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

Anonymous ACL submission

Abstract

When prompted to annotate text, Large Language Models (LLMs) may introduce subtle modifications that break the alignment between generated annotations and the original source. This makes standard generation unreliable for applications that require source-preserving structured outputs. We propose **sidetrack decoding**, a training-free constrained decoding algorithm for insertion-only annotation. The key idea is to frame annotation as insertion rather than regeneration: the decoder copies the source text by default and only "sidetracks" into generation mode when the model predicts markup should be inserted. Because source tokens are never regenerated, sidetrack decoding guarantees verbatim text preservation by construction. Across three structured extraction benchmarks, sidetrack decoding removes copy failures while maintaining competitive extraction accuracy on source-aligned tagging tasks. A speculative execution variant further improves efficiency, achieving 3.3× higher throughput than 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. In principle, 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 make subtle edits—correcting typos, normalizing formatting, or omitting words (Table 3). In our experiments, even the best regular span-generation model (Gemini 2.5 Pro) achieves only a 97.6% average perfect copy rate. These small changes break the alignment between tagged spans and the original source, undermining the attribution that span tagging is meant to provide. Simple fixes such as stronger copy instructions, rejection sampling, or format-constrained decoding can reduce failures but do not guarantee preservation, while token-classification systems require task-specific adaptation rather than directly using prompted generative models.

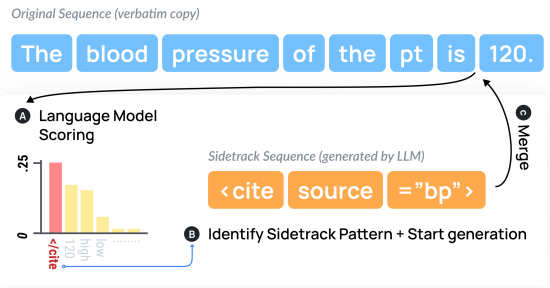

Figure 1: Illustration of the sidetrack decoding process.

We propose **sidetrack decoding**, a training-free constrained decoding algorithm for insertion-only annotation that guarantees source tokens are preserved verbatim while maintaining competitive extraction accuracy. The key observation is that document annotation can be framed as *text insertion*: rather than regenerating the entire document, the model only needs to decide *where* to insert markup tokens. Sidetrack decoding uses the model's next-token predictions to find these insertion points; when the model predicts a tag-opening token (e.g., <span) with high probability, it "sidetracks" into generation mode to produce the inserted markup, then returns to copying the original text. Because the original text is never regenerated, preservation

follows by construction. We evaluate sidetrack decoding on three structured extraction benchmarks and find that it removes copy failures in span generation while maintaining competitive extraction accuracy on source-aligned tagging tasks. We also implement a speculative execution variant that processes windows of source tokens in parallel, achieving 3.3× higher throughput than standard autoregressive decoding. Together, these results show that source preservation can be enforced at decoding time without giving up the flexibility of prompted language models.

2 Motivating Examples

Task formulation. We formalize document annotation as insertion over a fixed source token sequence. Given source tokens $w_1, w_2, w_3, \ldots, w_n$, the goal is to find positions at which to insert new token sequences. That is, for each insertion j, we predict an index k_j and an inserted sequence $z_1^{(j)}, z_2^{(j)}, \ldots, z_{m_j}^{(j)}$. The final text locally becomes

$$w_1, \ldots, w_{k_j}, z_1^{(j)}, z_2^{(j)}, \ldots, z_{m_j}^{(j)}, w_{k_j+1}, \ldots$$

For example, the j and $j+1$-th insertions can be `<span class="title">` and `</span>` that label a title span. One can leverage an autoregressive language model for this task with few-shot prompts. However, this is not efficient because models need to repeat the source tokens from scratch and, most importantly, *there is no guarantee that the existing text will be preserved*.

