DRAFT-GUIDED CONSTRAINED DIFFUSION FOR STRUCTURED GENERATION IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Structured generation requires language models to produce outputs that are both task-correct and compliant with hard structural constraints, such as JSON schemas, often within a fixed output budget. Constrained decoding (CD) enforces such constraints in autoregressive (AR) models, but restricting the next-token distribution can alter the model's solution and may still leave the output incomplete when the generation budget is exhausted. We observe that template-based diffusion offers a complementary mechanism, however, using a diffusion language model (dLLM) directly still limits task accuracy to the dLLM's own problem-solving ability. We propose *Draft-Guided Constrained Diffusion* (DGCD), which separates these roles. An AR model first produces an unconstrained draft, and a dLLM then expresses its content within a fixed schema template. For feasible template-expressible schemas under our validity assumptions, DGCD guarantees structural validity by construction, while its task accuracy depends on faithfully preserving the draft's answer. Across six AR drafters from 1B to 14B on mathematical reasoning and structured extraction, DGCD achieves 100% observed structure adherence on every evaluated setting and improves average accuracy over draft-conditioned constrained decoding by up to 19.3 percentage points. DGCD also remains robust under tight output budgets and is $3.6\times$ faster than DCCD in our latency evaluation.

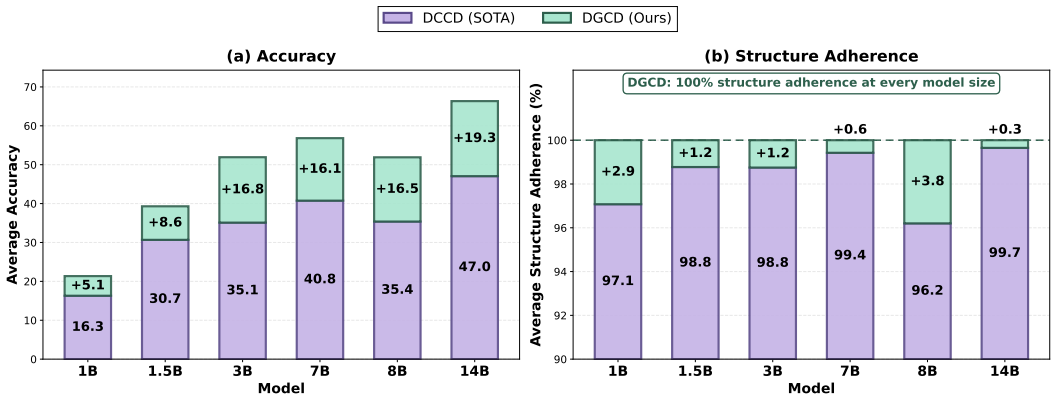


Figure 1: DGCD (ours) improves both accuracy and structure over the state of the art (Reddy et al., 2026) across model sizes from 1B to 14B. This figure shows averaged performance over mathematical reasoning (GSM8K, MATH500) and agentic extraction (Fin. Ent., PII). Purple bars show existing SoTA DCCD; green segments show the gain of proposed DGCD. (a) DGCD improves accuracy by 5.1–19.3 points. (b) DGCD achieves 100% structure adherence at every scale

1 INTRODUCTION

Large language models (LLMs) are increasingly embedded in agentic systems, where their outputs are consumed by programs, tools, and external environments (Liu et al., 2024). In such settings, semantic correctness alone is insufficient: an output must also satisfy a machine-readable structure,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

such as a JSON schema or an API specification. A correct result in an invalid format may be unusable, while a schema-valid response may still contain incorrect content. Structured generation, therefore, requires both task accuracy and reliable structural adherence, often within a fixed output budget. The central challenge is to enforce structure without reducing the model's ability to solve the task.

Existing approaches and the open challenge. Constrained decoding (CD) addresses the structural requirement by restricting autoregressive (AR) generation to schema-compatible continuations (Willard & Louf, 2023; Dong et al., 2025). Please refer to Appendix A for detailed context of related works. However, these restrictions change the model's generation distribution and can reduce task accuracy (Tam et al., 2024; Schall & De Melo, 2025), an effect we refer to as the *projection tax*. Moreover, standard prefix-based constraints do not ensure that the structure is completed within the available output budget. Providing an unconstrained draft before constrained formatting can mitigate the accuracy loss (Reddy et al., 2026), but the formatting stage still generates structural tokens autoregressively. Thus, separating solving from formatting addresses only part of the problem because the formatter remains exposed to token-level projection and budget exhaustion. This leaves a central question: *Can we produce complete, schema-valid outputs within a feasible output budget while retaining the task accuracy of unconstrained AR generation?*

Our key insight is that solving the task and enforcing the structure need not use the same generation mechanism. We note that masked diffusion language models (dLLMs) provide a useful opportunity because they can generate responses while conditioning on structural tokens fixed in advance (Xiong et al., 2026). This offers a different mechanism for structural enforcement, but asking the dLLM to populate the template directly from the prompt still relies on its ability to solve the task (Figure 4). But here, an unconstrained AR draft can help because it can supply a candidate solution whose task-relevant content the dLLM can preserve and organize, rather than discover independently. The dLLM therefore need not match the drafter's task accuracy to be useful as a structurer.

Proposed approach. We introduce *Draft-Guided Constrained Diffusion* (DGCD), a two-stage procedure that realizes this division of roles. First, an AR model generates a free-form draft without schema-specific token restrictions. Second, a dLLM conditions on the input and draft to populate the content positions of a schema template fixed before generation. The key novelty is therefore to use the unconstrained draft to guide a dLLM whose structural tokens are fixed rather than generated autoregressively. In the proposed approach, structural validity follows from the construction, while task accuracy depends on draft quality and faithful transfer of its answer.

We summarize our contributions as follows.

- **Projection tax in AR vs. diffusion models.** We show that dLLMs behave qualitatively differently under structural constraints: constrained decoding costs an autoregressive model 22.8 accuracy points on MATH500, while template-based constrained diffusion costs a dLLM only 0.2 points relative to its own unconstrained generation (Sections 2 and 4).
- **Draft-guided constrained diffusion.** We propose DGCD, combining an unconstrained AR draft with template-based constrained diffusion to produce structured task outputs (Section 3).
- **Empirical gains.** We evaluate DGCD on mathematical reasoning (GSM8K, MATH500) and agentic information extraction (financial entities, PII) with six AR drafters from 1B to 14B parameters. DGCD achieves 100% structure adherence on every model and benchmark and improves average accuracy across all four benchmarks over DCCD at every scale, by 5.1 to 19.3 points. It also retains nearly its full-budget accuracy under tight token budgets, where AR constrained decoders collapse, and runs $3.6\times$ faster than DCCD (Section 4).

2 Structured Generation: Problem Setup and Challenges

We now formalize the structured-generation objective introduced above. Let x denote an input prompt, z an output token sequence, and T the maximum length of the final output. Let $\mathcal{L}(x)$ contain the outputs satisfying the required structure, and define $\mathcal{L}_T(x) = \{z \in \mathcal{L}(x) : |z| \leq T\}$. We assume the constraint is feasible, so $\mathcal{L}_T(x) \neq \emptyset$. A decoding procedure induces a distribution $q(\cdot \mid x)$, whose task accuracy is $\mathrm{Acc}(q) = \mathbb{E}_x \mathbb{E}_{z \sim q(\cdot|x)}[U(x, z)]$, where $U(x, z) \in [0, 1]$ measures

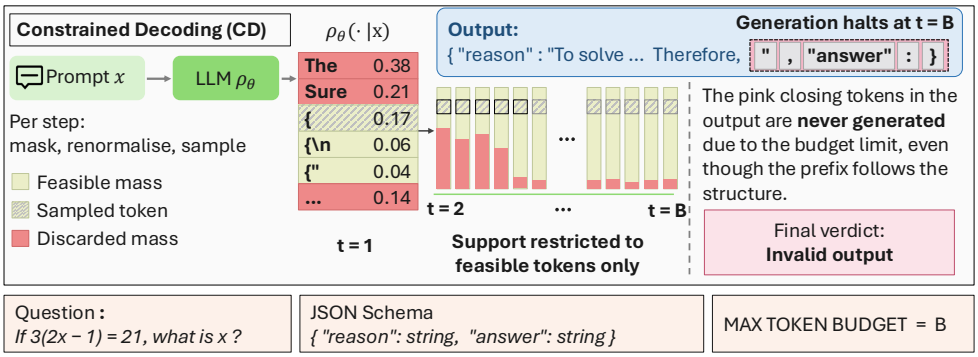


Figure 2: **Constrained decoding (CD) under a fixed token budget.** Given a prompt x (here, a math question with a two-field JSON schema), the model ρ_θ proposes a next-token distribution at every step. CD masks tokens that would violate the schema (discarded mass, red), renormalizes over the remaining feasible tokens (green), and samples from this restricted support (hatched). For example, at $t{=}1$ the model prefers `The` and `Sure`, but only `"{"` is feasible, so most of its probability mass is discarded. Each prefix remains valid, but CD does not reserve tokens for structure that must still be emitted. When the budget is reached at $t{=}B$, the closing tokens (`", "answer": }`) have not been generated, and the output is invalid even though every step respected the schema.

task-specific correctness. Structured generation can then be written as the hard-constrained problem

$$\max_{q} \ \mathrm{Acc}(q) \qquad \text{subject to} \qquad q\big(z \in \mathcal{L}_T(x) \mid x\big) = 1 \quad \text{for all } x. \tag{1}$$

We evaluate task accuracy independently of structural validity. Structure adherence (SA) is the fraction of outputs belonging to $\mathcal{L}_T(x)$. Our objective is therefore not only to improve adherence, but to guarantee valid outputs while retaining high task accuracy. We focus on template-expressible schemas; the corresponding construction is introduced in Section 3.

Autoregressive constrained decoding. Constrained decoding (CD) enforces structure during left-to-right generation by masking next tokens that would leave the current prefix with no valid completion, then renormalizing over the remaining tokens (Willard & Louf, 2023; Dong et al., 2025). Figure 2 illustrates this mechanism, which introduces two challenges.

First, enforcing structure can affect task accuracy. When the required structural tokens have low probability under the model, CD forces generation through prefixes the model would otherwise be unlikely to produce. Subsequent content is conditioned on these prefixes, so enforcing the schema can change the solution itself. We refer to the accuracy gap between unconstrained and constrained generation as the *projection tax*, $\Delta_{\mathrm{proj}} = \mathrm{Acc}(q_{\mathrm{free}}) - \mathrm{Acc}(q_{\mathrm{CD}})$, where both procedures use the same underlying model. A positive gap indicates a loss in task accuracy. Appendix H formalizes how masking and renormalization reweight the model's output distribution.

Second, prefix feasibility does not ensure completion within the output budget. Generation may exhaust its budget while populating an early field, before later fields or closing delimiters are emitted (Figure 2). The final output is then invalid even though every prefix admits a valid completion. Budget-aware restrictions can enforce completion, but further restrict the continuations available to express the content.

