Don't Rewrite My Text: Sidetrack Decoding for Faithful Document Annotation

Anonymous ACL submission

Abstract

When prompted to *annotate* some text, Large Language Models (LLMs) may introduce subtle modifications that break the alignment between generated and source text, and cannot be reliably used for structured generation tasks. We propose **sidetrack decoding**, a constrained decoding algorithm that guarantees verbatim text preservation during text annotation. By inspecting the model's next-token probabilities over the original text, the algorithm determines when to "sidetrack" into generation mode to insert markup tokens, and when to merge back to the main track to copy the original text. We evaluate on three extraction benchmarks and find that sidetrack decoding achieves 100% text preservation while maintaining competitive extraction accuracy on tasks suited for span tagging. A speculative execution variant further improves efficiency, achieving 3.3× throughput compared to standard autoregressive decoding.

1 Introduction

LLMs are increasingly used to extract structured information from text—named entity recognition, relation extraction, and document annotation. A natural approach is *span tagging*: prompting the model to rewrite the input document with XML-like tags inserted around relevant spans. This preserves the connection between extracted values and their source locations—enabling attribution that can help users verify outputs (Rashkin et al., 2022; Menick et al., 2022; Hennigen et al., 2024). Such verbatim preservation is critical in domains like healthcare, legal, and finance, where modified text can alter meaning, break audit trails, or violate compliance requirements.

However, LLMs struggle to copy text faithfully. When prompted to annotate documents, models frequently introduce subtle modifications—correcting typos, normalizing formatting, or omitting words (Table 2). In our experiments, even the best model

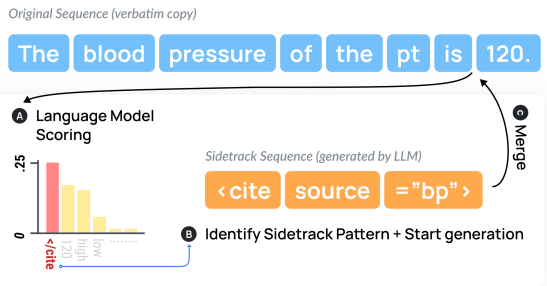

Figure 1: Illustration of the sidetrack decoding process.

(Gemini-2.5-flash) achieves only 92% perfect copy rate. This breaks the alignment between tagged spans and the original source, undermining the very attribution that span tagging is meant to provide. While prior work has studied attribution frameworks (Rashkin et al., 2022; Bohnet et al., 2023) and constrained decoding for syntactic validity (Microsoft, 2023; .txt, 2023), verbatim text preservation during annotation remains unaddressed (see Section A for detailed discussion).

We propose **sidetrack decoding**, a constrained decoding algorithm that *guarantees* verbatim text preservation at the token level while maintaining competitive extraction accuracy. We reframe structured generation as *text insertion*: rather than regenerating the entire document, we only need to determine *where* to insert markup tokens. Sidetrack decoding monitors the model's next-token predictions; when the model predicts a tag-opening token (e.g., <span) with high probability, we "sidetrack" into generation mode to produce the tag, then return to copying the original text.

We evaluate sidetrack decoding on three structured extraction benchmarks and find that it achieves 100% text preservation while maintaining competitive extraction accuracy on tasks where span tagging is appropriate. Because sidetrack decoding operates at the subword token level, it cannot insert tags at boundaries that fall within a single token—a limitation we discuss in Section 4. We

also implement a speculative execution variant that processes windows of tokens in parallel, achieving 3.3× higher throughput than standard autoregressive decoding. vspace-1mm

2 Sidetrack decoding

vspace-1mm

Many structured generation problems can be formulated as a text editing task: Given an existing sequence of words $w_1, w_2, w_3, \ldots, w_n$, the task is to find positions to insert some tokens. That is, we want to predict the index k_j and the new sequences to be inserted $w_1^{(j)}, w_2^{(j)}, \ldots, w_{n_j}^{(j)}$ at that position. The final text becomes

$$w_1, \ldots, w_{k_j}, w_1^{(j)}, w_2^{(j)}, \ldots, w_{n_j}^{(j)}, w_{k_j+1}, \ldots$$

For example, the j and $j+1$-th insertion can be `<span class="title">` and `</span>` that labels One can leverage an autoregressive language model for this task with few-shot prompts. However, it is slow in that they need to repeat the same tokens from scratch and, most importantly, *there is no guarantee that the existing text will be preserved.*

We propose a new constrained decoding method called sidetrack decoding to overcome the limitations of a naive prompting approach. As shown in Fig. 1, the overall goal is to have models search for locations in the original sequence to sidetrack from the main decoding path to *generate* the edits only, and then verbatim copy the rest original text.