LLMs can subtly modify source text when prompted to annotate. In a simple prompting baseline, we ask the model to label entities in the input text with span tags. Despite being explicitly instructed to copy the input verbatim, the model may still alter the source text while generating the annotated output. We measure *robustness*: the fraction of outputs whose generated text exactly matches the input after stripping span tags. As shown in Table 1, no span-generation model achieves perfect preservation. Robustness varies widely across model families: even the best performer, Gemini 2.5 Pro, reaches only 97.6%, while Claude Opus 4.5 achieves just 66.3%. Table 3 shows a representative failure: the model correctly identifies the entity span but silently drops "VIN" from the source. In high-stakes domains—healthcare, legal, finance—such modifications can introduce unintended errors (e.g., altering a dosage or date), and

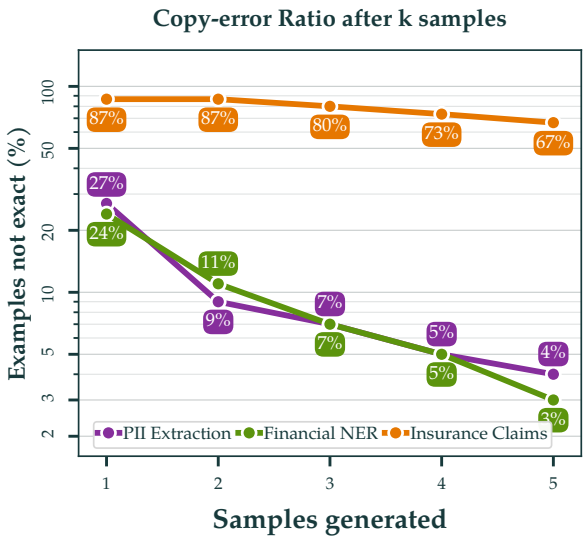

Figure 2: Rejection sampling reduces but does not eliminate copy errors. We sample Qwen3-8B up to k times and accept the first output whose non-markup text exactly matches the input after whitespace normalization. Even at $k = 5$, many examples remain non-exact, especially for the messier insurance-claims documents.

they break the alignment between extracted spans and source text that users rely on for verification.

Rejection sampling is insufficient. Since verbatim preservation is easy to check after generation—one can strip the markup tags and compare the remaining text against the input—a natural alternative is to generate multiple candidates and keep the first one that passes this check. We test this strategy on the three extraction settings used in our main experiments: financial named-entity recognition, PII extraction, and insurance-claims extraction. Rejection sampling improves robustness on cleaner inputs, but it does not solve the problem (Fig. 2). With Qwen3-8B, five attempts still leave non-exact outputs on PII and financial extraction, and the failure rate remains much larger on the messier insurance-claims documents, whose inputs often resemble emails or informal claim narratives.

Token classifiers preserve text but require task-specific adaptation. Span annotation is often solved with token classification, where a pretrained encoder such as BERT or RoBERTa (Devlin et al., 2019; Liu et al., 2019) is fine-tuned with a token-level prediction head on task-specific labeled spans. This approach is efficient and faithful to the input by construction, but its label space and decision boundary are fixed by supervised training data. Therefore, adapting to a new privacy schema or domain usually requires new annotations and fine-

tuning, whereas autoregressive language models can often be repurposed through few-shot prompting (Brown et al., 2020). As a diagnostic of this adaptation cost, we apply OpenAI Privacy Filter, a recent bidirectional token-classification model for PII masking (OpenAI, 2026), zero-shot to the i2b2/UTHealth clinical de-identification task (Stubbs and Uzuner, 2015; Stubbs et al., 2015). After mapping its labels into the shared-task schema, Privacy Filter reaches only 16.4% type-aware relaxed F1 and 24.0% type-agnostic relaxed detection F1 on the first 100 test examples. Under the same zero-shot setting, Qwen3-8B with sidetrack decoding reaches 45.4% and 49.6%; adding three in-context examples further improves performance to 55.5% and 59.3% (Table 4). This diagnostic is not a matched-capacity comparison or a critique of Privacy Filter in its intended setting; rather, it illustrates that token classifiers often require schema- and domain-specific adaptation, while prompted autoregressive models can be retargeted more directly.