Template-based constrained diffusion. An alternative is to place the required structure before generating its content. Masked diffusion language models (dLLMs) iteratively fill masked positions using bidirectional context, making it possible to fix structural tokens while generating only the content positions (Xiong et al., 2026). In the template-based construction considered here (Figure 3), keys, quotation marks, and delimiters are placed in advance, and content-token choices are restricted to preserve the validity of each slot. Structural tokens are therefore supplied as context rather than sampled through a restricted next-token distribution. All required fields and closing delimiters are also allocated in advance, so content generation cannot consume the space needed to complete the structure. For a template that fits within T, resolving all content positions yields a structurally valid output under the template and slot-validity assumptions specified in Section 3. However, the dLLM must still solve the original task to populate the template. The structural guarantee does not ensure

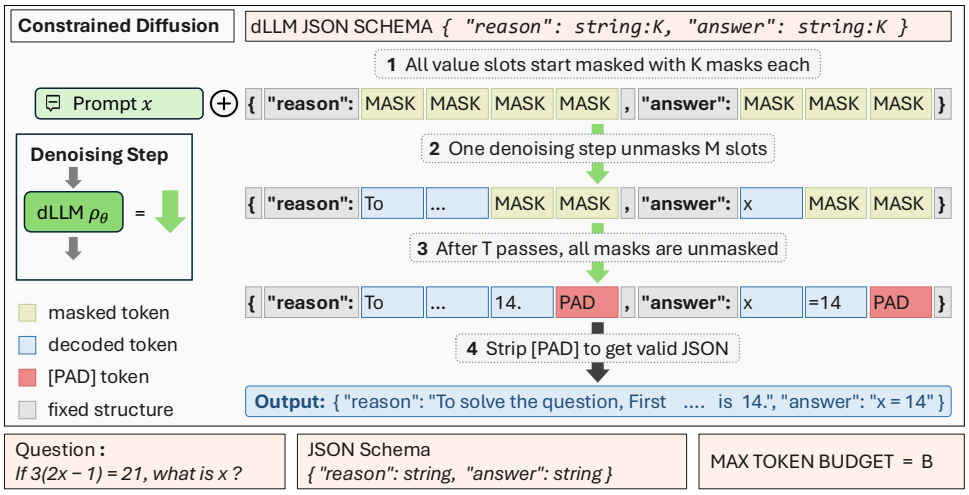


Figure 3: **Template-based constrained diffusion.** The complete schema is instantiated before content generation: structural tokens (gray) remain fixed, while masked content positions (green) are iteratively filled by the dLLM using bidirectional context. Unused positions are assigned `[PAD]` and removed after generation. With validity-preserving content restrictions and a feasible template, the resulting output is structurally complete by construction.

that the generated fields contain a correct solution: task accuracy still depends on the dLLM's ability to solve the task.

Empirical comparison and remaining challenge. Figure 4 makes these complementary limitations concrete. It compares Qwen2.5-7B-Instruct and LLaDA-8B-Instruct on MATH-500, with and without structural constraints. The autoregressive model achieves substantially higher unconstrained accuracy, but CD reduces its accuracy and does not attain full structure adherence under the evaluated token budget. Constrained diffusion instead reaches 100% adherence with almost unchanged aggregate accuracy, yet that accuracy remains far below that of the unconstrained autoregressive model. This comparison exposes the remaining gap: we would like to retain the task-relevant content produced by a strong unconstrained AR model, while using a generation mechanism that reserves and enforces the complete output structure in advance. This motivates changing the role of the diffusion model. Rather than asking it to solve the original task, we use it to structure a solution already provided by the autoregressive model. Section 3 develops this idea.

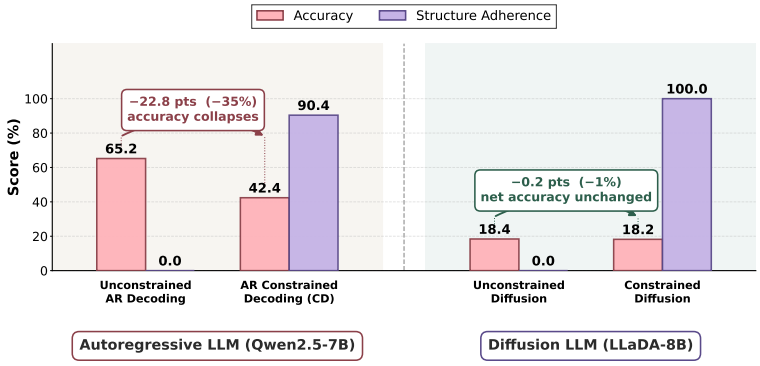


Figure 4: **Complementary limitations of CD and template-based diffusion.** Task accuracy and structure adherence on MATH-500 for Qwen2.5-7B-Instruct (left) and LLaDA-8B-Instruct (right). Under the evaluated output budget, CD raises adherence to 90.4% but lowers task accuracy. Template-based constrained diffusion reaches 100% adherence with nearly unchanged aggregate accuracy (18.4% → 18.2%), but remains substantially less accurate than unconstrained autoregressive generation. Accuracy is evaluated independently of structural validity.

3 PROPOSED APPROACH

We propose *Draft-Guided Constrained Diffusion* (DGCD), a two-stage procedure in which an autoregressive model first produces an unconstrained draft and a diffusion model then expresses that draft within a fixed schema template. The draft supplies the task-relevant content, while the template and content-token restrictions enforce structural validity.

Step 1: Generate an unconstrained draft. Given the input x, we sample a free-form draft $y \sim p_{\text{draft}}(\cdot \mid x)$ from an autoregressive model. The draft is not required to satisfy $\mathcal{L}(x)$, and no schema-specific token masking is applied during its generation. This stage allows the model to produce a candidate solution without enforcing the final output structure.

Step 2: Express the draft within a constrained template. The diffusion model π_ϕ receives the original input x, the draft y, and an instruction to express the draft's content in the required structure. We omit the instruction from the notation. Importantly, the draft remains fixed as a conditioning context; it is not itself the sequence being denoised. We instantiate the target structure as

$$z^{(N)} = \big(\sigma_0, \texttt{[MASK]}^{b_1}, \sigma_1, \ldots, \texttt{[MASK]}^{b_J}, \sigma_J\big), \tag{2}$$

where $\sigma_{0:J}$ are fixed structural token sequences and b_j is the number of content positions allocated to slot j. The allocation satisfies $\sum_{j=0}^{J} |\sigma_j| + \sum_{j=1}^{J} b_j \le T$. Slot capacities may depend on the available draft, but are fixed before denoising begins. Each content position i has a nonempty allowed-token set A_i. These sets exclude `[MASK]`; they may include `[PAD]` where padding is permitted, allowing unused positions to be removed after generation. For the validity analysis, we assume that every complete assignment of tokens from these sets yields a schema-valid output after padding removal. This condition must cover the full slot requirements, not merely prevent content from overwriting adjacent delimiters.

At denoising step n, let $\mathcal{M}_n = \{i : z_i^{(n)} = \texttt{[MASK]}\}$ denote the remaining masked positions. The model predicts their contents using the input, draft, and partially filled template. Writing $\pi_{\phi,i}$ for its prediction at position i, we restrict this distribution to A_i:

$$q_{\phi,i}\big(a \mid x, y, z^{(n)}\big) = \frac{\pi_{\phi,i}\big(a \mid x, y, z^{(n)}\big)\mathbb{I}[a \in A_i]}{\sum_{v \in A_i} \pi_{\phi,i}\big(v \mid x, y, z^{(n)}\big)}. \tag{3}$$

We assume the normalizer is positive. Positions are selected for unmasking according to their constrained confidence, $\max_{a \in A_i} q_{\phi,i}(a \mid x, y, z^{(n)})$, and their tokens are sampled from equation 3. All other positions remain unchanged. The reveal schedule resolves every mask within N steps. Structural positions are never masked or resampled. After denoising, we remove `[PAD]` tokens from the content slots to obtain the final output z. We denote the resulting two-stage output distribution by $q_{\text{DGCD}}(\cdot \mid x)$. Algorithm 1 summarizes the procedure.

Structural validity. For every input admitting the template construction above, DGCD satisfies $q_{\text{DGCD}}\big(z \in \mathcal{L}_T(x) \mid x\big) = 1$. The result follows from three properties: the structural tokens remain fixed, all content positions are resolved using the allowed sets, and the complete template fits within T. We note that removing padding cannot increase the output length and the hard constraint in equation 1 holds under the stated assumptions. This guarantee does not require the draft to be correct or schema-valid. Even an incorrect or incomplete draft cannot alter the fixed structural tokens. Unlike left-to-right generation, completing the structure does not depend on sampling its closing delimiters before exhausting the output budget because those delimiters are already present.

Accuracy transfer. Structural validity does not guarantee that the generated fields preserve the draft's task-relevant content. Draft conditioning is intended to make the diffusion model transfer a candidate solution rather than derive it independently, but this transfer can still introduce errors. We therefore evaluate task accuracy separately from structural adherence, including the accuracy change relative to the unconstrained drafter and comparison with DCCD (cf. Section 4).

Relation to draft-conditioned constrained decoding. DCCD (Reddy et al., 2026) also uses an unconstrained draft, but its formatting stage generates structural tokens autoregressively under token-level constraints. DGCD changes the structural mechanism itself. Structural tokens are fixed before generation, and diffusion is used only to populate the remaining content positions. Thus, both methods exploit draft guidance, but they differ in how the final structure is enforced. DGCD thus sepa-

Algorithm 1 Draft-Guided Constrained Diffusion

Require: Input x, draft model p_{draft}, diffusion model π_ϕ, schema template, budget T, denoising steps N

Sample unconstrained draft $y \sim p_{\text{draft}}(\cdot \mid x)$
Instantiate $z^{(N)} = (\sigma_0, \texttt{[MASK]}^{b_1}, \ldots, \texttt{[MASK]}^{b_J}, \sigma_J)$ with total length at most T
for $n = N, \ldots, 1$ **do**
  $\mathcal{M}_n \leftarrow \{i : z_i^{(n)} = \texttt{[MASK]}\}$
  Compute draft-conditioned predictions $\pi_{\phi,i}(\cdot \mid x, y, z^{(n)})$ for $i \in \mathcal{M}_n$
  Restrict each prediction to A_i and renormalize as in equation 3
  Select positions $\mathcal{R}_n \subseteq \mathcal{M}_n$ according to constrained confidence
  Reveal positions in $\mathcal{R}_n$; keep all remaining positions unchanged
end for
Remove `[PAD]` tokens from content slots
return Structured output z

rates the two requirements of structured generation: validity is enforced by the template construction, while retaining task accuracy depends on faithful transfer from the draft. Section 4 evaluates this transfer alongside structure adherence, output-budget sensitivity, and latency.

4 Experiments

We evaluate whether DGCD can combine accurate task outputs with reliable structural enforcement. We first compare accuracy and structure adherence on reasoning and extraction tasks, then examine how much accuracy each method loses during structuring, sensitivity to output budgets, and generation latency.

Table 1: Models used

Name	Model ID
14B	Qwen 2.5 14B
8B	Llama 3.1 8B
7B	Qwen 2.5 7B
3B	Qwen 2.5 3B
1.5B	Qwen 2.5 1.5B
1B	Llama 3.2 1B
dLLM	LLaDA 8B

Datasets and structural constraints. We evaluate numerical reasoning on GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021), and information extraction on the Financial Entities and PII tasks from the Cleanlab structured-output benchmark (Cleanlab, 2026). The math tasks require a JSON object containing reasoning steps and a final answer. The extraction tasks require flat schemas with list-valued or nullable fields. All outputs are checked against their target JSON schemas; complete schemas and worked examples are provided in Appendix C.

Models and baselines. We evaluate six instruction-tuned autoregressive models, ranging from 1B to 14B parameters (Table 1). Each model serves as the DGCD drafter, with LLaDA-8B-Instruct (Nie et al., 2025) performing the constrained diffusion stage. Model sizes in the results refer to the autoregressive model, not the total parameter count of the two-stage system. We compare against (i) *Constrained Decoding (CD)*, which guarantees prefix validity by masking invalid next tokens using XGrammar (Dong et al., 2025), and (ii) *Draft-Conditioned Constrained Decoding (DCCD)* (Reddy et al., 2026), the state-of-the-art method, which conditions constrained decoding on an unconstrained draft from the same model.

Evaluation metrics. Our evaluation assesses two aspects of each response: (i) *answer correctness*, by comparing the final answer with the ground truth for the math tasks and by exact match over all schema fields for the extraction tasks; and (ii) *structure adherence (SA)*, by verifying that the output parses and conforms to the target JSON schema. A response counts as a *joint success* only when both conditions hold. This joint metric captures the core challenge of structured generation: preserving reasoning quality while satisfying hard constraints.

4.1 Main Results

DGCD improves reasoning accuracy while guaranteeing structural validity. Table 2 (left) reports results on GSM8K and MATH500. DGCD achieves 100% structure adherence for every model and dataset, while CD and DCCD degrade on MATH500, dropping as low as 79.0% (CD, 8B) and 85.2% (DCCD, 8B). DGCD's accuracy gains are largest on the harder MATH500 benchmark. There

Table 2: **Main results.** Accuracy denotes answer correctness for mathematical reasoning and exact-match correctness for agentic extraction; SA (Structure Adherence) denotes the percentage of schema-valid outputs. Best results per model size are in **bold**.