Formally, for an autoregressive LM $\mathcal{M}$, for a given sequence S, we can obtain the next-token probability distributions at each position given the prefix of S up to that position: $P_{\mathcal{M}}(\cdot \mid S_{<t}), \forall t \in [1, |S|]$. We define the function $\textsc{TopLogProbs}(S; \mathcal{M}, k, \theta)$ that returns a list of up to k tokens with top probabilities above a threshold θ given the language model $\mathcal{M}$ and the prompt S. By inspecting the output, we can locate positions where the language model's predictions deviate from the original sequence S. Specifically, we define a sidetrack tracker T as a tuple of start r_{start} and stop r_{stop} regular expressions. We say that a token w is a sidetrack position if $w \in r_{\text{start}}$. However, sometimes the tokenization of the language model may require multiple tokens generated to satisfy a sidetrack pattern: for example, it needs to generate two tokens—"␣<" and "span"—to satisfy the start regex pattern `^<span`. In that case, we need to have **a lookahead search** with the LM $\mathcal{M}$ to find the needed tokens to determine if a position is a sidetrack position. The $\textsc{Satisfy}(w; T, \mathcal{M})$ function implements this lookahead search.

Sidetrack Decoding with Naive Search. Given the context C (optional, which provides the needed instruction for the token insertion task) and a target sequence S to modify, we can iterate over each token in S to look for sidetrack positions. Algorithm 1 implements this process: Given the combined context $C \oplus S$, we can use $\textsc{TopLogProbs}$ to score the sequence and $\textsc{Satisfy}$ to check whether to "sidetrack". When the condition satisfies, it switches to the "generation" mode and generates the corresponding tokens. To allow for **consecutive sidetracking** (i.e., generating `<span>` right after `</span>`), we need to check the sidetrack condition again at the same location after generating the sidetrack tokens. As such, we use separate pointers to track the index i relative to the original sequence S and the sidetrack length, and only increment i when the sidetrack condition is not satisfied.

Sidetrack Decoding with Speculative Execution. The naive search algorithm above is inefficient: it doesn't reduce the number of needed forward passes while the needed sidetrack location is often only a small fraction of the target sequence length $|S|$. To speed up the process, we adopt a similar idea as the speculative decoding (Leviathan et al., 2023).[1] Instead of obtaining $\textsc{TopLogProbs}$ per token, we run it on a window of consecutive w tokens in S. We define $\textsc{BlockTopLogProbs}(S, W; \mathcal{M}, k, \theta)$ that takes a prompt S and a window of tokens $W = [w_1, w_2, \ldots, w_w]$, and returns a list of top-k token candidates above threshold θ for each position in the window by computing $P_{\mathcal{M}}(w_i \mid S \oplus W_{<i})$ for $i \in [1, |W|]$. When it encounters a sidetrack position at $k < w$, we discard the token probability distribution after the k-th token, and then execute the sidetrack generation. Algorithm 2 implements this speculative execution approach. The window size w is a hyperparameter that can be tuned to optimize the efficiency given the frequency of sidetrack positions.

3 Experiments

We evaluate sidetrack decoding on structured information extraction tasks, measuring both extraction accuracy and text preservation.

[1] A direct analogy is that we use the original substrings of S as the "draft model", and here the language model $\mathcal{M}$ itself is the "verifier model".

Table 1: Extraction performance across three structured output benchmarks. FA = Field Accuracy, OA = Output Accuracy, F1 = F1 Score, d = Edit Distance, R = Robustness (Perfect Copy Ratio). All values in % except d.

	Financial NER			**Claims**			**PII Extraction**			**Average**			
Model	FA	OA	R	FA	F1	R	FA	OA	R	FA	OA	d	R
JSON Generation													
`GPT-4o-mini`	66.8	46.0	—	82.2	81.5	—	61.1	31.0	—	70.0	26.8	—	—
`Gemini-2.5-flash`	**71.9**	**55.0**	—	**89.1**	**87.4**	—	**81.5**	**58.0**	—	**80.9**	**46.6**	—	—
`Claude Haiku 4.5`	68.5	48.0	—	88.1	86.2	—	80.2	56.0	—	79.0	36.9	—	—
`Qwen3-4B`	60.0	41.0	—	64.4	71.0	—	65.4	37.0	—	63.3	27.1	—	—
`Qwen3-8B`	63.0	42.0	—	82.0	83.8	—	65.8	39.0	—	70.3	28.1	—	—
Span Generation with Regular Decoding													
`GPT-4o-mini`	53.2	34.0	76.0	32.8	39.6	3.3	67.1	34.0	92.0	51.0	22.7	25.5	57.1
`Gemini-2.5-flash`	**68.5**	**46.0**	**97.0**	**40.9**	42.6	**83.3**	**76.5**	**53.0**	95.0	**62.0**	**33.0**	1.5	**91.8**
`Claude Haiku 4.5`	60.9	42.0	84.0	37.2	43.1	6.7	75.5	52.0	**98.0**	57.8	31.3	20.3	62.9
`Qwen3-4B`	57.9	39.0	71.0	45.1	51.6	0.0	65.8	36.0	73.0	56.2	25.0	106.5	48.0
`Qwen3-8B`	60.9	39.0	91.0	**49.6**	**56.3**	13.3	67.1	39.0	70.0	59.2	26.0	62.4	58.1
Span Generation with Sidetrack Generation													
`Qwen3-4B`	60.4	39.0	97.0	24.5	33.0	96.7	62.1	35.0	**100**	49.0	24.7	3.8	97.9
`Qwen3-8B`	**63.8**	**41.0**	**100**	**29.1**	**36.0**	**100**	**65.4**	**38.0**	**100**	**52.8**	**26.3**	**0.0**	**100**