Together, these examples suggest a gap between existing approaches: we need the adaptability of prompted autoregressive models while guaranteeing that the source text remains unaltered in the final output.

3 Sidetrack decoding

We propose **sidetrack decoding**, a constrained decoding method for insertion-only annotation. As shown in Fig. 1, the method keeps the source text on a main track that is copied verbatim, and only lets the LM sidetrack into generation mode when it proposes an insertion such as an opening or closing tag. The key idea is to use the model's next-token probabilities to locate insertion points: if the model assigns sufficient probability to a "tag-opening" token (defined later), we generate the markup, then return to the main track and continue copying the source text. This guarantees preservation because source tokens are never regenerated by the model. We also show that this procedure can be combined with speculative decoding by checking windows of source tokens in parallel, improving efficiency when insertions are sparse.

Formally, let $\mathcal{M}$ be an autoregressive LM and let S be the source token sequence. At source position t, we inspect the next-token distribution conditioned on the current annotated prefix, denoted $P_{\mathcal{M}}(\cdot \mid S_{<t})$ in the no-insertion case. We define $\textsc{TopLogProbs}(P; \mathcal{M}, k, \theta)$ to return up to k next-token candidates whose probabilities exceed threshold θ under prompt prefix P. These candidates indicate whether the model is likely to insert markup before continuing to copy the next source token. A sidetrack tracker T specifies which generated tokens start and stop an insertion, using regular expressions r_{start} and r_{stop}. A candidate token activates tracker T if it satisfies r_{start}. Because tokenizer boundaries may split a tag prefix across multiple tokens, a single candidate token may not be enough to determine whether a tracker should fire: for example, the model may need to generate "␣<" followed by "span" to match an opening-span pattern. The function $\textsc{Satisfy}(w; T, \mathcal{M})$ implements this lookahead search.

Sidetrack Decoding with Naive Search. Given an instruction context C and a source sequence S, the naive algorithm scans each source position for possible sidetracks. At position i, it queries TopLogProbs on the current prefix and uses Satisfy to determine whether any candidate activates a tracker. If so, the algorithm enters generation mode and generates until the tracker's stop pattern is reached; otherwise, it copies the next source token. To support consecutive insertions, such as generating immediately after , the algorithm rechecks the same source position after each generated sidetrack and advances i only when no sidetrack is activated.

Sidetrack Decoding with Speculative Execution. The naive scan requires a model query at every source position, even though insertion points are typically sparse. We therefore adapt the draft-and-verify idea from speculative decoding (Leviathan et al., 2023): the source text itself serves as the draft, and the LM verifies a window of copied source tokens in one pass. We define $\textsc{BlockTopLogProbs}(P, W; \mathcal{M}, k, \theta)$ for prompt prefix P and source-token window $W = [w_1, \ldots, w_b]$; it returns top-k candidates above threshold θ for each position by computing $P_{\mathcal{M}}(\cdot \mid P \oplus W_{<i})$. If no candidate activates a tracker within the window, the algorithm accepts the whole window as copied text. If a sidetrack is found at position j, it discards later window predictions, generates the sidetrack, and resumes scanning from that source position. The window size b controls the trade-off between parallel verification and wasted computation after an early sidetrack.

Table 1: Span-generation extraction performance with fixed Claims span evaluation. Highlighted rows use sidetrack decoding. FA = Field Accuracy, F1 = F1 Score, p90 d = average of dataset-level 90th-percentile edit distances, R = Robustness (Perfect Copy Ratio). All values in % except p90 d; higher is better except p90 d.

