KL}}(q_{\text{IS}} \parallel \sigma_{\text{IS}}^{\text{gpoe}})$ with bias $D_{\text{KL}}(q_{\text{IS}}(k, \boldsymbol{x}_{-k} \parallel \boldsymbol{x}) \mid \sigma(k, \boldsymbol{x}_{-k} \mid \boldsymbol{x}))$.

The same logic as above can be applied to SMC: Appendix F in Zhao et al. (2024) explains the standard extended state-space construction for SMC. Using this construction, the extended-space importance ratio between σ_{SMC} and q_{SMC} is exactly the average particle weight in the final particle collection returned by the algorithm, just as in IS, and the expected log average particle weight is (by the same logic as above) equal to $\log Z - D_{\text{KL}}(q_{\text{SMC}} \parallel \sigma_{\text{SMC}})$.

D.2 Estimating inference quality for rejection-sampled variations of our algorithms

When attempting to estimate the discrepancy between samples for algorithms q_{alg} and the posterior $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$, one difficulty is that $D_{KL}(q_{\text{alg}}||[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}])$, is infinite for all the algorithms we consider in this work, as all of them make it possible (given a finite number of particles) to generate samples that have probability zero under $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$. A potential solution to

this issue, explored by Zhao et al. (2024), is to instead estimate $D_{KL}([p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]||q_{\text{alg}})$. But this requires exact samples from $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]$, which are impractical to obtain in our setting. We thus take a different approach and consider rejection-sampled versions of each of our algorithms, q^r_{alg}, which draw samples $\boldsymbol{x} \sim q_{alg}$ repeatedly until $[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}](\boldsymbol{x}) > 0$. In this case, we have

$$D_{\text{KL}}(q^r_{\text{alg}}||[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}]) = \mathbb{E}_{q^r_{\text{alg}}}\left[\log \frac{q^r_{\text{alg}}(\boldsymbol{x})}{[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}](\boldsymbol{x})}\right] \tag{30}$$

$$= \mathbb{E}_{q^r_{\text{alg}}}\left[\log \frac{q_{\text{alg}}(\boldsymbol{x})}{[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}](\boldsymbol{x}) Z^r_{alg}}\right] \tag{31}$$

$$= \mathbb{E}_{q^r_{\text{alg}}}\left[\log \frac{q_{\text{alg}}(\boldsymbol{x})}{[p_{\text{LM}} \otimes \{\phi_{\text{CFG}}, \phi_{\text{sem}}\}](\boldsymbol{x})}\right] - \log Z^r_{alg}, \tag{32}$$

where Z^r_{alg} is the acceptance rate of q^r_{alg} (which we can estimate with standard Monte Carlo), and we can estimate the first term in Eq. (32) up to an instance-specific constant, using the derivations from above.

E Domain details

E.1 Spider

Spider is a large-scale text-to-SQL dataset of natural language questions and database schemas; given a question and schema, the task is to generate a valid SQL query that is semantically equivalent to a ground-truth query. In this domain, ϕ_{CFG} is used to enforce valid SQL syntax according to the SQL grammars released by Roy et al. (2024). These grammars include schema-specific constraints that limit table and column names to those present in the given schema, but do not ensure correct table-column associations. Thus, we use ϕ_{sem} to verify whether the (partial) SQL query references a column that exists in a table, returning 0 in the case that it does not and 1 otherwise. Note that ϕ_{sem} is only semantically meaningful when a generated query has fully specified the necessary table or alias information to check correspondences (e.g., after the `FROM` clause is complete). We evaluate on the development split of Spider with execution accuracy, which checks whether the predicted SQL query's output matches that of the ground-truth query. We define $p_{\text{LLM}}(\boldsymbol{x})$ by prompting Llama-3.1 (8b) instruct with 3 examples followed by a rendering of the database and the natural language question. We sample 10 particles for all methods, and set an ESS threshold of 9 for SMC. This threshold is chosen to trigger a resampling step as soon as a particle is deemed invalid by ϕ_{slow}. We use a multinomial sampling scheme.