Datasets. We use three datasets from the Structured Output Benchmark (Goh and Mueller, 2025): **Financial Relation Extraction (FIRE)** contains financial news text annotated with entity types including Company, Person, Location, Date, Money, Product, and Quantity; **PII Entity Recognition** contains text with 56 types of personally identifiable information (e.g., credit card numbers, emails, SSNs, phone numbers); **Insurance Claims Extraction** involves extracting hierarchical structured fields from insurance claim documents across sections such as policy details, insured objects, and incident descriptions. Full prompts and dataset examples are provided in Appendix D.

Baselines. We compare against two baselines: *JSON Generation*, where models are prompted to directly output extracted values as structured JSON (e.g., `{"AMOUNT": "964480"...}`); and *Span Generation*, standard autoregressive decoding where the model generates the entire annotated text with inline span tags without constrained copying (See the prompts in Section D).

Models and Implementation. We evaluate on both open-source models (Qwen-4B and Qwen-8B (Yang et al., 2025)) and commercial models (GPT-4o-mini, Gemini-2.5-Flash, and Claude Haiku 4.5). For JSON generation, we allow a maximum of 16,384 output tokens. For span generation, we set temperature to 0 for deterministic outputs and allow up to 8,192 thinking tokens before generating the final output. We implement sidetrack decoding on top of the SGLang framework (Zheng et al., 2024), which provides efficient batched inference with prefix caching. All experiments are conducted on NVIDIA H100 GPUs. Similarly, we use temperature 0 and allow 8,192 reasoning tokens before sidetrack decoding.

Evaluation Metrics. We report accuracy metrics using *partial matching*, where a prediction is correct if it contains the ground-truth value as a substring (see Appendix B for rationale): **Field Accuracy** (FA), the proportion of ground-truth fields correctly extracted; **Output Accuracy** (OA), the proportion of samples where all fields are correct; and **F1 Score**, the harmonic mean of precision and recall. We also report robustness metrics (ignoring whitespace): **Edit Distance** (d), the average Levenshtein distance between original input and output after stripping span tags; and **Perfect Copy Ratio** (R), the proportion of samples with exact text preservation. Sidetrack decoding guarantees $\mathrm{R} = 100\%$ by construction.

4 Results

Span generation is competitive with JSON generation when source text is clean. On Financial NER (Table 1), where outputs are flat entity types and source text is well-formed, span generation achieves accuracy close to JSON generation (e.g.,

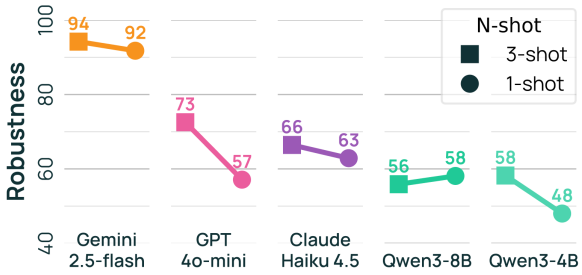

Figure 2: Generation robustness decreases with fewer in-context examples.

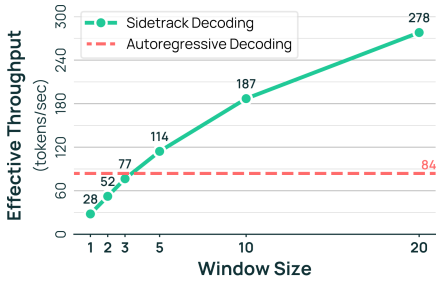

Figure 3: Effective throughput of sidetrack decoding on Qwen3-8B as window size increases.

68.5% vs 71.9% FA for Gemini-2.5-flash). However, on Claims extraction with complex hierarchical outputs and noisy informal emails, span generation significantly underperforms JSON generation (40.9% vs 89.1% FA). This is expected: when normalization of source text is desirable, JSON generation is more suitable than span tagging.