	Financial NER			**Claims**			**PII Extraction**			**Average**			
Model	FA	OA	R	FA	F1	R	FA	OA	R	FA	OA	p90 d	R
Commercial Models with Regular Decoding													
`Claude Haiku 4.5`	60.9	42.0	84.0	65.0	74.4	6.7	75.5	52.0	**98.0**	67.1	31.3	26.7	62.9
`Claude Sonnet 4.5`	62.6	43.0	80.0	65.9	75.6	10.0	**87.6**	**71.0**	34.0	72.0	38.0	43.3	41.3
`Claude Opus 4.5`	67.2	47.0	94.0	71.3	79.3	10.0	83.2	65.0	95.0	73.9	37.3	8.0	66.3
`Gemini 2.5 Flash Lite`	57.4	36.0	97.0	46.5	61.1	63.3	64.8	34.0	97.0	56.3	23.3	2.3	85.8
`Gemini 2.5 Flash`	65.1	43.0	99.0	77.8	80.2	80.0	77.2	53.0	96.0	73.4	32.0	1.3	91.7
`Gemini 2.5 Pro`	**70.6**	**54.0**	**100.0**	**78.7**	**81.6**	**96.7**	84.6	65.0	96.0	**78.0**	**39.7**	**0.0**	**97.6**
Open Models													
`Llama-3.1-8B`	14.5	7.0	20.0	**58.1**	66.7	3.3	52.0	21.0	38.0	41.5	9.3	246.6	20.4
`Llama-3.1-8B + Sidetrack`	40.4	26.0	**100.0**	42.9	56.3	**100.0**	54.0	21.0	99.0	45.8	15.7	**0.0**	99.7
`Qwen3-8B`	60.9	39.0	91.0	55.7	61.2	3.3	**67.1**	**39.0**	70.0	61.2	26.0	474.3	54.8
`Qwen3-8B + Sidetrack`	**65.1**	**46.0**	**100.0**	55.9	**68.4**	**100.0**	65.4	37.0	**100.0**	**62.1**	**27.7**	**0.0**	**100.0**

4 Experiments

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

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. The original Claims annotations often store normalized values rather than source-verbatim spans: in our audit, 267 of 506 gold fields (52.77%) are not substrings of the source document after basic normalization. For example, the source may contain "$5,834" while the gold field stores "5834", or mention "December 20th" while the gold field stores an ISO-formatted date. This creates a structural mismatch for source-preserving span annotation, where copying the correct source span can be penalized for not canonicalizing it. We therefore report a fixed source-verbatim Claims span evaluation for span-generation and sidetrack comparisons; Appendix E.1 describes the correction procedure. Full prompts and dataset examples are provided in Appendix E.

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 E).

Models and Implementation. We evaluate regular span generation across open-weight models (Qwen3-8B (Yang et al., 2025) and Llama-3.1-8B) and commercial model families (Claude Haiku/Sonnet/Opus 4.5 and Gemini 2.5 Flash Lite/Flash/Pro). We apply sidetrack decoding to the open-weight models, where next-token probabilities are available, and compare against JSON generation on Qwen3-8B and the commercial models. For JSON generation, we allow a maximum of 16,384 output tokens. For span generation, we set temperature to 0 for deterministic outputs; when a model exposes a reasoning-token budget, we allow up to 8,192 reasoning tokens before 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. For sidetrack decoding, we use the same deterministic generation setting and

Table 2: JSON-generation extraction performance and Qwen3-8B sidetrack comparison. FA = Field Accuracy, OA = Output Accuracy, F1 = F1 Score, d = Edit Distance, R = Robustness. All values in % except d. For JSON generation, R measures the proportion of extracted fields that appear as substrings in the source text; for sidetrack decoding, R measures exact source preservation after removing span tags.

	Financial NER			**Claims**			**PII Extraction**			**Average**			
Model	FA	OA	R	FA	F1	R	FA	OA	R	FA	OA	d	R
JSON Generation													
`Qwen3-8B`	65.1	48.0	99.0	79.4	80.5	6.7	67.1	34.0	97.0	70.6	32.9	—	67.6
`Claude Haiku 4.5`	68.5	49.0	91.0	82.8	81.3	6.7	77.5	51.0	**100.0**	76.3	36.7	—	65.9
`Claude Sonnet 4.5`	76.2	57.0	98.0	84.6	84.2	10.0	85.2	64.0	**100.0**	82.0	49.2	—	69.3
`Claude Opus 4.5`	**79.1**	**63.0**	**100.0**	**86.2**	**85.5**	3.3	84.6	**69.0**	**100.0**	**83.3**	**52.9**	—	67.8
`Gemini 2.5 Flash Lite`	70.2	50.0	99.0	85.2	83.6	10.0	60.1	28.0	**100.0**	71.8	32.7	—	**69.7**
`Gemini 2.5 Flash`	75.3	58.0	99.0	82.8	81.1	3.3	79.9	55.0	99.0	79.3	44.3	—	67.1
`Gemini 2.5 Pro`	74.9	59.0	**100.0**	85.4	84.4	6.7	79.5	57.0	94.0	79.9	48.7	—	66.9
Source-Preserving Span Generation													
`Qwen3-8B + Sidetrack`	65.1	46.0	100.0	55.9	68.4	100.0	65.4	37.0	100.0	62.1	27.7	0.0	100.0