		Mathematical Reasoning				Agentic Extraction			
		Accuracy		SA		Accuracy		SA	
Size	**Method**	GSM8K	MATH500	GSM8K	MATH500	Fin. Ent.	PII	Fin. Ent.	PII
1B	CD	11.3	9.0	95.4	79.4	3.9	0.0	99.9	97.0
	DCCD	40.2	20.0	99.2	90.0	5.0	0.0	99.1	**100.0**
	DGCD	**40.3**	**25.0**	**100.0**	**100.0**	**20.1**	0.0	**100.0**	**100.0**
1.5B	CD	53.6	22.0	99.8	89.0	11.5	1.0	99.9	**100.0**
	DCCD	**71.6**	38.8	99.8	95.8	10.3	2.0	99.5	**100.0**
	DGCD	**71.6**	**52.0**	**100.0**	**100.0**	**27.6**	**6.0**	**100.0**	**100.0**
3B	CD	77.5	37.4	99.9	91.0	18.6	2.0	99.8	98.0
	DCCD	82.2	47.4	99.5	96.6	10.8	0.0	98.9	**100.0**
	DGCD	**83.9**	**66.2**	**100.0**	**100.0**	**49.6**	**8.0**	**100.0**	**100.0**
7B	CD	82.5	42.4	98.3	90.4	15.8	2.0	**100.0**	98.0
	DCCD	84.0	52.8	99.9	97.8	24.2	2.0	**100.0**	**100.0**
	DGCD	**84.1**	**63.6**	**100.0**	**100.0**	**63.6**	**16.0**	**100.0**	**100.0**
8B	CD	82.8	35.6	99.4	79.0	10.3	1.0	**100.0**	**100.0**
	DCCD	**84.5**	37.2	99.6	85.2	17.8	2.0	**100.0**	**100.0**
	DGCD	83.6	**45.2**	**100.0**	**100.0**	**68.8**	**10.0**	**100.0**	**100.0**
14B	CD	90.1	55.8	99.9	99.0	32.0	2.0	**100.0**	99.0
	DCCD	**93.9**	57.0	**100.0**	99.6	32.3	5.0	**100.0**	99.0
	DGCD	**93.9**	**75.8**	**100.0**	**100.0**	**73.7**	**22.0**	**100.0**	**100.0**

it improves over DCCD at every scale, by $+5.0$ (1B) up to $+18.8$ (3B and 14B), and raises average accuracy from 33.7 (CD) and 42.2 (DCCD) to 54.6. Notably, a 3B model with DGCD (66.2) surpasses both 14B baselines on MATH500 (55.8 for CD and 57.0 for DCCD). On GSM8K, DGCD stays within 1.7 points of DCCD at every scale and yields large gains over CD for small models (e.g., $11.3 \rightarrow 40.3$ at 1B), indicating that the easier benchmark is already close to saturation once reasoning is decoupled from formatting. **Takeaway:** DGCD removes the trade-off between format compliance and reasoning quality: it guarantees schema-valid outputs while delivering the largest accuracy gains on the most challenging reasoning tasks.

DGCD substantially improves agentic structured extraction. Table 2 (right) evaluates Financial Entities and PII extraction, where outputs must populate schema fields exactly. Structure adherence is near-saturated for all methods ($\geq$97%), so the differences here come from semantic correctness rather than format validity. On Financial Entities, DGCD outperforms the strongest baseline at every scale, and the gap grows from $+15.1$ at 1B to a peak of $+51.0$ at 8B ($17.8 \rightarrow 68.8$), remaining large at 14B ($+41.4$). Averaged across models, DGCD reaches 50.6 exact accuracy compared to 15.4 for CD and 16.7 for DCCD, a $3.0\times$ improvement over the stronger baseline. A 3B model with DGCD (49.6) again exceeds both 14B baselines (32.0, 32.3). PII extraction is considerably more challenging due to its broader schema, which contains substantially more keys than the other tasks; all baselines achieve at most 5.0% exact-match accuracy. DGCD still raises performance at every scale above 1B, reaching 22.0 at 14B ($11\times$ over CD and $4.4\times$ over DCCD). In contrast to the math tasks, draft conditioning gives little and inconsistent benefit on extraction: DCCD underperforms CD on Financial Entities at 1.5B and 3B (10.3 vs. 11.5; 10.8 vs. 18.6), gains at most 8.4 points elsewhere, and improves the cross-model average by only 1.3 points, which suggests that free-form drafting alone does not transfer to field-level extraction. **Takeaway:** For agentic extraction, where schema compliance is easy but exact field values are hard, DGCD delivers large gains in correctness, with accuracy rising steadily with model capacity, whereas draft conditioning alone provides little benefit.

Qualitative examples are provided in Appendix I.

4.2 Ablations

How much accuracy does structuring cost? Final accuracy alone does not show how much of the unconstrained model's accuracy survives structuring. We therefore measure, on MATH500, how many accuracy points each method loses relative to generation without structural constraints. For CD, DCCD, and DGCD, the reference is the unconstrained output of Qwen2.5-7B-Instruct (65.2%), which is also the draft used by DCCD and DGCD; for constrained diffusion, it is the unconstrained generation of LLaDA-8B-Instruct. As shown in Figure 5, CD is largely destructive, losing 22.8 points. DCCD roughly halves this loss to 12.4 points but does not eliminate it. Constrained diffusion flips 10.0% of examples from correct to incorrect, but 9.8% flip in reverse, so its net accuracy changes by only 0.2 points. We attribute this gap to how each paradigm enforces structure. An autoregressive decoder commits to each forced token, and all later tokens condition on it. A diffusion model sees the complete structure from the first step and generates content around it, so the structure is part of the context rather than a forced detour. DGCD inherits this behavior while transcribing the draft: it loses only 1.6 points (65.2 → 63.6), about 8× less than DCCD and 14× less than CD, retaining 97.5% of the draft's accuracy. **Takeaway:** Constraints degrade autoregressive models because forced tokens cannot be revised. Diffusion models adapt the surrounding content instead, and DGCD retains nearly all of the draft's accuracy. Appendix F reports retention for each method and shows that net loss lower-bounds the correct-to-incorrect rate.

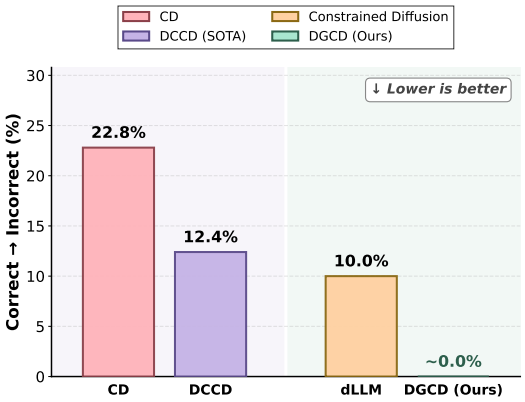


Figure 5: **Correct-to-incorrect transitions under structured generation** on MATH500, as a percentage of all 500 examples. For CD, DCCD, and DGCD, each structured output is compared with the unconstrained Qwen2.5-7B-Instruct output (65.2% accuracy), which also serves as the draft; for constrained diffusion, it is compared with LLaDA-8B-Instruct's own unconstrained generation. Lower is better.

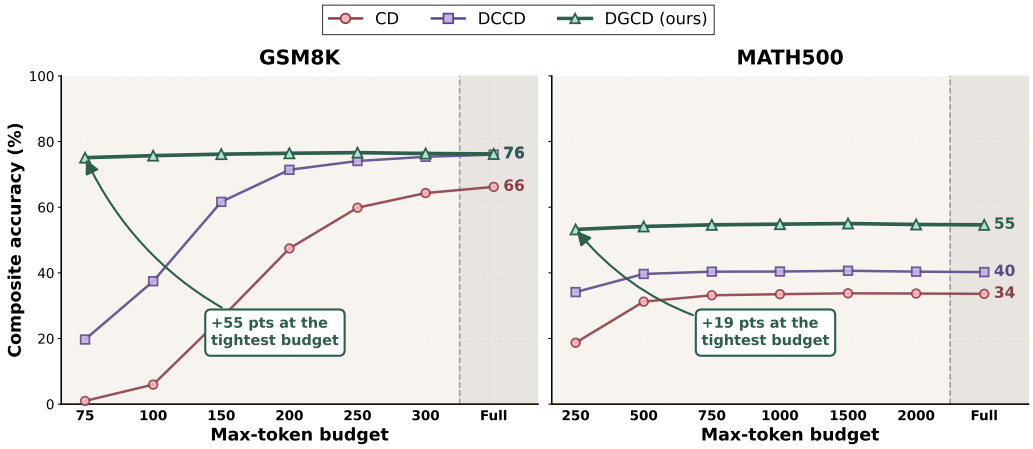


Figure 6: **Composite accuracy under varying max-token budgets** on GSM8K (left) and MATH500 (right), averaged across models. Composite accuracy requires both a correct answer and a schema-valid output. The shaded region marks the unrestricted (Full) budget. CD and DCCD degrade sharply as the budget shrinks, while DGCD remains nearly flat, outperforming DCCD by +55 points on GSM8K and +19 points on MATH500 at the tightest budget.

Sensitivity to output budgets. We next evaluate the second challenge identified in Section 2: whether the structured response can still be completed as the available output budget shrinks. Figure 6 reports composite accuracy, which counts a response as successful only when it is both correct and schema-valid. On GSM8K, CD drops from approximately 66% at the full budget to near zero

at 75 tokens, while DCCD decreases from approximately 76% to 20%. DGCD remains between 75% and 77% across the tested budgets. The same pattern appears on MATH500: DGCD remains at approximately 53–55%, whereas DCCD and CD fall to 34% and 19%, respectively, at 250 tokens. **Takeaway:** DGCD delivers near-full accuracy at a fraction of the token budget, making it well suited to token-constrained deployments where autoregressive constrained decoding collapses.

Generation latency. Reserving the output structure improves robustness to short output budgets, but DGCD adds a second model and a denoising stage. We therefore measure whether this robustness incurs a substantial latency cost. Figure 7 reports the average wall-clock time per response over Llama-3.2-1B, Qwen2.5-1.5B, Qwen2.5-3B, and Qwen2.5-7B (setup in Appendix E). Token-by-token schema enforcement is expensive: CD takes 1.163 minutes per response, 4.5× slower than unconstrained generation (0.256 minutes), and DCCD's additional drafting stage raises this to 1.470 minutes (5.7×). DGCD completes a response in 0.409 minutes, only 1.6× the unconstrained cost, making it 2.8× faster than CD and 3.6× faster than DCCD. **Takeaway:** DGCD guarantees structural validity at 1.6× the unconstrained cost, compared with 4.5–5.7× for constrained decoding.

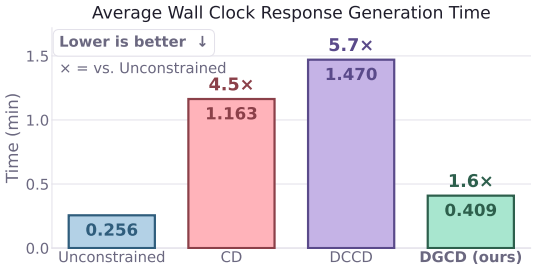


Figure 7: **DGCD is faster than constrained decoding methods.** Average wall-clock time per response across models; multipliers are relative to unconstrained generation. DGCD also achieves 100% structure adherence on every model and benchmark (Table 2).

Parameter efficiency. As DGCD employs both an AR drafter and a 1.5B diffusion structurer (Appendix G), we assess whether its improvements persist after accounting for total parameter count. As shown in Figure 8, DGCD attains comparable or higher accuracy than DCCD with substantially smaller drafters, achieving on average 4.1× higher accuracy per billion parameters across benchmarks.