Sidetrack decoding guarantees robustness without sacrificing accuracy. For tasks like Financial NER and PII extraction, where span generation is appropriate, sidetrack decoding guarantees perfect text preservation ($\mathrm{R} = 100\%$, $d = 0$) while maintaining competitive accuracy. On Qwen3-8B, sidetrack decoding actually *improves* Financial NER accuracy (60.9%→63.8%), with minimal gap on PII (65.4% vs 67.1%).

Robust generation is non-trivial. Standard span generation shows variable robustness: GPT-4o-mini achieves 57.1%, Claude Haiku 4.5 achieves 62.9%, and even the best (Gemini-2.5-flash at 91.8%) falls short of 100%. As Figure 2 shows, more in-context examples help (GPT-4o-mini: 57%→73% from 1-shot to 3-shot), but no prompting strategy achieves perfect preservation. Robustness varies across models and is not simply a function of capability—constrained decoding is necessary for perfect robustness.

Speculative execution significantly improves throughput. Figure 3 shows efficiency gains on Qwen3-8B. We measure *effective throughput*: total output tokens divided by decoding time, enabling direct comparison with standard autoregressive decoding. With window size 1, sidetrack decoding achieves only 28 tokens/sec (vs. 84 for autoregressive). However, window sizes >3 surpass autoregressive decoding, reaching 278 tokens/sec at window size 20—a 3.3× improvement.

Case study: verbatim preservation trade-offs. Table 2 shows a PII extraction example where standard decoding removes "VIN" from the output, breaking alignment with the source despite correct entity identification. With sidetrack decoding, the original text is preserved verbatim, including the "VIN" prefix. This example highlights both the benefit of sidetrack decoding—guaranteed text preservation—and its constraint: the model cannot normalize or clean up the input even when doing so might be desirable. A further limitation is that sidetrack decoding operates at the subword token level, so it cannot insert tags at boundaries that fall within a single token (e.g., tagging only "York" in "NewYork" if tokenized as one unit).

Table 2: Comparison of standard vs. sidetrack decoding outputs on a PII extraction example (Qwen3-8B). Sidetrack decoding preserves the original text verbatim, while standard decoding may inadvertently modify it.

Standard Decoding
Dear Norbert, welcome onboard. For your vehicle, the Parking space number is next to [missing: VIN] 05BHMWB78RK026069.
Sidetrack Decoding
Dear Norbert , welcome onboard. For your vehicle, the Parking space number is next to VIN 05BHMWB78RK026069 .

5 Conclusion

We present sidetrack decoding, a constrained decoding algorithm that guarantees verbatim text preservation for structured extraction tasks. By monitoring next-token probabilities and selectively "sidetracking" to generate markup tokens, our approach achieves 100% text preservation while maintaining competitive extraction accuracy on tasks suited for span tagging. A speculative execution variant further improves efficiency, achieving 3.3× throughput over standard autoregressive decoding. Sidetrack decoding is particularly suited for applications requiring verifiable outputs with precise source attribution, though it cannot normalize text or insert tags within subword token boundaries.

Limitations

Sidetrack decoding operates at the subword token level, meaning it cannot insert tags at arbitrary character positions within a token (e.g., tagging only "York" in "NewYork" if tokenized as one unit). Additionally, by design, sidetrack decoding preserves source text verbatim and cannot normalize, correct, or transform the input; for tasks where text cleaning is desirable, JSON generation may be more appropriate.

Potential Risks. Sidetrack decoding is a constrained decoding algorithm that does not introduce risks beyond those inherent to standard LLM-based extraction. Because the method preserves source text verbatim, any errors, misinformation, or sensitive content in the original document will be retained in the output. Users should not interpret verbatim preservation as validation of the source content's accuracy.

Use of AI Assistants. AI assistants were used to support both programming and writing in this work. We used Claude Code and Cursor (with Claude and Gemini models) during the implementation and experimentation process. We also used these models to proofread the paper and check for grammatical errors.

References

1rgs. 2023. Jsonformer: A bulletproof way to generate structured JSON from language models. https://github.com/1rgs/jsonformer. Accessed: Jan 6, 2026.

Bernd Bohnet, Vinh Q. Tran, Pat Verga, Roee Aharoni, Daniel Andor, Livio Baldini Soares, Massimiliano Ciaramita, Jacob Eisenstein, Kuzman Ganchev, Jonathan Herzig, Kai Hui, Tom Kwiatkowski, Ji Ma, Jianmo Ni, Lierni Sestorain Saralegui, Tal Schuster, William W. Cohen, Michael Collins, Dipanjan Das, and 3 others. 2023. Attributed question answering: Evaluation and modeling for attributed large language models. *Preprint*, arXiv:2212.08037.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations. *Preprint*, arXiv:2305.14627.