the tracker parameters specified in Appendix D.

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 A 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 **Robustness** (R), the proportion of samples with exact text preservation. Sidetrack decoding guarantees $\mathrm{R} = 100\%$ by construction.

5 Results

Sidetrack decoding removes copy failures in span generation. Table 1 shows that the benefit is largest for open models whose regular decoding is brittle. For Llama-3.1-8B, sidetrack decoding eliminates most copy failures and improves average extraction quality, suggesting that part of the regular baseline's weakness comes from failing to produce a faithful annotated copy in the first place. Qwen3-8B shows a cleaner version of the same pattern: sidetrack decoding preserves or improves accuracy on source-aligned tasks while making source preservation exact. The robustness gain is especially consistent on messy Claims inputs, where ordinary decoding frequently rewrites the source but sidetrack decoding keeps the document fixed; the remaining errors are therefore span-selection errors rather than transcription errors.

JSON generation helps when the target requires normalization. Table 2 separates source-aligned extraction from normalized extraction. On Financial NER and PII extraction, Qwen3-8B with sidetrack decoding is close to direct JSON generation while preserving the source text exactly. Claims is different: JSON generation achieves much higher extraction accuracy because the benchmark often expects canonical values rather than copied spans. This is the mismatch described in Section E.1: more than half of the original Claims gold fields are not source substrings, so a source-preserving method can identify the right evidence while still disagreeing with the canonical label. The trade-off is that JSON outputs are not in-place annotations and provide no preservation guarantee; users must map generated fields back to document evidence before verifying them. Thus JSON generation is appropriate when canonicalization is the goal, while sidetrack decoding is preferable when extraction must remain auditable against the original text.

Robust generation is non-trivial. Standard span generation shows variable robustness: Claude Sonnet 4.5 achieves only 41.3% average robustness, Claude Haiku 4.5 achieves 62.9%, Gemini 2.5 Flash reaches 91.7%, and even the strongest model in this setting, Gemini 2.5 Pro, falls short of perfect preservation at 97.6% (Table 1). As Figure 3 shows,

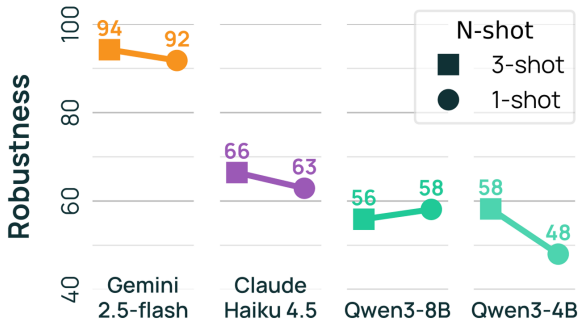

Figure 3: Generation robustness decreases with fewer in-context examples.

changing the number of in-context examples affects robustness, 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 4 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 $3.3\times$ improvement.

Case study: verbatim preservation trade-offs. Table 3 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).