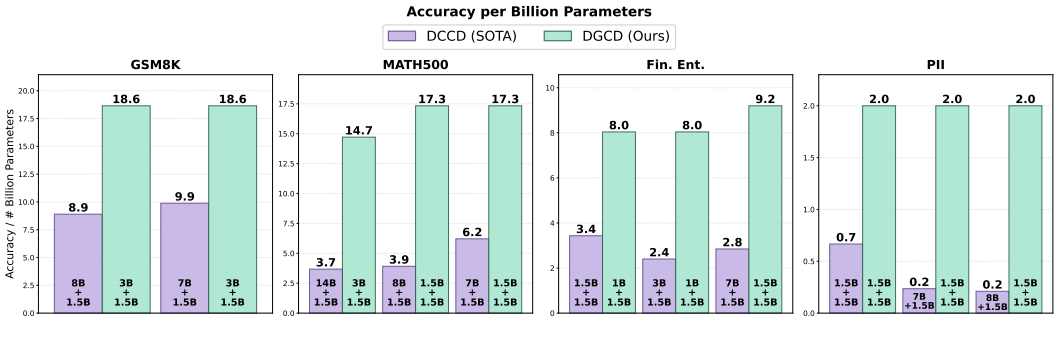


Figure 8: **Accuracy per billion parameters** for DCCD and DGCD on GSM8K, MATH500, Fin. Ent., and PII. Accuracy is divided by the combined parameter count of both models in each configuration (shown under each bar). DGCD matches or exceeds DCCD's accuracy with smaller models, yielding 1.9–4.4× higher accuracy per billion parameters, and up to 9.5× on PII.

5 Conclusion

We studied structured generation in large language models, where the goal is to satisfy the required output structure without sacrificing task accuracy. We introduced *Draft-Guided Constrained Diffusion* (DGCD), which uses an unconstrained autoregressive model to produce the task solution and a diffusion model to express that solution within a fixed schema template. This separation enables structural validity by construction while retaining the accuracy of the draft. Across reasoning and structured extraction tasks, DGCD achieves 100% observed structure adherence, improves accuracy over constrained-decoding baselines, remains robust under tight output budgets, and is substantially faster than DCCD. Overall, our results highlight the benefit of separating *task solving* from *structural enforcement* in structured generation. **Limitations and future work:** DGCD requires a separate diffusion structurer and currently focuses on template-expressible schemas with slot capacities fixed

before denoising. Extending it to richer constraints and more adaptive structuring is an important direction.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code implementation. We have not used generative AI tools for research ideation, experimental design, or the analysis and interpretation of results, and the remaining required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to polish the grammar and wording of the manuscript. We have reviewed all AI-assisted work. All AI-generated code was manually verified by the authors against the proposed algorithm to ensure that the implementation faithfully matches its specification, and all AI-assisted edits to the text were reviewed to confirm that they preserve the intended technical meaning. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work proposes Draft-Guided Constrained Diffusion (DGCD), a decoding method that improves the accuracy and structure adherence of language model outputs under formal output constraints. Our study does not involve human subjects, user studies, or the collection of new personal data, and therefore did not require IRB approval. All experiments use publicly available pretrained models, which we use in accordance with their respective licenses, together with established public benchmarks for mathematical reasoning (GSM8K, MATH500) and agentic extraction (Financial Entity extraction and PII extraction). The PII benchmark consists of synthetic customer-service messages and contains no information about real individuals.

We recognize that reliable structured extraction is dual-use. In particular, improving the accuracy of PII extraction could in principle be misused to harvest personal information from unstructured text at scale. We include this task because accurate PII detection is a core component of privacy-protecting systems such as redaction, data-loss prevention, and compliance auditing, and we evaluate it only on benchmark data. We encourage practitioners deploying DGCD for extraction to apply it within appropriate data-governance and privacy frameworks. More broadly, DGCD constrains the *format* of model outputs but does not alter the underlying models' knowledge or values; outputs may therefore inherit biases or factual errors from the base models, and schema-valid outputs should not be mistaken for correct ones. We are not aware of any conflicts of interest or sponsorship that would bias this work.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure the reproducibility of our results. Section 3 describes DGCD in full, and Algorithm 1 gives step-by-step pseudocode for the decoding procedure. Appendix G describes how we convert Qwen2.5-1.5B-Instruct into the masked diffusion structurer model. This includes the rejection-sampling procedure used to build the 1,500-sample training set, the acceptance criteria (perfect structure adherence and no projection tax), and the training configuration (Table 5). Section 4 and Appendix C specify all base models, benchmarks, output schemas and grammars, prompts, decoding hyperparameters, and evaluation metrics, including the definitions of answer accuracy, exact-match accuracy, and structure adherence used in Table 2. We will publicly release the code, the training data, and the trained diffusion structurer model upon publication.

REFERENCES

Anthropic. Introducing structured outputs on the claude developer platform, December 2025. URL https://claude.com/blog/structured-outputs-on-the-claude-developer-platform. Accessed: 2025-12-05.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. Crane: Reasoning with constrained llm generation. *arXiv preprint arXiv:2502.09061*, 2025.

Dylan Castillo. Structured outputs can hurt the performance of llms, December 2024. URL https://dylancastillo.co/posts/say-what-you-mean-sometimes.html. Accessed: 2024-12-08.

Cleanlab. Structured output benchmark. https://github.com/cleanlab/structured-output-benchmark, 2026. GitHub repository.

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

Meihua Dang and Stefano Ermon. Constrained decoding for diffusion language models via efficient inference over finite automata. *arXiv preprint arXiv:2607.07026*, 2026.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *Proceedings of Machine Learning and Systems*, 7, 2025.

Guidance AI. Guidance: A guidance language for controlling large language models. https://github.com/guidance-ai/guidance, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1535–1546, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1141. URL https://aclanthology.org/P17-1141/.

Yifan Li, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. Diffusion models for non-autoregressive text generation: A survey. *arXiv preprint arXiv:2303.06574*, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin Vechev. Type-constrained code generation with language models. *Proceedings of the ACM on Programming Languages*, 9(PLDI):601–626, 2025.

Niels Mündler, Jasper Dekoninck, and Martin Vechev. Constrained decoding of diffusion llms with context-free grammars. In *International Conference on Learning Representations*, volume 2026, pp. 99362–99408, 2026.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

OpenAI. Introducing structured outputs in the api, August 2024. URL https://openai.com/index/introducing-structured-outputs-in-the-api/. Accessed: 2024-08-24.

Kanghee Park, Timothy Zhou, and Loris D'Antoni. Flexible and efficient grammar-constrained decoding. *arXiv preprint arXiv:2502.05111*, 2025.

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.

Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1314–1324, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1119. URL https://aclanthology.org/N18-1119/.

Avinash Reddy, Thayne T Walker, James S Ide, and Amrit Singh Bedi. The hidden cost of structured generation in llms: Draft-conditioned constrained decoding. *arXiv preprint arXiv:2603.03305*, 2026.

Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184, 2024.

Maximilian Schall and Gerard De Melo. The hidden cost of structure: How constrained decoding affects language model performance. In *Proceedings of the 15th International Conference on Recent Advances in Natural Language Processing-Natural Language Processing in the Generative AI Era*, pp. 1074–1084, 2025.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, volume 37, pp. 103131–103167, 2024.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, 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 Conference on Empirical Methods in Natural Language Processing*, 2021.

Tarun Suresh, Debangshu Banerjee, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. DINGO: Constrained inference for diffusion LLMs. In *Advances in Neural Information Processing Systems*, 2025.

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. Syncode: Llm generation with grammar augmentation, 2024.

Darren Yow-Bang Wang, Zhengyuan Shen, Soumya Smruti Mishra, Zhichao Xu, Yifei Teng, and Haibo Ding. Slot: Structuring the output of large language models. *arXiv preprint arXiv:2505.04016*, 2025.

Brandon T Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

Zhen Xiong, Yujun Cai, Zhecheng Li, and Yiwei Wang. Unveiling the potential of diffusion large language model in controllable generation. In *International Conference on Learning Representations*, volume 2026, pp. 66670–66686, 2026.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b, 2025. URL https://hkunlp.github.io/blog/2025/dream.

Yitong Zhang, Yongmin Li, Yuetong Liu, Jia Li, Xiaoran Jia, Zherui Li, and Ge Li. Lookahead-then-verify: Reliable constrained decoding for diffusion LLMs under context-free grammars. *arXiv preprint arXiv:2602.00612*, 2026.

Zhanhui Zhou, Lingjie Chen, Hanghang Tong, and Dawn Song. dllm: Simple diffusion language modeling, 2026. URL https://arxiv.org/abs/2602.22661.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

CONTENTS

1 Introduction **1**

2 Structured Generation: Problem Setup and Challenges **2**

3 Proposed Approach **5**

4 Experiments **6**

4.1 Main Results 6

4.2 Ablations 8

5 Conclusion **9**

A Additional Context of Related Work **14**

B Per-Model Token Budget Analysis **15**

C Datasets, Target Schemas, and Worked Examples **15**

C.1 Overview of Task Families 17

D System Prompts **18**

D.1 Mathematical Reasoning: GSM8K and MATH500 20

D.2 Financial Entity Extraction 21

D.3 PII Extraction 22

E Experimental Details **22**

F Accuracy Retention Under Structuring **23**

G Training the Diffusion Structurer Model **23**

H Additional discussions about preliminaries **24**

H.1 Diffusion Language Models 25

I Case Study: Robustness of the Diffusion Structurer to Budget Cutoff **25**

I.1 Example 1: GSM8K with a 250-Token Budget 26

I.2 Example 2: MATH500 with a 500-Token Budget 27

I.3 Example 3: Financial Entity Extraction with an 80-Token Budget 29

A ADDITIONAL CONTEXT OF RELATED WORK

Constrained decoding guarantees that a model's output follows a required structure, such as a JSON schema or a grammar, by masking invalid tokens at each generation step. Early work forced translation models to include specific words or phrases (Hokamp & Liu, 2017; Post & Vilar, 2018). Later

methods used parsers or type systems to block invalid tokens during semantic parsing and code generation (Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Park et al., 2025; Mündler et al., 2025). Modern frameworks compile regular expressions into finite automata (Willard & Louf, 2023) or support context-free grammars (Ugare et al., 2024; Dong et al., 2025; Guidance AI, 2023), and are deployed in production LLM APIs (OpenAI, 2024; Anthropic, 2025) as the backbone of tool-calling infrastructure. However, masking and renormalizing the next-token distribution distorts the model's distribution (Park et al., 2025), and recent works (Tam et al., 2024; Castillo, 2024; Schall & De Melo, 2025) report accuracy drops of 10–30% compared to unconstrained generation. Two-stage methods reduce this distortion by separating solving from formatting: they predict an intermediate structure (Wang et al., 2025), interleave free-form reasoning with structured generation (Banerjee et al., 2025), or, as in draft-conditioned constrained decoding (DCCD) (Reddy et al., 2026), condition the constrained pass on an unconstrained draft. Yet in all of these approaches, the formatting stage is still an autoregressive constrained decoder: it checks validity only against a growing prefix, samples structural tokens under projection, and can run out of token budget before required fields or closing delimiters are emitted.

Diffusion language models (dLLMs) generate text by iteratively denoising a corrupted sequence (Li et al., 2023; Sahoo et al., 2024; Shi et al., 2024), and at scale reach instruction-following ability competitive with autoregressive models of similar size (Nie et al., 2025; Zhu et al., 2025; Ye et al., 2025). Because a dLLM can start from a sequence with some positions already fixed, S^3 (Xiong et al., 2026) fixes the schema tokens and denoises only the content positions, achieving near-perfect but not guaranteed adherence. Other methods enforce constraints during denoising, using dynamic programming over automata (Suresh et al., 2025; Dang & Ermon, 2026) or per-step rejection sampling (Mündler et al., 2026; Zhang et al., 2026); the latter offers no guarantee under a finite sampling budget. In all of these methods, the dLLM must solve the task itself, so accuracy is limited by its own reasoning ability, which generally trails that of strong autoregressive models.

Our work combines the strengths of both model families. An autoregressive model, which has strong task performance, first generates an unconstrained draft. A dLLM, which can guarantee structure, then transcribes the draft into a schema template with fixed structural tokens and restricted content tokens. This guarantees exact structural adherence and keeps accuracy close to that of the unconstrained draft.