Hui Wen Goh and Jonas Mueller. 2025. LLM structured output benchmarks are riddled with mistakes. https://cleanlab.ai/blog/structured-output-benchmark/. Accessed: Jan 5, 2026.

Lucas Torroba Hennigen, Shannon Shen, Aniruddha Nrusimha, Bernhard Gapp, David Sontag, and Yoon Kim. 2024. Towards verifiable text generation with symbolic references. *Preprint*, arXiv:2311.09188.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Nelson F. Liu, Tianyi Zhang, and Percy Liang. 2023. Evaluating verifiability in generative search engines. *ArXiv*, abs/2304.09848.

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, and Nat McAleese. 2022. Teaching language models to support answers with verified quotes. *Preprint*, arXiv:2203.11147.

Microsoft. 2023. Guidance: A guidance language for controlling large language models. https://github.com/guidance-ai/guidance. Accessed: Jan 6, 2026.

Hannah Rashkin, Vitaly Nikolaev, Matthew Lamm, Lora Aroyo, Michael Collins, Dipanjan Das, Slav Petrov, Gaurav Singh Tomar, Iulia Turc, and David Reitter. 2022. Measuring attribution in natural language generation models. *Preprint*, arXiv:2112.12870.

.txt. 2023. Outlines: Structured text generation. https://github.com/outlines-dev/outlines. Accessed: Jan 6, 2026.

Theodora Worledge, Judy Hanwen Shen, Nicole Meister, Caleb Winston, and Carlos Guestrin. 2024. Unifying corroborative and contributive attributions in large language models. In *2024 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

A Related Work

Attribution in LLM Generation. Our work connects to the broader challenge of attribution in LLM outputs. Prior frameworks define when generations are "Attributable to Identified Sources" (AIS) (Rashkin et al., 2022; Bohnet et al., 2023), and methods exist for generating text with explicit citations (Gao et al., 2023; Liu et al., 2023). Worledge et al. (2024) provide a unified framework distinguishing *corroborative* attribution (citing evidence that supports a claim) from *contributive* attribution (identifying training data that influenced generation). Our work addresses a complementary concern: preserving source text verbatim so that extracted spans remain verifiable against the original document—what might be called *source-preserving* attribution.

Constrained Decoding for Structured Generation. Constrained decoding methods ensure LLM outputs conform to specified formats. Tools like Guidance (Microsoft, 2023) and Outlines (.txt, 2023) use grammar-based constraints to guarantee syntactically valid outputs (e.g., valid JSON). JSONformer (1rgs, 2023) similarly ensures JSON schema compliance. However, these methods focus on *output format* rather than *input preservation*—they do not address the problem of maintaining verbatim alignment with source text during annotation tasks.

Speculative Decoding. Our speculative execution variant draws inspiration from speculative decoding (Leviathan et al., 2023), which accelerates inference by using a smaller "draft" model to propose tokens that are then verified by the target model. In our setting, the original text itself serves as the draft, and we verify whether the model wants to insert markup at each position.

B Evaluation Metrics and Additional Results

We evaluate structured extraction tasks using the following metrics.

Accuracy Metrics. Let N be the total number of samples, G_i the set of ground-truth fields for sample i, and P_i the set of predicted fields. We define:

$$\mathrm{FA} = \frac{\sum_i |\{f \in G_i : \textsc{Match}(f, P_i)\}|}{\sum_i |G_i|} \tag{1}$$

$$\mathrm{OA} = \frac{|\{i : \forall f \in G_i,\ \textsc{Match}(f, P_i)\}|}{N} \tag{2}$$

Field Accuracy (FA) measures the proportion of ground-truth fields correctly extracted. **Output Accuracy** (OA) measures the proportion of samples where *all* ground-truth fields are correct—a stricter metric for applications requiring complete correctness. **F1 Score** is the harmonic mean of precision $\mathrm{P} = |\textsc{Match}|/|P|$ and recall $\mathrm{R} = |\textsc{Match}|/|G|$:

$$\mathrm{F1} = \frac{2 \cdot \mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{3}$$

Partial vs. Exact Match. The $\textsc{Match}$ function can operate in two modes. In **exact match**, the normalized predicted value must equal the ground truth: $\textsc{Match}(f, P) \Leftrightarrow \texttt{norm}(P_f) = \texttt{norm}(G_f)$. In **partial match**, the ground truth need only appear as a substring: $\textsc{Match}(f, P) \Leftrightarrow \texttt{norm}(G_f) \subseteq \texttt{norm}(P_f)$.