6 Related Work

Text Editing and Insertion-Based Generation. Early copy-based generation methods augment sequence-to-sequence models with mechanisms for reusing source tokens. CopyNet (Gu et al., 2016) and pointer-generator networks (See et al., 2017), for example, learn when to generate from the vocabulary and when to copy from the input. These mechanisms are useful when source and target text overlap, but copying remains a model decision and does not provide a hard guarantee that the source will be preserved. Later work makes the edit structure more explicit: the Insertion Transformer (Stern et al., 2019) generates sequences by inserting tokens at arbitrary positions, and the Levenshtein Transformer (Gu et al., 2019) models generation through insertions and deletions. Task-oriented editing systems such as LaserTagger (Malmi et al., 2019), GECToR (Omelianchuk et al., 2020), and Seq2Edits (Stahlberg and Kumar, 2020) similarly exploit high input-output overlap through learned edit operations; Malmi et al. (2022) survey this broader landscape. Sidetrack decoding is inspired by this insertion view, but differs in mechanism and guarantee: it uses an off-the-shelf autoregressive LM to identify insertion points, while the algorithm itself copies the original text and therefore preserves it by construction.

Speculative Decoding and Input Reuse. The efficiency component of sidetrack decoding builds on speculative decoding (Leviathan et al., 2023), which accelerates autoregressive inference by using a draft model to propose tokens that a target model verifies in parallel. In sidetrack decoding, the draft is not a separate model: it is the original input text itself. This connects to recent input-reuse variants, including LLMA (Yang et al., 2023), which uses reference documents as draft text, and Prompt Lookup Decoding (Saxena, 2023), which proposes draft spans through n-gram matches from the prompt. These methods primarily use input reuse to improve speed. Sidetrack decoding uses the same property for both speed and robustness: accepting a draft token means copying the original token verbatim, while rejection triggers only the generation of inserted markup. Thus the speculative variant improves throughput without changing the preservation guarantee.

Attribution in LLM Generation. Our work also 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 (cit-

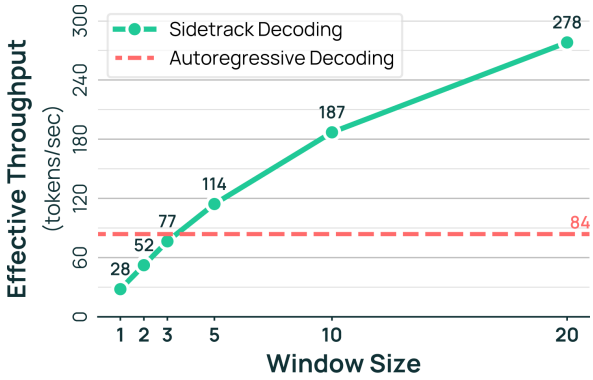

Figure 4: Effective throughput of sidetrack decoding on Qwen3-8B as window size increases.

Table 3: 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 .

ing evidence that supports a claim) from *contributive* attribution (identifying training data that influenced generation). Symbolically grounded generation (Hennigen et al., 2024) shares a similar motivation: making LLM outputs easier to manually verify in high-stakes settings by linking generated spans to fields in structured source data. Once the model produces a valid symbolic reference, a renderer can faithfully substitute the corresponding source value, but the surrounding prose remains freely generated. Sidetrack decoding addresses the complementary setting where the source is an existing text document: the markup is freely generated, but all original document text outside the inserted markup is preserved verbatim—what might be called *source-preserving* attribution.

7 Conclusion

We present sidetrack decoding, a training-free decoding algorithm for annotation tasks where the source text must remain unchanged. By reframing annotation as insertion rather than regeneration, sidetrack decoding copies the source on the main track and only invokes the language model to insert markup tokens. This eliminates copy failures by construction while maintaining competitive extraction accuracy on source-aligned span-tagging tasks. A speculative execution variant further improves efficiency, achieving 3.3× higher throughput than standard autoregressive decoding. The method is therefore best suited for applications that require verifiable, source-preserving outputs.

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.

Ethical Considerations

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

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.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind

Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

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.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1631–1640.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, volume 32.

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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Eric Malmi, Yue Dong, Jonathan Mallinson, Aleksandr Chuklin, Jakub Adamek, Daniil Mirylenka, Felix Stahlberg, Sebastian Krause, Shankar Kumar, and Aliaksei Severyn. 2022. Text generation with text-editing models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics*.