B Per-Model Token Budget Analysis

Figure 9 breaks down the averaged results of Figure 6 for each of the six models. The trend is consistent across all model families and scales. CD and DCCD degrade sharply as the budget tightens, while DGCD stays nearly flat. On GSM8K, the collapse of the autoregressive baselines is most severe: at 75 tokens, CD falls to near-zero accuracy for every model, and DCCD falls to roughly 10–30%, whereas DGCD retains close to its full-budget accuracy. The baselines only approach DGCD at 250–300 tokens or more, and for larger models (7B–14B) they roughly match it at the full budget. On MATH500, the budget sensitivity of the baselines is milder, but DGCD outperforms both baselines at every budget for nearly all models, and the gap persists even at the full budget (e.g., Qwen2.5-3B and Qwen2.5-14B). The advantage at tight budgets is largest for small and mid-sized models, where the baselines have the least slack to complete their reasoning and close the schema within the token limit.

C Datasets, Target Schemas, and Worked Examples

This section describes the four datasets employed in our evaluation. For each family, we specify the target schema to which all structured decoding methods are constrained, present a representative input–output example, and define the corresponding evaluation protocol.

The task families are grouped into two categories. The first category, *reasoning under structure*, consists of multi-step mathematical reasoning on GSM8K and MATH500, which share a common schema. Because the schema prescribes where intermediate reasoning must be expressed, these tasks permit an isolated assessment of the effect of structural constraints on reasoning quality. The second category, *information extraction under structure*, consists of two tasks in which the reasoning demands are limited and task difficulty is determined primarily by the complexity of the schema.

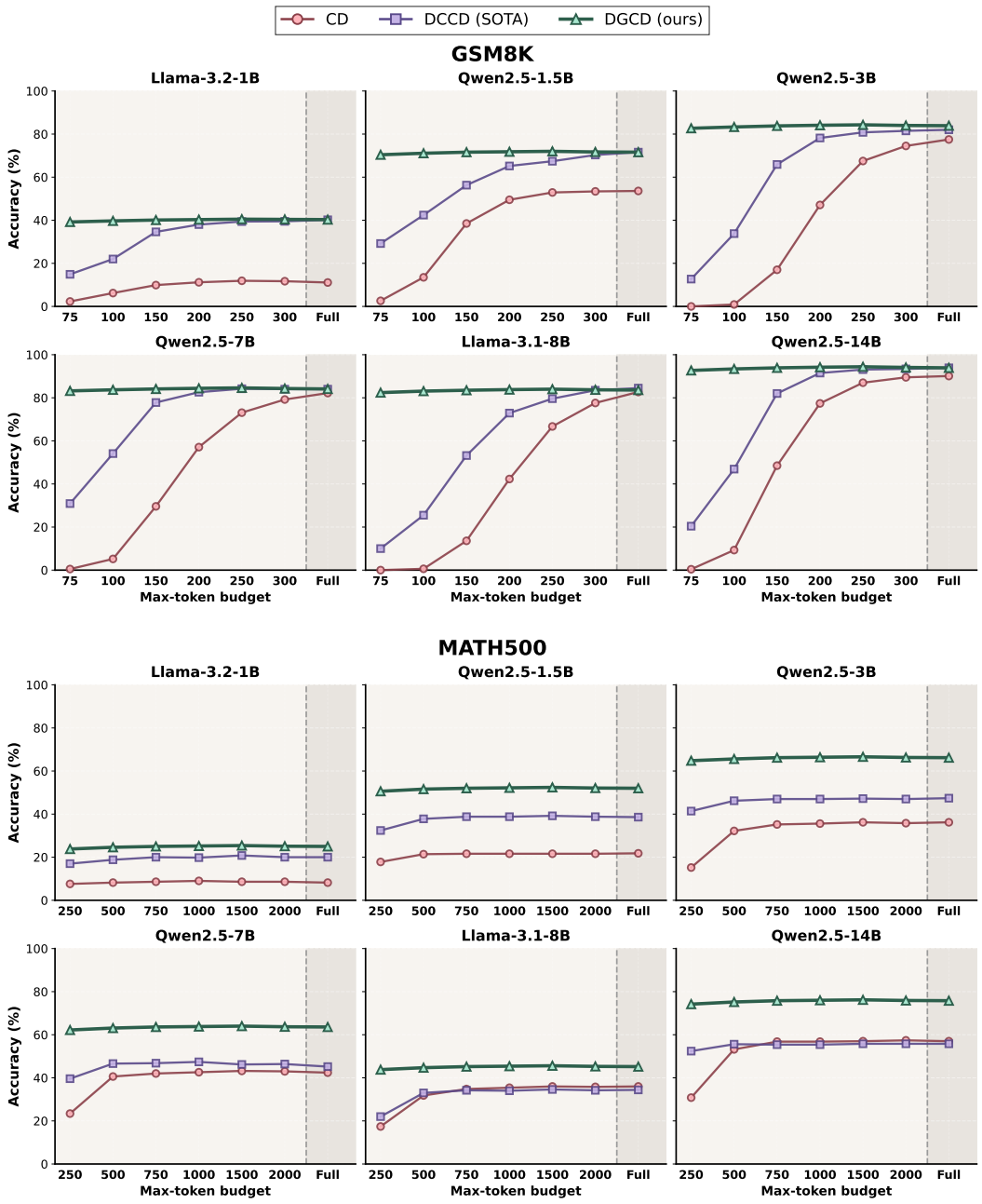


Figure 9: **Per-model composite accuracy under varying max-token budgets.** The top two rows show GSM8K and the bottom two rows show MATH500; each panel corresponds to one model, from Llama-3.2-1B to Qwen2.5-14B. Composite accuracy requires both a correct answer and a schema-valid output, and the shaded column marks the unrestricted (Full) budget. Across all models, CD and the state-of-the-art DCCD lose accuracy as the budget tightens, most sharply on GSM8K, whereas our DGCD stays nearly constant, retaining close to its full-budget accuracy even at the smallest budget. At the full budget, DGCD is on par with DCCD on GSM8K and consistently higher on MATH500.

These tasks are ordered by increasing structural complexity: a flat schema with list-valued fields and a flat schema with 56 scalar fields. All extraction datasets are obtained from the *Cleanlab structured-output-benchmark* suite.

All examples reported in this section are representative constructions. In certain cases they have been paraphrased or selected from the underlying data distributions for clarity and brevity; they do not constitute verbatim transcripts of logged model outputs.

C.1 Overview of Task Families

This subsection specifies the complete target structure of each task family in order of increasing schema complexity. All structured decoding methods are constrained to precisely these structures. In each listing, the value associated with a field denotes its type, and comments (`//`) describe its intended content. Worked examples and evaluation protocols are provided in the subsequent subsections.

Mathematical Reasoning (GSM8K & MATH500)

Source: GSM8K and MATH500, which consist of natural-language mathematical word problems, each admitting a unique final answer.
Structural constraint: The output must be a JSON object comprising exactly two required fields. The field `"steps"` is an ordered, non-empty list of strings, each of which records a single step of the intermediate derivation. The field `"answer"` is a non-empty string containing the final answer, which is numeric for GSM8K and either numeric or a closed-form symbolic expression for MATH500.
Capability assessed: Multi-step reasoning under the requirement that intermediate computations be expressed within fixed JSON fields.

Complete target structure (`GSM8KSchema`, shared by both datasets).

```
{
  "steps": [              // required; list[string], at least 1 item
    "string",             // step 1 of the reasoning
    "string",             // step 2 of the reasoning
    ...
    "string"              // step N of the reasoning
  ],
  "answer": "string"      // required; non-empty; final answer only
}
```

Financial Entity Extraction

Source: `Cleanlab/fire-financial-ner-extraction`, a corpus of financial news headlines and short news excerpts.
Structural constraint: The output must be a flat JSON object comprising seven entity fields. Each field contains a list of surface forms reproduced verbatim from the input, or `null` if no entity of the corresponding category is present. The ordering of elements within each list is not significant.
Capability assessed: Exhaustive recall of all entity mentions within each category.

Complete target structure (`ExtractedEntities`).

```
{
  "Company":  "list[string] | null",   // company and organization
    names
  "Date":     "list[string] | null",   // dates and date expressions
  "Location": "list[string] | null",   // geographic locations
  "Money":    "list[string] | null",   // monetary amounts
  "Person":   "list[string] | null",   // names of people
  "Product":  "list[string] | null",   // product names
  "Quantity": "list[string] | null"    // non-monetary quantities
}
```

PII Extraction

Source: `Cleanlab/pii-extraction`, a corpus of short synthetic customer-service messages and items of correspondence.
Structural constraint: The output must be a flat JSON object comprising 56 optional fields, one for each category of personally identifiable information (PII). Each field contains the corresponding span reproduced from the input, or `null` if that category is absent. The grouping of fields shown below is provided solely for readability; the schema itself is flat and imposes no such grouping.
Capability assessed: Correct abstention, that is, the prediction of `null` for the large majority of fields that are absent from any given input.

Complete target structure (56 fields, each of type `string | null`).

```
{
  // Identity (10)
  "PREFIX":            "string | null",
  "FIRSTNAME":         "string | null",
  "MIDDLENAME":        "string | null",
  "LASTNAME":          "string | null",
  "AGE":               "string | null",
  "SEX":               "string | null",
  "GENDER":            "string | null",
  "EYECOLOR":          "string | null",
  "HEIGHT":            "string | null",
  "DOB":               "string | null",

  // Contact / account (11)
  "EMAIL":             "string | null",
  "PHONENUMBER":       "string | null",
  "USERNAME":          "string | null",
  "PASSWORD":          "string | null",
  "PIN":               "string | null",
  "ACCOUNTNAME":       "string | null",
  "ACCOUNTNUMBER":     "string | null",
  "COMPANYNAME":       "string | null",
  "JOBTITLE":          "string | null",
  "JOBAREA":           "string | null",
  "JOBTYPE":           "string | null",

  // Financial (14)
  "CREDITCARDNUMBER":  "string | null",
  "CREDITCARDCVV":     "string | null",
  "CREDITCARDISSUER":  "string | null",
  "CURRENCY":          "string | null",
  "CURRENCYCODE":      "string | null",
  "CURRENCYNAME":      "string | null",
  "CURRENCYSYMBOL":    "string | null",
  "AMOUNT":            "string | null",
  "IBAN":              "string | null",
  "BIC":               "string | null",
  "MASKEDNUMBER":      "string | null",
  "BITCOINADDRESS":    "string | null",
  "ETHEREUMADDRESS":   "string | null",
  "LITECOINADDRESS":   "string | null",

  // Location / address (9)
  "STREET":            "string | null",
  "SECONDARYADDRESS":  "string | null",
  "BUILDINGNUMBER":    "string | null",
  "CITY":              "string | null",
  "COUNTY":            "string | null",
  "STATE":             "string | null",
  "ZIPCODE":           "string | null",
  "NEARBYGPSCOORDINATE": "string | null",
  "ORDINALDIRECTION":  "string | null",

  // Device / network (7)
  "IP":                "string | null",
  "IPV4":              "string | null",
  "IPV6":              "string | null",
  "MAC":               "string | null",
  "USERAGENT":         "string | null",
  "URL":               "string | null",
  "PHONEIMEI":         "string | null",

  // Vehicle (2)
  "VEHICLEVIN":        "string | null",
  "VEHICLEVRM":        "string | null",

  // Other identifiers (3)
  "SSN":               "string | null",
  "DATE":              "string | null",
  "TIME":              "string | null"
}
```

D SYSTEM PROMPTS

This section lists the system prompts used by each method. Prompts are selected per task from three sets: a *constrained* prompt for single-pass structured generation, a *Stage 1* prompt for free-form drafting, and a *Stage 2* prompt for converting a draft into the target structure. Table 3 summarizes which prompt each method uses. DGCD uses the same Stage 2 prompt as DCCD, and the two methods differ only in decoding: DCCD restructures the draft autoregressively under a grammar constraint, whereas DGCD fixes the JSON skeleton as literal tokens and denoises only the masked spans corresponding to field values.