E.2 GBD-17

A recent line of work has applied LMs to the problem of molecular synthesis, with the aim of generating candidate molecules with properties similar to molecules from known databases (see Oliveira et al., 2022, for review)—most commonly (e.g. Flam-Shepherd et al. (2022); Wang et al. (2024)) by prompting with examples of molecules in SMILES format (Weininger, 1988). We follow this approach, constructing prompts from random subsets of 20 molecules from the GDB-17 dataset (Ruddigkeit et al., 2012). We evaluate generations using the standard molecule fitness function Quantitative-Estimated Drug-likeness (QED; Bickerton et al., 2012) implemented in the Python RDKit library (Landrum et al., 2024). This metric combines eight physicochemical properties of a compound: Molecular weight, LogP, H-bond donors, H-bond acceptors, Charge, Aromaticity, Stereochemistry, Solubility. Here, ϕ_{CFG} enforces SMILES syntax. To enforce properties not encoded by this syntax, we define ϕ_{sem}using a molecule validator that can be applied to partial SMILES strings, implemented in the Python *partialsmiles* library (O'Boyle, 2024). The validator checks the SMILES prefix to ensure that atom's valences are in a list of allowed valences, and attempts to find alternating patterns of single and double bonds to cover all aromatic systems in the partial string. The additional metrics reported in Fig. 3 are *Validity* (proportion of valid SMILES), *Weight* (exact molecular weight), *De Novo Similarity* (average pairwise Tanimoto similarity to the target

Table	Columns
`singer`	`singer_id`, `name`, ...
`concert`	`concert_id`, `concert_name`, ...

(a) Example schema

Query	ϕ_{CFG}	ϕ_{sem}	Description
`SELECT song_id FROM singer ...`	✗	✗	Invalid column name
`SELECT singer_id FROM concert ...`	✓	✗	Invalid column name for table
`SELECT singer_id FROM singer ...`	✓	✓	Valid column name for table

(b) Example queries and potential values

Table 9: Overview of potentials used in Spider experiments for a given schema. We condition the base language model using two potentials. ϕ_{CFG} ensures syntactically valid SQL queries that only include table and column names present in the schema. ϕ_{sem} further restricts SQL queries by ensuring a correct correspondence between column and table names.

distribution, excluding exact duplicates), and *Diversity* (inverse average pairwise Tanimoto similarity among compounds generated by a particular method).

E.3 PLANETARIUM

Recent work has explored using LMs for planning with languages like the Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998), by either generating plans directly (Silver et al., 2022; Wong et al., 2023; Ying et al., 2023; Zhang et al., 2024; Zhi-Xuan et al., 2024), or generating descriptions of a task's initial and/or goal conditions, which classical planning algorithms can use to search for plans (Liu et al., 2023; Xie et al., 2023; Guan et al., 2023). In the spirit of the latter, we use the Blocksworld tasks from the Planetarium benchmark (Zuo et al., 2024), which provides natural language descriptions of a task's initial and goal conditions along with their ground-truth symbolic representations in the STRIPS subset of PDDL (Fikes & Nilsson, 1971). The original dataset is extremely challenging, requiring the LM to output a full STRIPS description of tasks with up to 100 objects—Zuo et al. (2024) report fewer than 2% of the outputs of Gemma 1.1 7B to be even parseable. We therefore simplify the task by limiting our evaluation to examples with fewer than 10 objects, and requiring the LM to generate only the goal conditions given a description of initial conditions. Here, ϕ_{CFG} encodes STRIPS syntax for goals within Planetarium's Blocksworld domain definition; ϕ_{sem} uses a gold-standard plan known to satisfy the ground-truth task description, and calls the VAL plan validator (Howey et al., 2004) to test whether partial goal descriptions are valid according to that plan. The gold-standard plans for each instance were derived using the fast downward algorithm (Helmert, 2006). Note that it is only possible to apply this ϕ_{sem} potential to partial strings if goal descriptions are *monotonic*, that is, if any goal prefix describes a superset of the states that the full goal describes. This is the case for STRIPS, where goals must be described as conjunctions of literals, so that we can evaluate the potential after each literal in the conjuction is completed.