Partial matching is the appropriate evaluation mode for sidetrack decoding. Because sidetrack decoding cannot modify the original text, extracted spans may contain additional context beyond the minimal entity value (see Table 2: the span includes "VIN" alongside the actual vehicle identification number). We report partial-match metrics as the primary results (FA, OA, F1) and exact-match metrics with a star ($\mathrm{FA}^\star$, $\mathrm{OA}^\star$, $\mathrm{F1}^\star$) for completeness. Table 3 reports the full exact-match results.

Robustness Metrics. Edit Distance (d) measures the average character-level Levenshtein distance between the original input and the model output after stripping span tags, ignoring whitespace. **Perfect Copy Ratio** (R) measures the proportion of samples with exact text preservation (ignoring whitespace). Sidetrack decoding guarantees $\mathrm{R} = 100\%$ by construction.

Table 3: Extraction performance with **exact match** evaluation across three structured output benchmarks. FA* = Field Accuracy (exact), OA* = Output Accuracy (exact), F1* = F1 Score (exact), d = Edit Distance, R = Robustness (Perfect Copy Ratio). All values in % except d.

	Financial NER			**Claims**			**PII Extraction**			**Average**			
Model	FA*	OA*	R	FA*	F1*	R	FA*	OA*	R	FA*	OA*	d	R
JSON Generation													
`GPT-4o-mini`	54.0	31.0	—	80.2	79.5	—	60.4	31.0	—	64.9	21.8	—	—
`Gemini-2.5-flash`	**60.0**	**40.0**	—	85.8	84.1	—	**81.2**	**57.0**	—	**75.7**	**33.4**	—	—
`Claude Haiku 4.5`	59.1	36.0	—	**87.0**	**85.0**	—	80.2	56.0	—	75.4	31.8	—	—
`Qwen3-4B`	50.6	31.0	—	62.3	68.6	—	65.4	37.0	—	59.4	23.8	—	—
`Qwen3-8B`	54.9	33.0	—	79.1	80.8	—	64.8	37.0	—	66.2	24.4	—	—
Span Generation with Regular Decoding													
`GPT-4o-mini`	51.5	31.0	76.0	31.2	37.7	3.3	62.8	30.0	92.0	48.5	20.3	25.5	57.1
`Gemini-2.5-flash`	**61.7**	**42.0**	**97.0**	**37.2**	38.7	**83.3**	**75.5**	**53.0**	95.0	**58.1**	**31.7**	1.5	**91.8**
`Claude Haiku 4.5`	55.7	38.0	84.0	36.2	**42.0**	6.7	**75.5**	52.0	**98.0**	55.8	30.0	20.3	62.9
`Qwen3-0.6B`	32.8	20.0	17.0	26.1	32.3	0.0	25.2	6.0	23.0	28.0	8.7	450.0	13.3
`Qwen3-1.7B`	53.6	31.0	22.0	35.6	39.5	0.0	40.6	11.0	51.0	43.3	14.0	373.1	24.3
`Qwen3-4B`	54.0	35.0	71.0	42.1	48.2	0.0	65.1	35.0	73.0	53.7	23.3	106.5	48.0
`Qwen3-8B`	59.6	38.0	91.0	48.0	54.5	13.3	65.8	37.0	72.0	57.8	25.0	62.1	58.8
Span Generation with Sidetrack Decoding													
`Qwen3-0.6B`	18.7	12.0	99.0	7.9	13.7	100.0	13.4	2.0	100.0	13.4	4.7	0.3	99.7
`Qwen3-1.7B`	34.5	18.0	94.0	10.3	16.1	100.0	32.9	8.0	100.0	25.9	8.7	2.0	98.0
`Qwen3-4B`	50.6	33.0	97.0	20.9	28.2	96.7	55.4	28.0	**100**	42.3	20.3	3.8	97.9
`Qwen3-8B`	**60.9**	**38.0**	**100**	**22.1**	**27.5**	**100**	**58.4**	**27.0**	**100**	**47.1**	**21.7**	**0.0**	**100**

C Full Algorithms of Sidetrack Decoding

We provide the full algorithms of sidetrack decoding with naive search and speculative execution in Algorithm 1 and Algorithm 2.

D Dataset Details and Prompts

The original FIRE dataset contains 2.2k examples, from which we use the first 100; PII Entity Recognition contains 100 examples; and Insurance Claims Extraction contains 30 examples. Below we provide the full prompts used for each dataset in our evaluation.