User messages. For single-pass methods and for Stage 1, the user turn contains the raw task input (the problem statement or source text). For Stage 2 of DCCD and DGCD, the user turn wraps the Stage 1 draft as follows:

```
Prior solution (source text; do not add new facts):
{draft}
```

Method	Stage	System prompt
Unconstrained	Single pass, free-form generation	Stage 1
CD	Single pass, schema-constrained decoding	Constrained
DCCD	Stage 1: free-form draft (AR)	Stage 1
	Stage 2: constrained restructuring (AR)	Stage 2
Constrained diffusion	Single pass, template filling by masked diffusion	Constrained
DGCD (ours)	Stage 1: free-form draft (AR)	Stage 1
	Stage 2: template filling by masked diffusion	Stage 2

Table 3: System prompt used by each method.

Mathematical Reasoning (GSM8K & MATH500)

MATH500 uses the same prompts as GSM8K.

Constrained prompt (CD and constrained diffusion).

```
Solve the problem step by step. Format the final response as a simple
    JSON object with two fields: 'steps', a list of strings for the
    reasoning steps, and 'answer', a string for the final answer. put
    the final answer in \boxed{}.
```

Stage 1 prompt (unconstrained, DCCD, and DGCD).

```
You are a meticulous math tutor. Provide a complete, detailed
    solution in plain text. Complete the full solution in very few
    tokens. Do not use JSON. Get to the final solution quickly. Put
    the final answer in \boxed{}.
```

Stage 2 prompt (DCCD and DGCD).

```
You are a precise extractor. Convert the prior free-form solution
    into a JSON object with the schema: {steps: string[], answer:
    string}. Use only information present in the prior solution. Put
    the final answer in \boxed{}. No extra text, no code
    fences---emit ONLY the JSON.
```

Financial Entity Extraction

Shared instruction prefix (constrained and Stage 1 prompts).

```
Identify and extract entities from the financial news text into these
    categories:
- Company: official or unofficial name of a registered company or
    brand (e.g. Apple Inc.; Uber; Bank of America).
- Date: any explicit or implicit time period (e.g. June 2nd, 2010;
    last week; prior year; Wednesday).
- Location: geographical locations --- regions, countries, states,
    cities, roads (e.g. California; Paris; Europe).
- Money: a monetary value in any currency, including digital ones
    (e.g. $76.3 million; Rs 33.80 crore; 1.2 BTC).
- Person: the name of an individual (e.g. Meg Whitman; Warren Buffet).
- Product: a physical object or service made or provided by a
    company, excluding businesses/sectors (e.g. iPhone; cloud
    services; laptops).
- Quantity: a numeric value that is not Money --- percentages,
    counts, measurements, including units (e.g. 15%; 25,000 units;
    100 tons).
Extract all relevant entities per category as a list of strings,
    preserving the wording from the text. Use None for a category
    with no entities. Only extract entities explicitly mentioned in
    the text; do not infer. Extract individual items rather than
    compound or ranged entities.
```

Constrained prompt (CD and constrained diffusion): shared prefix, followed by

```
Return ONLY a JSON object with the fields Company, Date, Location,
    Money, Person, Product, Quantity, each a list of strings or null.
```

Stage 1 prompt (unconstrained, DCCD, and DGCD): shared prefix, followed by

```
List the entities you find for each category in plain text. Do not
    use JSON. Be concise.
```

Stage 2 prompt (DCCD and DGCD).

```
You are a precise extractor. Convert the prior entity list into a
    JSON object with the fields: Company, Date, Location, Money,
    Person, Product, Quantity --- each a list of strings or null. Use
    only information present in the prior text. No extra text, no
    code fences---emit ONLY the JSON.
```

PII Extraction

Constrained prompt (CD and constrained diffusion).

```
Your task is to extract structured information about PII entities
    from the text provided by the user. For each field in the
    response format, extract the corresponding PII entity if it
    exists in the text. If a particular PII entity is not present in
    the text, set that field to null. Return the complete structured
    JSON response with all fields, setting missing entities to null.
```

Stage 1 prompt (unconstrained, DCCD, and DGCD).

```
Read the text and list every piece of personally identifiable
    information (PII) you find, naming each value's type (e.g.
    FIRSTNAME, EMAIL, SSN, PHONENUMBER, CITY). Write it in plain
    text. Do not use JSON.
```

Stage 2 prompt (DCCD and DGCD).

```
You are a precise extractor. Convert the prior list of PII into a
    flat JSON object where each key is a PII type and each value is
    the extracted string, with null for any type not present. Use
    only information present in the prior text. No extra text, no
    code fences---emit ONLY the JSON.
```

D.1 Mathematical Reasoning: GSM8K and MATH500

Both datasets consist of natural-language word problems, each with a unique final answer that is numeric for GSM8K and either numeric or a closed-form symbolic expression for MATH500. Both are evaluated against the `GSM8KSchema` structure specified in Appendix C.1. The schema is deliberately minimal. Since it introduces negligible difficulty of its own, any difference in accuracy between free-form and structured decoding may be attributed to the projection of the reasoning process onto JSON fields rather than to the complexity of the output format.

GSM8K-Style Example

Question.

A bakery uses 3 cups of flour for every batch of muffins. The baker has 42 cups of flour on hand but needs to set aside 6 cups for a wedding cake ordered separately. How many full batches of muffins can she bake with the flour that is left?

Free-form (unconstrained) output. The model generates its reasoning as unstructured text and designates the final answer; no JSON structure is imposed.

```
First, subtract the flour reserved for the cake: 42 - 6 = 36 cups
remain for muffins. Each batch needs 3 cups, so the number of
batches is 36 / 3 = 12. The answer is \boxed{12}.
```

Structured output. The same reasoning, projected onto the target schema.

```
{
  "steps": [
    "Flour available for muffins: 42 - 6 = 36 cups.",
    "Batches possible: 36 / 3 = 12 batches."
  ],
  "answer": "12"
}
```

D.2 Financial Entity Extraction

The first extraction task employs the structurally simplest schema in the suite. The data are obtained from `Cleanlab/fire-financial-ner-extraction`, which comprises financial news headlines and short news excerpts. The model is required to extract entities belonging to seven categories; each field contains a list of surface forms reproduced from the text, or `null` if the category is absent. The complete structure is specified in Appendix C.1.

Financial NER Example

Text.

Skyline Foods completed its $2.3 billion acquisition of Harvest Grocers on March 14, 2024, according to a statement from CEO Laura Chen in Chicago.

Gold annotation.

```
{
  "Company":  ["Skyline Foods", "Harvest Grocers"],
  "Date":     ["March 14, 2024"],
  "Location": ["Chicago"],
  "Money":    ["$2.3 billion"],
  "Person":   ["Laura Chen"],
  "Product":  null,
  "Quantity": null
}
```

Scoring. Each field is evaluated by order-independent set matching. For instance, the lists `["Skyline Foods", "Harvest Grocers"]` and `["Harvest Grocers", "Skyline Foods"]` are regarded as equivalent. An example is considered fully correct only if all seven fields match. The following prediction illustrates the difference in stringency between this exact-match criterion and a per-field metric.

Partially Correct Prediction

```
{
  "Company":  ["Skyline Foods"],
  "Date":     ["March 14, 2024"],
  "Location": ["Chicago"],
  "Money":    ["$2.3 billion"],
  "Person":   ["Laura Chen"],
  "Product":  null,
  "Quantity": null
}
```

Analysis: The prediction omits `"Harvest Grocers"` from the `Company` field but is otherwise correct. It accordingly attains a score of $6/7$ under `per_field_accuracy` and a score of 0 under exact-match `accuracy`. A single omitted mention therefore suffices to render the entire example incorrect under the exact-match criterion, and the discrepancy between the two metrics grows with the number of fields in the schema.

D.3 PII Extraction

The second extraction task retains a flat structure but substantially increases the number of fields. The data are obtained from `Cleanlab/pii-extraction`, which comprises short synthetic customer-service messages and items of correspondence. The schema contains 56 optional string fields, one for each PII category, and fields corresponding to absent categories are assigned `null`. The complete list of fields is specified in Appendix C.1.

PII Extraction Example

Text.

Dear Customer Support, I am writing regarding account 4471-2290, which was charged $58.40 on 04/02/2024. Please send confirmation to marcus.webb82@mailbox.example or contact me at (312) 555-0199. Sincerely, Marcus Webb.

Gold annotation (non-null fields only; the remaining 49 fields are `null`).

```
{
  "FIRSTNAME":       "Marcus",
  "LASTNAME":        "Webb",
  "ACCOUNTNUMBER":   "4471-2290",
  "AMOUNT":          "58.40",
  "DATE":            "04/02/2024",
  "EMAIL":           "marcus.webb82@mailbox.example",
  "PHONENUMBER":     "(312) 555-0199"
}
```

Scoring. In this example, only 7 of the 56 fields are populated; a substantial proportion of the task therefore consists of correct abstention on absent fields. The whole-example metric requires all 56 fields to match, including the 49 fields that must be predicted as `null`, and is consequently highly conservative. For consistency with Financial Entities, we nevertheless report whole-example exact match in Table 2; the low absolute values on PII largely reflect this stringency.

E Experimental Details

Hardware. All experiments were run on NVIDIA H100 GPUs with Intel Xeon CPUs.

Main results. Our main results use LLaDA-8B-Instruct as the base model, with 32 denoising steps. To keep the comparison fair, the generation length for each dataset is set to the average number of tokens produced by the DCCD and CD baselines on that dataset, so all methods operate under a comparable output budget. Unless stated otherwise, the structurer is LLaDA-8B-Instruct. The latency (Figure 7) and parameter-efficiency (Figure 8) analyses instead use a 1.5B structurer obtained by converting Qwen2.5-1.5B-Instruct into a masked diffusion model and fine-tuning it (Appendix G).

Token-budget experiments. For DGCD, slot capacities are scaled proportionally so that the full template, including structural tokens, fits within each budget.

Accuracy per billion parameters. In this comparison both methods use a 1.5B second-stage model: DGCD uses the diffusion structurer described in Appendix G, converted from Qwen2.5-

1.5B-Instruct, and DCCD uses Qwen2.5-1.5B-Instruct as its autoregressive constrained restructurer. DGCD achieves higher accuracy per billion parameters in all 11 comparisons: 1.9–4.4× on GSM8K, MATH500, and Fin. Ent., and 3.0–9.5× on PII, for an average of 4.1×.

Wall-clock time. For the wall-clock measurements, we fix the generation length at 2,000 tokens and use 32 denoising steps for DGCD. These experiments use a diffusion language model obtained by converting Qwen2.5-1.5B-Instruct with the dLLM library.

F Accuracy Retention Under Structuring

Net loss as a bound on correct-to-incorrect transitions. Let a_{free} and a_{struct} denote accuracy without and with structural constraints on the same examples, and let $p_{\text{c}\to\text{i}}$ and $p_{\text{i}\to\text{c}}$ denote the fractions of examples that flip from correct to incorrect and from incorrect to correct, respectively. Then

$$a_{\text{struct}} = a_{\text{free}} - p_{\text{c}\to\text{i}} + p_{\text{i}\to\text{c}}, \qquad \text{so} \qquad p_{\text{c}\to\text{i}} = (a_{\text{free}} - a_{\text{struct}}) + p_{\text{i}\to\text{c}} \;\geq\; a_{\text{free}} - a_{\text{struct}}. \tag{4}$$

The net accuracy loss reported in Figure 5 is therefore a lower bound on the correct-to-incorrect rate, with equality when no example flips from incorrect to correct.

Retention on MATH500. Table 4 reports net loss and the fraction of unconstrained accuracy retained after structuring. For CD, DCCD, and DGCD, the reference is the unconstrained output of Qwen2.5-7B-Instruct, which is also the Stage 1 draft used by DCCD and DGCD. For constrained diffusion, the reference is LLaDA-8B-Instruct's own unconstrained generation.