Eric Malmi, Sebastian Krause, Sascha Roesner, Bhaskar Uber, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 5054–5065.

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.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170.

OpenAI. 2026. Model card for OpenAI privacy filter. https://cdn.openai.com/pdf/c66281ed-b638-456a-8ce1-97e9f5264a90/OpenAI-Privacy-Filter-Model-Card.pdf. Accessed: 2026-05-26.

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.

Apoorv Saxena. 2023. Prompt lookup decoding. https://github.com/apoorvumang/prompt-lookup-decoding. Accessed: May 2026.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1073–1083.

Felix Stahlberg and Shankar Kumar. 2020. Seq2edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 5147–5159.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

Amber Stubbs, Christopher Kotfila, and Özlem Uzuner. 2015. Automated systems for the de-identification of longitudinal clinical narratives: Overview of 2014 i2b2/UTHealth shared task track 1. *Journal of Biomedical Informatics*, 58:S11–S19.

Amber Stubbs and Özlem Uzuner. 2015. Annotating longitudinal clinical narratives for de-identification: The 2014 i2b2/UTHealth corpus. *Journal of Biomedical Informatics*, 58:S20–S29.

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*.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Inference with reference: Lossless acceleration of large language models. *Preprint*, arXiv:2304.04487.

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 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 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 3: the span includes "VIN" alongside the actual vehicle identification number). We therefore report partial-match metrics as the primary results (FA, OA, F1).

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.

B 2014 i2b2/UTHealth Clinical De-identification Setup

We use the 2014 i2b2/UTHealth clinical de-identification corpus (Stubbs and Uzuner, 2015; Stubbs et al., 2015): the test split contains 514 clinical notes with character-offset PHI annotations, and the diagnostic comparison in the main text uses the first 100 test notes. We use Qwen3-8B with a dataset-specific span-annotation prompt and report both zero-shot and three-shot prompting. The prompt asks the model to copy the clinical note exactly while inserting HTML tags of the form `<span class="DATE"> ... </span>` for the shared-task PHI label set. Similar to the main experiments, we use temperature 0 and up to 8,192 reasoning tokens before producing the annotated output with thinking enabled.

We evaluate the task with a character-offset span evaluator. For generated span-annotated text, we first strip span tags to compute copy fidelity against the original clinical note. We then align predicted spans back to the original note by scanning the annotated output while skipping tag characters, yielding predicted start and end offsets. Type-aware metrics count a true positive only when both the PHI type and span boundary match a gold span; detection metrics ignore PHI type and match only the span boundary. Relaxed matching allows both predicted start and predicted end to be within two characters of the gold offsets.

For the token-classification baseline, we run the local OpenAI Privacy Filter model (OpenAI, 2026) with the Hugging Face token-classification pipeline and simple span aggregation. This baseline is zero-shot with respect to the clinical de-identification task: we apply the released privacy label space directly to the clinical notes, then map its predicted labels to shared-task PHI types before evaluation. BIOES prefixes are stripped before applying the mapping in Table 5; spans without valid character offsets are discarded.

Method	Shot	F1	DetectF1
OpenAI Privacy Filter	0	16.4	24.0
Qwen3-8B + Sidetrack	0	45.4	49.6
Qwen3-8B + Sidetrack	3	55.5	59.3

Table 4: Diagnostic results on the first 100 test notes from the 2014 i2b2/UTHealth clinical de-identification task. Relaxed F1 is type-aware; DetectF1 ignores PHI type and evaluates only span detection. All values are percentages.

Privacy Filter label	Mapped PHI type
`account_number`	`IDNUM`
`private_address`	`STREET`
`private_date`	`DATE`
`private_email`	`EMAIL`
`private_person`	`PATIENT`
`private_phone`	`PHONE`
`private_url`	`USERNAME`
`secret`	`IDNUM`

Table 5: Label mapping used to evaluate OpenAI Privacy Filter predictions on the 2014 i2b2/UTHealth clinical de-identification task.

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.