D.1 FIRE (Financial Relation Extraction)

FIRE Prompt

(System Message)

`Your task is to identify and annotate financial named entities in the given text. Annotate each entity by wrapping it with an HTML span tag, using the entity type as the class attribute.`

Entity Types

- Company: Business organizations (e.g., Apple, Microsoft, Goldman Sachs)
- Date: Temporal references (e.g., February 21, 2024, last quarter)
- Location: Geographic identifiers (e.g., New York, EU, India)
- Money: Financial amounts (e.g., $117 billion, Rs 960 crore)
- Person: Individual names (e.g., Elon Musk, Warren Buffett)
- Product: Items or services (e.g., iPhone 15, Model S)
- Quantity: Numerical values (e.g., 40 million shares, 2 percent)

Example Annotations

Algorithm 1 Sidetrack Decoding with Naive Search

Require: LLM $\mathcal{M}$, Context C, Target sequence S, Sidetrack trackers $\mathcal{T}$, Top-k, Probability threshold θ
Ensure: Augmented sequence S' with inserted tokens specified by the sidetrack trackers

Initialize: $S' \leftarrow S$, offset $\leftarrow 0$, position $i \leftarrow 0$
while $i \leq |S|$ **do**
  pos $\leftarrow i +$ offset ▷ Actual position in modified sequence
  candidates $\leftarrow$ TopLogProbs$(C \oplus S'_{:\text{pos}}; \mathcal{M}, k, \theta)$ ▷ Tokens with prob $> \theta$
  tracker $\leftarrow$ None ▷ Check if any candidate satisfies a sidetrack tracker
  for each tracker $T_j \in \mathcal{T}$ **do**
    for each candidate $\in$ candidates **do**
      if Satisfy(candidate, T_j) **then**
        tracker $\leftarrow T_j$
        break ▷ Stops greedily
  if tracker $\neq$ None **then** ▷ Execute sidetrack if activated
    generated $\leftarrow$ Generate$(C \oplus S'[:\text{pos}] \oplus \text{candidate}; \mathcal{M}, \text{stop} = \text{tracker.stop})$
    sidetrack_tokens $\leftarrow$ candidate $\oplus$ generated
    $S' \leftarrow S'[:\text{pos}] \oplus$ sidetrack_tokens $\oplus S'[\text{pos}:]$
    offset $\leftarrow$ offset $+ |$sidetrack_tokens$|$ ▷ Don't increment i - recheck the position again
  else
    $i \leftarrow i + 1$ ▷ Move to next text position
return S'

- Apple reported earnings of $94.8 billion
- Tim Cook announced the new iPhone 15

Important: Copy the text exactly, only adding span tags around named entities. You should tag ALL the named entities in a sentence whenever it matches the entity types listed above, and you should not miss any.

Here are some examples:
{n-shot examples}

(Example Generation)
Input
Albertsons and Rite Aid called off the planned merger after several advisory firms recommended voting against the proposal .

Annotated Output
 Albertsons and Rite Aid called off the planned merger after several advisory firms recommended voting against the proposal .

D.2 PII Entity Recognition

PII Entity Recognition Prompt

(System Message)
Your task is to identify and annotate PII (Personally Identifiable Information) entities in the given text. Annotate each PII entity by wrapping it with an HTML span tag, using the entity type as the class attribute.

Format Examples

- Credit card numbers: 1234567890123456
- Names: John
- Emails: john@example.com

Available PII types include: ACCOUNTNAME, ACCOUNTNUMBER, AGE, AMOUNT, BIC, BITCOINADDRESS, BUILDINGNUMBER, CITY, COMPANYNAME, COUNTY, CREDITCARDCVV, CREDITCARDISSUER, CREDITCARDNUMBER, CURRENCY, CURRENCYCODE, DATE, DOB, EMAIL, ETHEREUMADDRESS, EYECOLOR, FIRSTNAME, GENDER, HEIGHT, IBAN, IP, IPV4, IPV6, JOBTITLE, JOBAREA, JOBTYPE, LASTNAME, LITECOINADDRESS, MAC, MASKEDNUMBER,

Algorithm 2 Sidetrack Decoding with Speculative Execution

Require: LLM $\mathcal{M}$, Context C, Target sequence S, Sidetrack trackers $\mathcal{T}$, Top-k, Probability threshold θ, Window size w

Ensure: Augmented sequence S' with inserted tokens specified by the sidetrack trackers