Method	Model	a_{free}	a_{struct}	Net loss (pts)	Retained (%)
CD	Qwen2.5-7B	65.2	42.4	22.8	65.0
DCCD	Qwen2.5-7B	65.2	52.8	12.4	81.0
Constrained diffusion	LLaDA-8B	18.4	18.2	0.2	98.9
DGCD	Qwen2.5-7B + LLaDA-8B	65.2	63.6	1.6	97.5

Table 4: Accuracy retention under structuring on MATH500. Retained $= a_{\text{struct}}/a_{\text{free}}$.

Net loss can hide offsetting transitions. A small net loss does not imply that individual answers are preserved. For constrained diffusion, a per-example comparison shows that 10.0% of examples flip from correct to incorrect and 9.8% from incorrect to correct, so the near-zero net change ($18.4 - 10.0 + 9.8 = 18.2$) reflects roughly symmetric perturbation rather than faithful preservation. This is why DGCD conditions the diffusion model on a draft: the draft anchors the answer, so the structurer's role is to transcribe rather than re-solve.

G Training the Diffusion Structurer Model

Model conversion. We build the diffusion structurer on top of Qwen2.5-1.5B-Instruct. Using the dLLM library (Zhou et al., 2026), we convert this autoregressive (AR) model into a masked diffusion language model (MDLM). The conversion keeps the pretrained transformer weights and changes two things: the causal attention mask is replaced with bidirectional attention, and the next-token prediction objective is replaced with the masked diffusion objective. Under this objective, the model learns to reconstruct tokens that have been randomly masked at a sampled noise level.

Training data. We construct the training data by rejection sampling from the DCCD pipeline, using Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct as the generating models. For each training prompt, the AR model first produces an unconstrained draft, which DCCD then converts into the target structured output. A sample is accepted only if it meets two criteria. First, the structured output must be fully schema-valid, i.e., the draft is converted into a perfect structure. Second, the conversion must incur no *projection tax*: the structured output must preserve the correctness of the draft, so that a correct draft maps to a correct structured answer and an incorrect draft maps to an incorrect one. The second criterion ensures that the structurer learns a faithful transformation from free-form text to structure, rather than learning to repair or corrupt the underlying answer. The

final dataset contains 1,500 samples, with 500 from each of three domains: structured mathematical reasoning, financial entity extraction, and PII extraction.

Training details. Table 5 lists the training configuration used.

Parameter	Value
Base model	Qwen2.5-1.5B-Instruct
Diffusion formulation	MDLM (masked diffusion)
Data generation models	Qwen2.5-7B/14B-Instruct (via DCCD)
Training samples	1,500 (500 per domain)

Table 5: Training Configuration for training the diffusion structurer model.

H Additional discussions about preliminaries

Autoregressive constrained decoding. An autoregressive language model π_θ induces the sequence distribution

$$\rho_\theta(z_{1:T} \mid x) \;=\; \prod_{t=1}^{T} \pi_\theta(z_t \mid h_t), \qquad h_t \triangleq (x, z_{<t}), \tag{5}$$

and commits to each token z_t before generating z_{t+1}.

Constrained decoding. Constrained decoding (CD) enforces $\mathcal{L}(x)$ through the valid-next-token set

$$A(h_t) \;\triangleq\; \big\{a \in \mathcal{V} :\; \exists\, w \in \mathcal{V}^\star \text{ s.t. } (z_{<t}, a, w) \in \mathcal{L}(x)\big\}. \tag{6}$$

Termination is treated as a token that belongs to $A(h_t)$ if and only if $z_{<t} \in \mathcal{L}(x)$. At each step, CD masks invalid tokens and renormalizes:

$$q(z_t \mid h_t) \;=\; \frac{\pi_\theta(z_t \mid h_t)\, \mathbb{I}[z_t \in A(h_t)]}{\alpha(h_t)}, \qquad \alpha(h_t) \;\triangleq\; \sum_{a \in A(h_t)} \pi_\theta(a \mid h_t), \tag{7}$$

where $\alpha(h_t)$ is the feasible mass. This procedure has two limitations.

Distortion from low feasible mass. Each projection step incurs $\mathrm{KL}\big(q(\cdot \mid h_t) \,\|\, \pi_\theta(\cdot \mid h_t)\big) = \log \frac{1}{\alpha(h_t)}$. For any valid z, the induced sequence distribution $\rho_q(z \mid x) \triangleq \prod_{t=1}^{T} q(z_t \mid h_t)$ satisfies

$$\rho_q(z \mid x) \;=\; \frac{\rho_\theta(z \mid x)}{\prod_{t=1}^{T} \alpha(h_t)}. \tag{8}$$

Compare this with the base model conditioned on validity, $\rho^\star(z \mid x) \triangleq \rho_\theta(z \mid x)\, \mathbb{I}[z \in \mathcal{L}(x)] / \rho_\theta(\mathcal{L}(x) \mid x)$, which rescales every valid sequence by the same constant. CD instead rescales each valid sequence by a trajectory-dependent factor $\big(\prod_t \alpha(h_t)\big)^{-1}$. Hence $\rho_q \neq \rho^\star$ whenever feasible mass varies across prefixes. Structural tokens (braces, quotes, field names) are a main source of low feasible mass. When $A(h_t) = \{s_t\}$ is a singleton, $\alpha(h_t) = \pi_\theta(s_t \mid h_t)$, which is small if the model would otherwise respond in free-form text. Forcing such tokens drives generation through unlikely prefixes and shifts the distribution at the content positions that follow. For Qwen2.5-7B-Instruct on MATH-500, CD lowers accuracy from 65.2% to 42.4%.

Validity is guaranteed only for prefixes. By equation 6, CD guarantees that $z_{\le t}$ can be completed to a valid sequence, but not that the completion fits within the budget. If generation reaches $t = T$ before termination, z is a valid prefix but $z \notin \mathcal{L}_T(x)$. With an exact constraint oracle $A(\cdot)$,

$$q\big(z \in \mathcal{L}_T(x) \mid x\big) \;=\; 1 - q\big(\text{no termination within } T \text{ tokens} \mid x\big). \tag{9}$$

Guaranteeing completion within the budget requires the stricter set

$$A_T(h_t) \;\triangleq\; \big\{a \in \mathcal{V} :\; \exists\, w \in \mathcal{V}^\star,\; |w| \le T - t,\; \text{s.t. } (z_{<t}, a, w) \in \mathcal{L}(x)\big\}. \tag{10}$$

Since $A_T(h_t) \subseteq A(h_t)$, this can only lower the feasible mass. It also forces the structure to close early and truncates content, which adds to the distortion above.

Draft-conditioned constrained decoding. DCCD (Reddy et al., 2026) increases feasible mass by conditioning the constrained pass on an unconstrained draft $y \sim p_{\text{draft}}(\cdot \mid x)$. It decodes from $\tilde{q}(z_t \mid \tilde{h}_t) \propto p_{\text{proj}}(z_t \mid \tilde{h}_t)\,\mathbb{I}[z_t \in A(h_t)]$ with $\tilde{h}_t \triangleq (x, y, z_{<t})$, so the draft-conditioned feasible mass $\tilde{\alpha}(\tilde{h}_t)$ replaces $\alpha(h_t)$. The valid set is still $A(h_t)$, so DCCD inherits the prefix-level guarantee of CD. It reaches 97.8% rather than 100% adherence, with 52.8% accuracy.

H.1 Diffusion Language Models

A masked diffusion language model π_ϕ generates all T positions of a canvas jointly by iterative unmasking. Let $z^{(n)} = (z_1^{(n)}, \ldots, z_T^{(n)})$ denote the canvas at reverse step $n \in \{N, \ldots, 0\}$, and let $\mathcal{M}_n \triangleq \{i : z_i^{(n)} = \texttt{[MASK]}\}$ be its masked positions. At each step $n \to n-1$, the model computes $\pi_\phi(z_i \mid z^{(n)}, x)$ for every $i \in \mathcal{M}_n$, attending bidirectionally to the whole canvas. It then reveals a subset $\mathcal{R}_n \subseteq \mathcal{M}_n$, for example the most confident positions:

$$z_i^{(n-1)} \sim \pi_\phi(\cdot \mid z^{(n)}, x) \;\text{ if } i \in \mathcal{R}_n, \qquad z_i^{(n-1)} = z_i^{(n)} \;\text{ otherwise,} \tag{11}$$

with $\mathcal{M}_0 = \emptyset$. Positions outside $\mathcal{M}_N$ are therefore never modified.

Template instantiation. Suppose valid outputs decompose into J content slots separated by fixed structural strings:

$$z = (\sigma_0, c_1, \sigma_1, \ldots, c_J, \sigma_J), \qquad c_j \in \mathcal{L}_j. \tag{12}$$

Here each σ_j is a structural string (keys, quotes, braces), and c_j is the content of slot j, drawn from a slot language $\mathcal{L}_j$ (e.g., the contents of a JSON string). Given slot sizes $b_1, \ldots, b_J$ with $\sum_{j=0}^{J} |\sigma_j| + \sum_{j=1}^{J} b_j \le T$, we initialize

$$z^{(N)} = (\sigma_0, \texttt{[MASK]}^{b_1}, \sigma_1, \ldots, \texttt{[MASK]}^{b_J}, \sigma_J). \tag{13}$$

Let $\mathcal{P}_\sigma$ denote the structural positions and $\mathcal{P}_j$ the positions of slot j, so that $\mathcal{M}_N = \bigcup_j \mathcal{P}_j$. Positions a slot does not need are filled with `[PAD]`, and each slot's content is recovered as $c_j = \text{strip}_{\texttt{[PAD]}}(z_{\mathcal{P}_j}^{(0)})$. By analogy with $A(h_t)$, each content position $i \in \mathcal{P}_j$ has a valid set $A_i \triangleq \mathcal{V}_j \cup \{\texttt{[PAD]}\}$ with $\mathcal{V}_j \subseteq \mathcal{V}$. Its predictions are masked and renormalized:

$$q_\phi(z_i \mid z^{(n)}, x) \;=\; \frac{\pi_\phi(z_i \mid z^{(n)}, x)\,\mathbb{I}[z_i \in A_i]}{\alpha_i(z^{(n)})}, \qquad \alpha_i(z^{(n)}) \;\triangleq\; \sum_{a \in A_i} \pi_\phi(a \mid z^{(n)}, x). \tag{14}$$

Unlike $A(h_t)$, the set A_i depends on the position but not on the tokens generated so far.

I Case Study: Robustness of the Diffusion Structurer to Budget Cutoff

Deployed structured-output systems typically cap the number of generated tokens. An autoregressive (AR) decoder writes a JSON object from left to right, so the answer field, which comes last, receives only the budget that remains after the reasoning steps. When the budget is exhausted, generation stops mid-string: the object is never closed, and the output is both invalid JSON and missing its answer, even when the underlying reasoning is correct. A standard grammar constraint restricts *which* tokens may be emitted, but it cannot ensure that the object closes before the budget ends.

DGCD avoids this failure mode by construction. Its diffusion structurer does not generate the JSON skeleton. Instead, every structural token (braces, keys, quotes, and commas) is fixed as a literal, and only a predefined set of masked field spans is denoised. For the mathematical reasoning tasks, the template contains three 64-token step slots and one 32-token answer slot, for 224 content tokens in total. Because the answer slot is reserved before decoding begins, it cannot be crowded out by verbose steps, and every output is closed.

We present one example from each of three datasets, all using Qwen2.5-7B-Instruct as the AR model. Examples 1 and 2 (mathematical reasoning) report measured outputs with LLaDA-8B-Instruct as the structurer. Example 3 (financial entity extraction) uses measured inputs, drafts, and AR outputs, but its DGCD output is *illustrative*: it shows how the per-field template would process the same draft.

I.1 Example 1: GSM8K with a 250-Token Budget

GSM8K Problem

A tank has a capacity of 18000 gallons. Wanda and Ms. B decided to pump water from a pond to fill the tank in two days. On the first day, working in shifts, Wanda filled $1/4$ of the tank's capacity with water, and Ms. B pumped $3/4$ as much water as Wanda pumped into the tank that day. On the second day, Wanda pumped $2/3$ of the amount of water she pumped on the previous day, while Ms. B only pumped $1/3$ of the number of gallons she pumped on the first day. How many gallons of water are remaining for the tank to be full?
Ground truth answer: $\boxed{6000}$

CD and DCCD are given a 250-token budget, slightly more than the 224 content tokens available to the diffusion structurer. With an unlimited budget, both CD and DCCD solve this problem correctly.