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$(C \oplus S'[:\text{pos}] \oplus \text{candidate}; \mathcal{M}, \text{stop} = \text{z.stop})$
    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'

D Sidetrack Tracker Settings

For all extraction-accuracy experiments, we use top-k value $k = 1$ and probability threshold $\theta = 0$. Thus, TopLogProbs returns the highest-probability next-token candidate without probability filtering; the effective filter is the tracker regex match. The tracker patterns are defined over decoded candidate text

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 b

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

```
Initialize: S' ← S, offset ← 0, position i ← 0
while i ≤ |S| do
    pos ← i + offset                                   ▷ Actual position in modified sequence
    window_end ← min(i + b, |S|)                       ▷ Define window of size b
    window_tokens ← S[i : window_end]
    candidates_block ← BlockTopLogProbs(C ⊕ S'_:pos, window_tokens; M, k, θ)
                                                       ▷ Returns list of top-k candidates for each position in window
    sidetrack_found ← False
    for j ← 0 to |window_tokens| − 1 do
        candidates ← candidates_block[j]               ▷ Candidates at position j in window
        tracker ← None
        for each tracker T_m ∈ T do
            for each candidate ∈ candidates do
                if Satisfy(candidate, T_m) then
                    tracker ← T_m
                    break
        if tracker ≠ None then                         ▷ Sidetrack found at position j in window
            generated ← Generate(C ⊕ S'[: pos + j] ⊕ candidate; M, stop = tracker.stop)
            sidetrack_tokens ← candidate ⊕ generated
            S' ← S'[: pos + j] ⊕ sidetrack_tokens ⊕ S'[pos + j :]
            offset ← offset + |sidetrack_tokens|
            i ← i + j                                  ▷ Advance to sidetrack position
            sidetrack_found ← True
            break                                      ▷ Restart from sidetrack position
    if ¬sidetrack_found then
        i ← i + |window_tokens|                        ▷ No sidetrack, advance by window size
return S'
```

after stripping leading tokenizer whitespace, and SATISFY performs lookahead generation when a tag prefix is split across multiple tokens.

Tracker	Start regex	Stop regex
Opening span tag	^<span	">
Closing span tag	^	None

Table 6: Sidetrack tracker specification used for the span-tagging experiments. The opening tracker generates the full opening tag, including the task-specific class or field label, then returns to copying source text. The closing tracker inserts the fixed closing tag and returns immediately.

The class labels inside opening tags are task-specific: FIRE uses entity labels such as `Company` and `Money`; PII uses labels such as `EMAIL` and `VEHICLEVIN`; and Claims uses hierarchical field labels such as `header.claim_id` and `policy_details.policy_number`. The tracker logic is otherwise unchanged across tasks.

E 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.

E.1 Fixed Source-Verbatim Claims Evaluation

The Insurance Claims benchmark stores many gold values in canonical form rather than as spans copied from the source document. We audited the 30-example Claims evaluation set and found that 267 of 506 gold fields (52.77%) are not substrings of the corresponding source document after basic whitespace and punctuation normalization. Common cases include numeric normalization (e.g., source "$5,834" vs. gold "5834"), date normalization (e.g., source "December 20th" vs. gold ISO-style dates), and categorical or capitalization normalization (e.g., source phrasing mapped to canonical channel or coverage labels).

This annotation choice is appropriate for JSON-style extraction, where models may normalize values, but it is not appropriate for evaluating source-preserving span annotation. A sidetrack output can locate the correct source evidence and still be marked wrong if the gold field expects a normalized value that never appears in the document. For the span-generation and sidetrack rows in Table 1, we therefore use a fixed source-verbatim Claims evaluation: affected gold fields are mapped back to the supporting surface form in the source document when the evidence span is unambiguous. The model outputs are unchanged; only the evaluation targets for source-preserving span comparison are corrected.

E.2 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

- 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

```
<span class="Company"> Albertsons </span> and <span class="Company"> Rite Aid </span> called off
the planned merger after several advisory firms recommended voting against the proposal .
```

E.3 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, 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.

E.4 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]