Initialize: $S' \leftarrow S$, offset $\leftarrow 0$, position $i \leftarrow 0$
while $i \leq |S|$ **do**
  pos $\leftarrow i +$ offset ▷ Actual position in modified sequence
  window_end $\leftarrow \min(i + w, |S|)$ ▷ Define window of size w
  window_tokens $\leftarrow S[i :$ window_end$]$
  candidates_block $\leftarrow$ BLOCKTOPLOGPROBS$(C \oplus S'_{:\text{pos}}, \text{window_tokens}; \mathcal{M}, k, \theta)$
  ▷ Returns list of top-k candidates for each position in window
  sidetrack_found $\leftarrow$ False
  for $j \leftarrow 0$ to $|\text{window_tokens}| - 1$ **do**
    candidates $\leftarrow$ candidates_block$[j]$ ▷ Candidates at position j in window
    tracker $\leftarrow$ None
    for each tracker $T_m \in \mathcal{T}$ **do**
      for each candidate $\in$ candidates **do**
        if SATISFY(candidate, T_m) **then**
          tracker $\leftarrow T_m$
          break
    if tracker $\neq$ None **then** ▷ Sidetrack found at position j in window
      generated $\leftarrow$ GENERATE$(C \oplus S'[: \text{pos} + j] \oplus \text{candidate}; \mathcal{M}, \text{stop} = \text{tracker.stop})$
      sidetrack_tokens $\leftarrow$ candidate $\oplus$ generated
      $S' \leftarrow S'[: \text{pos} + j] \oplus \text{sidetrack_tokens} \oplus S'[\text{pos} + j :]$
      offset $\leftarrow$ offset $+ |\text{sidetrack_tokens}|$
      $i \leftarrow i + j$ ▷ Advance to sidetrack position
      sidetrack_found $\leftarrow$ True
      break ▷ Restart from sidetrack position
  if $\neg$sidetrack_found **then**
    $i \leftarrow i + |\text{window_tokens}|$ ▷ No sidetrack, advance by window size
return S'

MIDDLENAME, NEARBYGPSCOORDINATE, ORDINALDIRECTION, PASSWORD, PHONEIMEI, PHONENUMBER, PIN, PREFIX, SECONDARYADDRESS, SEX, SSN, STATE, STREET, TIME, URL, USERAGENT, USERNAME, VEHICLEVIN, VEHICLEVRM, ZIPCODE

Important: Only annotate entities that are actually PII. Copy the text exactly, only adding span tags around PII entities.

Here are some examples:
{n-shot examples}

(Example Generation)

Input

Dear valued patient, keep in mind that your Eating Disorder therapy session has been scheduled at 12. The therapy cost Tala 964480 is to be paid via mastercard card and confirmed via 73-675301-103047-0 number.

Annotated Output

Dear valued patient, keep in mind that your Eating Disorder therapy session has been scheduled at 12 . The therapy cost Tala   964480 is to be paid via mastercard card and confirmed via 73-675301-103047-0 number.

D.3 Insurance Claims Extraction

Insurance Claims Extraction Prompt

(System Message)
Your task is to extract structured information from insurance claim documents. Annotate each piece of relevant information with HTML span tags using hierarchical field names.

Field Types to Extract

- header.claim_id, header.report_date, header.incident_date, header.reported_by, header.channel
- policy_details.policy_number, policy_details.policyholder_name, policy_details.coverage_type, policy_details.effective_date, policy_details.expiration_date
- insured_objects.object_id, insured_objects.object_type, insured_objects.make_model, insured_objects.year, insured_objects.location_address, insured_objects.estimated_value
- incident_description.incident_type, incident_description.location_type, incident_description.estimated_damage_amount, incident_description.police_report_number

Example Annotation Format

- Claim CLM-424063 was filed on 2024-01-15
- Policy holder John Smith has coverage type Auto

Important: Copy the text exactly, only adding span tags around relevant information. For multiple insured objects, use indexed paths like insured_objects.0.object_id, insured_objects.1.object_id.

Here are some examples:
{n-shot examples}

(Example Generation)
Input
[Email received: 2024-04-22]
Subject: Property claim from Jennifer Garcia—CLM-424063 (Shopping Center Incident)
Hi—this is Jennifer Garcia, and I'm writing because, honestly, I'm still kind of in shock about what happened this morning. They gave me this claim reference CLM-424063 when I called the help line earlier, but I figured I should also email so everything's in writing—it's just been a really stressful day, and I want to make sure nothing gets missed. Not sure how these things usually work, I guess you'll get back to me? Anyway...

I'm the owner on Policy POL-787532354, with property coverage for my home at 6650 Broadway in Georgetown (you probably have it already under the id 656048, it's the single-family one built in '72—big front yard, needs new gutters, but that's not the issue today). [truncated]

Annotated Output
[Email received: 2024-04-22]
Subject: Property claim from Jennifer Garcia — CLM-424063 (Shopping Center Incident)
Hi—this is Jennifer Garcia, and I'm writing because, honestly, I'm still kind of in shock about what happened this morning. They gave me this claim reference CLM-424063 when I called the help line earlier, but I figured I should also email so everything's in writing—it's just been a really stressful day, and I want to make sure nothing gets missed. Not sure how these things usually work, I guess you'll get back to me? Anyway...

I'm the owner on Policy POL-787532354 , with property coverage for my home at 6650 Broadway in Georgetown (you probably have it already under the id 656048, it's the single-family one built in '72—big front yard, needs new gutters, but that's not the issue today). [truncated]