CD Output (truncated, invalid JSON)

```
{
"steps": [
"Wanda filled 1/4 of the tank's capacity on the first day, which is
    1/4 * 18000 = 4500 gallons.",
"Ms. B pumped 3/4 as much water as Wanda on the first day, which is
    3/4 * 4500 = 3375 gallons.",
"Combined, Wanda and Ms. B filled 4500 + 3375 = 7875 gallons on the
    first day.",
"On the second day, Wanda pumped 2/3 of the amount she pumped on
   the first day, which is 2/3 * 4500 = 3000 gallons.",
"Ms. B pumped 1/3 of the amount she pumped on the first day, which
   is 1/3 * 3375 = 1125 gallons.",
"Combined, Wanda and Ms. B filled 3000 + 1125 = 4125 gallons on the
    second day.",
"The total amount of water filled in the tank over the two days is
   7875 +
```

Analysis: Every completed step is arithmetically correct, but the single-pass decoder spends its entire budget on full-sentence steps and is cut off at `7875 +`, before it computes the total or reaches the answer. The output is invalid JSON and has no `answer` field.

Stage 1 Draft (shared by DCCD and DGCD)

On the first day, Wanda pumped $\frac{1}{4} \times 18000 = 4500$ gallons. Ms. B pumped $\frac{3}{4} \times 4500 = 3375$ gallons. Total for the first day: $4500 + 3375 = 7875$ gallons.
On the second day, Wanda pumped $\frac{2}{3} \times 4500 = 3000$ gallons. Ms. B pumped $\frac{1}{3} \times 3375 = 1125$ gallons. Total for the second day: $3000 + 1125 = 4125$ gallons.
Total water pumped: $7875 + 4125 = 12000$ gallons. Remaining gallons: $18000 - 12000 = 6000$.

$$\boxed{6000}$$

DCCD Stage 2 Output (AR structurer; truncated, invalid JSON)

```
{
  "steps": [
    "On the first day, Wanda pumped \\( \\frac{1}{4} \\times 18000
    = 4500 \\) gallons.",
    "Ms. B pumped \\( \\frac{3}{4} \\times 4500 = 3375 \\) gallons
    .",
    "Total for the first day: \\( 4500 + 3375 = 7875 \\) gallons.",
    "On the second day, Wanda pumped \\( \\frac{2}{3} \\times 4500
    = 3000 \\) gallons.",
    "Ms. B pumped \\( \\frac{1}{3} \\times 3375 = 1125 \\) gallons
    .",
    "Total for the second day: \\( 3000 + 1125 = 4125 \\) gallons
    .",
    "Total water pumped: \\( 7875 + 4125 = 12000 \\) gallons.",
    "Remaining gallons: \\( 18000 -
```

Analysis: The Stage 1 draft is correct and only 241 tokens long, yet the AR structurer cannot fit it into 250 tokens. It copies the draft almost sentence for sentence, and JSON serialization makes the copy *longer* than the original: each string gains quotes, indentation, and a trailing comma, and every LaTeX backslash is escaped as \\. The output stops just before the final value, at `18000 -`. Although the draft contains the correct answer, the structured output is invalid JSON with no `answer` field.

DGCD uses the same Stage 1 draft. In Stage 2, LLaDA-8B-Instruct fills the fixed template {"steps": ["<s_1:64>", "<s_2:64>", "<s_3:64>"], "answer": "<a:32>"}.

DGCD Stage 2 Output (diffusion structurer; valid, correct)

```
{
  "steps": [
    "On the first day",
    "Total for the first day: \\( 4500 + 3375 = 7875 \\) gallons",
    "Ms. B pumped \\( \\frac{1}{3} \\times 3375 = 1125 \\) gallons"
  ],
  "answer": "Remaining gallons: \\( 18000 - 12000 = 6000 \\)\\boxed
    {6000}"
}
```

Analysis: The output is valid JSON and contains the correct answer, 6000. The key difference lies in how the budget is allocated. The answer slot is reserved in advance and denoised jointly with the step slots rather than after them, so the structurer places the final line of the draft, `18000 - 12000 = 6000`, directly into it. The cost appears in the step slots: with only three slots, the structurer retains a fragment (`"On the first day"`), and the third step records only Ms. B's second-day contribution, omitting Wanda's second-day amount and the remaining totals. DGCD thus trades the completeness of the reasoning trace for a guaranteed, well-formed answer.

I.2 Example 2: MATH500 with a 500-Token Budget

Here CD and DCCD receive a 500-token budget, more than twice the structurer's 224 content tokens. Both nevertheless fail, owing to a different source of token inflation: escaped Unicode characters.

MATH500 Problem

Let $\mathbf{a}$ and $\mathbf{b}$ be vectors such that the angle between $\mathbf{a}$ and $\mathbf{b}$ is 29°, and the angle between $\mathbf{b}$ and $\mathbf{a} - \mathbf{b}$ is 84°. Find the angle between $\mathbf{a}$ and $\mathbf{a} - \mathbf{b}$.

Ground truth answer: $\boxed{55^\circ}$

Baseline Outputs (500-token budget)

Unicode escape sequences are shown decoded for readability, except for the final, truncated escape in each output.

CD: truncated, invalid JSON

```
{
"steps": [
"Let the angle between Δ and α be θ, and the angle between α and α
    - β be δ. We know θ = 29° and δ = 84°.",
  ... (3 steps omitted) ...
"Since α · β = |α| |β| cos(29°), we can substitute and simplify to
    get |α|^2 cos(29°) - |α| |β| cos(29°) = |α| |α - β| cos(84\u00
```

DCCD Stage 2 (AR structurer): truncated, invalid JSON

```
{
  "steps": [
    "Define θ as the angle between Σ and Σ - β",
      ... (5 steps omitted) ...
    "Use θ = arccos(Σ ∘ (Σ - β) / (Σ × Σ - β × β)) and simplify,
    find θ = 55°",
    "Thus, the angle between Σ and Σ - β is θ = 55\u
```

Analysis: Both AR decoders replace the bold vectors $\mathbf{a}$ and $\mathbf{b}$ with Greek letters and emit these characters as literal `\uXXXX` escape sequences, each costing five tokens. In the DCCD Stage 2 output, 62 escapes consume 310 of the 500 tokens, and generation stops in the middle of the escape for the degree sign after `55`. The draft reaches the correct answer, and the value 55 even appears in the final step, but the output lacks an `answer` field and is invalid JSON. CD fails earlier, before completing its reasoning.

DGCD: Stage 1 Draft and Stage 2 Output

Let θ be the angle between $\mathbf{a}$ and $\mathbf{a}-\mathbf{b}$. Using the dot product properties and the given angles, we can derive that $\cos 29^\circ = \mathbf{a}\cdot\mathbf{b}$ and $\cos 84^\circ = \frac{\mathbf{b}\cdot(\mathbf{a}-\mathbf{b})}{\|\mathbf{b}\|\|\mathbf{a}-\mathbf{b}\|}$. Simplifying, we find $\cos 84^\circ = \frac{\mathbf{a}\cdot\mathbf{b}-\|\mathbf{b}\|^2}{\|\mathbf{b}\|\|\mathbf{a}-\mathbf{b}\|}$. Using $\mathbf{a}\cdot\mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\|\cos 29^\circ$, we get $\cos 84^\circ = \frac{\|\mathbf{a}\|\cos 29^\circ - \|\mathbf{b}\|}{\|\mathbf{a}-\mathbf{b}\|}$. The angle θ satisfies $\cos\theta = \frac{\mathbf{a}\cdot(\mathbf{a}-\mathbf{b})}{\|\mathbf{a}\|\|\mathbf{a}-\mathbf{b}\|} = \frac{\|\mathbf{a}\|^2-\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{a}-\mathbf{b}\|} = \frac{\|\mathbf{a}\|-\|\mathbf{b}\|\cos 29^\circ}{\|\mathbf{a}-\mathbf{b}\|}$. Given the angles, we find $\theta = 55^\circ$. Thus, the angle between $\mathbf{a}$ and $\mathbf{a}-\mathbf{b}$ is $\boxed{55^\circ}$.

Stage 2 (diffusion structurer): valid, correct

```
{
  "steps": [
    "Let θ be the angle between a and a - b",
    "Simplifying, we find cos(84°) = a · b - ||b||^2 / ||b|| ||a - b
    ||",
    "The angle θ satisfies cos(θ) = a · (a - b) / ||a|| ||a - b|| =
    ||a||^2 - a · b / ||a|| ||a - b|| = (||a|| - ||b|| cos(29)) / ||
    a - b||"
  ],
  "answer": "Given the angles, we find θ = 55°. Thus, the angle
    between a and a - b is \\\\boxed{55\\\\circ}"
}
```

Analysis: The structurer stays within its 224-token template and produces valid JSON with the correct answer, 55°. It writes $\mathbf{a}$ and $\mathbf{b}$ as plain letters and θ, $\cdot$, and $^\circ$ as raw characters, so none of its fixed budget is spent on Unicode escapes. As in Example 1, the step slots contain a compressed trace: some intermediate derivations are omitted, and several fractions lose their grouping parentheses (e.g., the numerator in the second step). Because the answer slot is filled independently, it still captures the final sentence of the draft.

I.3 Example 3: Financial Entity Extraction with an 80-Token Budget

Extraction schemas aggravate budget cutoff in a different way. The AR decoder must write out every key, including those whose value is `null`, and in this example three of the four empty categories appear after the populated ones. Because decoding is greedy, the truncated outputs below correspond to the first 80 tokens of the full-budget outputs, which is exactly what a run capped at 80 tokens produces.

Financial Entity Extraction Problem

Idaho Power 's construction expenditures were $ 240 million , $ 338 million , and $ 338 million in 2012 , 2011 , and 2010 , respectively .
Ground truth: `Company: [Idaho Power]; Date: [2010, 2011, 2012]; Money: [$ 240 million, $ 338 million];` all other categories `null`.

CD and DCCD Stage 2 Output (identical; truncated, invalid JSON)

```
{
  "Company": ["Idaho Power"],
  "Date": ["2012", "2011", "2010"],
  "Location": null,
  "Money": ["$ 240 million", "$ 338 million", "$ 338 million"],
  "Person": null,
  "Product": null,
  "Quantity":
```

Analysis: With an unlimited budget, both methods recover every gold entity (repeating `$ 338 million`, which appears twice in the source), and their full outputs are 83 tokens long. At 80 tokens, all extracted content is present, but the object stops at `"Quantity":`, one `null` and a closing brace short of valid JSON. Only 48 of the 83 tokens are field values; the remainder are keys, quotes, and indentation, which the AR decoder must emit for every category, including the four empty ones, before it can close the object. The output is invalid and receives a score of zero.

DGCD: Stage 1 Draft and Stage 2 Output

```
Company: Idaho Power
Money: $ 240 million, $ 338 million, $ 338 million
Date: 2012, 2011, 2010
```

Stage 2 (diffusion structurer, illustrative): valid, correct

Every key and delimiter is fixed in the template. The seven value spans are sized according to the dataset's 95th-percentile gold length for each category and scaled to the same 80-token total (Company 17, Date 20, Location 7, Money 15, Person 6, Product 7, Quantity 8).

```
{"Company": ["Idaho Power"], "Date": ["2012", "2011", "2010"], "
    Location": null, "Money": ["$ 240 million", "$ 338 million"], "
    Person": null, "Product": null, "Quantity": null}
```

Analysis: The draft lists only the three non-empty categories, using 47 tokens. In the DGCD template, all keys are already in place, so each empty category requires only a `null` within its reserved span and cannot delay the closing of the object. The fixed spans also cap the length of each field: the draft's repeated `$ 338 million` does not fit within the 15-token Money span, so the two-item list, which matches the gold annotation, is the one that fits.