E.4 DS1000

DS-1000 (Lai et al., 2023) is a challenging code generation benchmark on data science problems in Python with problem solutions allowing the usage of a number of popular libraries, including NumPy, PyTorch, and Pandas. For each problem instance, the language model is prompted with an English description of the problem and a sample test case in Python and is tasked to generate code that solves the problem and passes the test case. Each test case includes a result variable, and success depends on the execution.

In preliminary experiments, we observed that our language model was able to generating syntactically correct Python programs for every sample. We, therefore, set $\phi_{\text{CFG}} = 1$ for our experiments in this domain. Thus, unlike the other three domains, the proposal distribution for all evaluations of DS1000 was simply p_{LM}.

In this domain, ϕ_{sem} simply executes the test cases provided in the prompts from Lai et al. (2023) on generated (partial) Python programs, and returns 1 if no errors are produced and 0 otherwise (in particular, we did not make use the output of test cases). Note that it is only possible to execute Python code when the generated sequence $\boldsymbol{x}$ consists entirely of well-formed Python statements, thus in this domain ϕ_{sem} can only be meaningfully applied at the boundary of statements—this motivates aligning SMC particles using statements as their steps, as explained in the "Further extensions" section in (§2).

F Extended Related Work

Grammar-constrained semantic parsing with LMs

Shin et al. (2021) presented a system allowing LMs to be locally intersected with (boolean) CFGs to restrict generations to conform to target formal languages, and that with only a few in-context examples, such an inference-time strategy could outperform more substantial fine-tuning. Concurrently, PICARD (Scholak et al., 2021) presented an approach for intersecting LMs with an incremental parsing algorithm, and showed how additional context-sensitive constraints could be imposed such as requiring table-column matching for SQL generation via the use of programmable "guards". Synchromesh (Poesia et al., 2022) generalized these frameworks and extended the idea of incremental guards that can impose semantic restrictions during generation—such as typing and scoping rules—by dynamically constructing constraints as regular expression on-the-fly. A great deal of other work has explored variants of LM-grammar intersection including the effectiveness of pre-training models on code for these settings (Shin & Van Durme, 2022), the runtime compilation of individual task instances into highly-specific, task-specialized grammars (Geng et al., 2023), and even using the LM to generate grammars directly at runtime, that then restrict their own generation to solve a task (Wang et al., 2024). Other work has focused more closely on the standard syntactic-constraint problem but with an emphasis on optimizing efficient data structures and algorithms for fast LM-CFG intersection (Ugare et al., 2024; Zheng et al., 2023a; Moskal et al., 2024).

A parallel line of work in this space has been concerned with the efficient construction and application of constraints for sequential inference problems. Deutsch et al. (2019) first noted that regular and context-free grammar constraints could be pre-compiled to automata—these could then be used during sequential inference to impose constraints with near-zero runtime overhead. This approach was independently developed and efficiently implemented in the context of restricting LM generations to regular expressions by the Outlines (Willard & Louf, 2023) and ReLM libraries (Kuchnik et al., 2023). Similar work was later developed by Koo et al. (2024), who extended several formal automata-theoretic characteristics of these constructions.

This work has noted the complications of efficiently intersecting grammars whose atoms are terminals and LMs whose atoms are tokens, which we refer to as the *token-terminal alignment problem*. An efficient and accurate solution to this problem space was one of several desiderata for our proposal algorithm (see Appendix 1 for more details). These works have also discussed considerations that arise in the construction of automata, whose arcs are tokens, in the assignment of probabilities to strings. Namely, there are exponentially many latent token trajectories that correspond to a generated string. While the correct method for assigning string probabilities involves marginalizing over these trajectories (Cao & Rimell, 2021), in practice simply using the *canonical tokenization* accounts for the overwhelming majority of the probability mass and can be well justified (Chirkova et al., 2023; Kuchnik et al., 2023; Berglund et al., 2024). In the present work, we do not enforce this assumption and allow all token trajectories.

Conditional generation subject to constraints

Language models pre-trained on a next-word objective reflect the distribution of their pre-training corpora, but often the inference-time needs of tasks necessitate that LMs modify this base distribution.

One approach to this class of problems is fine-tuning or reinforcement learning via some set of data that more closely mirrors the target task, such as via reinforcement learning from human feedback (RLHF) (Ziegler et al., 2019; Stiennon et al., 2020; Bai et al., 2022; Ouyang et al., 2022), but this method comes with challenges such as hyperparameter sensitivity and distributional collapse (Zheng et al., 2023b; Zhu et al., 2023; Xiong et al., 2024). Some of these drawbacks can be mitigated by utilizing on-policy data (Tajwar et al., 2024) and imposing a KL penalty that penalizes shifting an LM

too far from its prior distribution, casting optimization as a variational inference problem (Korbak et al., 2022; Amini et al., 2024).

Another inference-time approach to controlled generation for an LM is via direct modification to the LM's sampling distribution. This may be done via controlling intermediate layer activations with classifier guidance (Cheng et al., 2024), guiding autoregresssive generation with a proxy probabilistic model for which estimation of the conditional density is tractable (Zhang et al., 2023), or most commonly by directly intervening on the final logits before sampling to impose intersection with a potential function. Pascual et al. (2021) presented an early variant of such *logit-biasing* to encourage the presence of predefined guide words in generations.

This pattern is employed more broadly for hard constraints via *logit-masking*, setting the probability associated with particular tokens to zero, forcing the LM to sample from a subset of its distribution over sequences. This approach is used in most of the grammar-constrained semantic parsing work outlined in the previous section. Most recently, there have been attempts to restricting and re-weight generations not only via grammars but through additional semantic potentials such as grounded affordances in robotics settings (Ahn et al., 2022; Huang et al., 2024). However, in all of these works, constraints are imposed greedily, resulting in a local product of experts construction, and care is not taken to appropriately target the implied global product of experts. It should then come as no surprise that while standard approaches to grammar-constrained generation have been successful, they have been far from a silver bullet (Tam et al., 2024).

Approximate posterior inference via sampling

This leads to a third line of work that aims to formulate conditional generation subject to constraints as posterior inference, and employ approximate inference algorithms to appropriately sample from such global target distributions. Lew et al. (2023) propose SMC steering of LMs via probabilistic programming specifications. This work enables provably accurate posterior sampling from such conditional targets, globally steering generation while only ever computing local constraints. Our approach builds on the results in that paper.

Shortly thereafter, Zhao et al. (2024) independently developed a framework for expressing various LM tasks as probabilistic inference problems that can be tackled with SMC. Similar to our work, Zhao et al. (2024) guide SMC with intermediate targets—in their case learned twist functions via a novel contrastive method—that enable estimation of the expected future value of each candidate partial sequence. Their work also developed methods for evaluating LM inference algorithms via bi-directional bounds on the log-partition function that can be used to estimate the KL-divergence between the inference and target distribution. In contrast to this prior work, our approach to SMC leverages incremental static and dynamic analyses to inform our proposal distributions and twist functions, as opposed to learning components of these algorithms via a costly contrastive fine-tuning procedure. In addition, our results directly relate the quality of our posterior approximation to improved performance on a series of standard, difficult benchmark tasks.

Concurrent with our work, Park et al. (2024) have highlighted the distinction between the prevalent locally constrained decoding approach and the more accurate targeting of the global distribution that arises from combining language models with constraints. Park et al. (2024)'s approach to approximate the global distribution is based on the concept of *expected future grammaticality*, which is the probability that a string sampled from the LM is compliant with a given grammar. The authors describe an iterative algorithm that approximates the global distribution by refining the estimates of the expected future grammatically. However, the proposed strategy shows relatively slow convergence, was specifically designed for a CFG constraint, and may not be easily adaptable to constraining with multiple potential functions.