Strategies for Span Labeling with Large Language Models

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly used for text analysis tasks, such as named entity recognition or error detection. Unlike encoder-based models, however, generative architectures lack an explicit mechanism to refer to specific parts of their input. This leads to a variety of ad-hoc prompting strategies for span labeling, often with inconsistent results. In this paper, we categorize these strategies into three families: *tagging* the input text, *indexing* numerical positions of spans, and *matching* span content. To address the limitations of content matching, we introduce LogitMatch, a new constrained decoding method that forces the model's output to align with valid input spans. We evaluate all methods across four diverse tasks. We find that while tagging remains a robust baseline, LogitMatch improves upon competitive matching-based methods by eliminating span matching issues and outperforms other strategies in certain conditions.[1]

1 Introduction

Generative large language models (LLMs) are mostly based on the Transformer decoder architecture (Vaswani et al., 2017; Radford et al., 2019), which makes them well-equipped for open-ended generative tasks. However, LLMs are increasingly being used also as a tool for automatic text analysis: from finding factual inconsistencies or grammatical errors to extracting structured information about events and entities. In these tasks, an issue often arises: **how can LLMs explicitly refer to parts of an input text?**

While structured outputs – i.e. the task of constraining the model output format – are being actively investigated and increasingly supported by LLM frameworks (Liu et al., 2024; Willard and Louf, 2023; guidance-ai, 2025), the issue of referring to the given text is rather overlooked. Unlike encoder-based models (Devlin et al., 2019; Warner et al., 2025) that can attach a label to each input token, generative LLMs, by contrast, have no explicit mechanism to refer to their input. This issue becomes apparent in span labeling tasks, where we need to identify and categorize parts of the given text (see Figure 1 for an example). In these cases, we find ourselves needing to *refer to spans in the given input text by generating another text*, which offers no guarantees on alignment between the two texts.

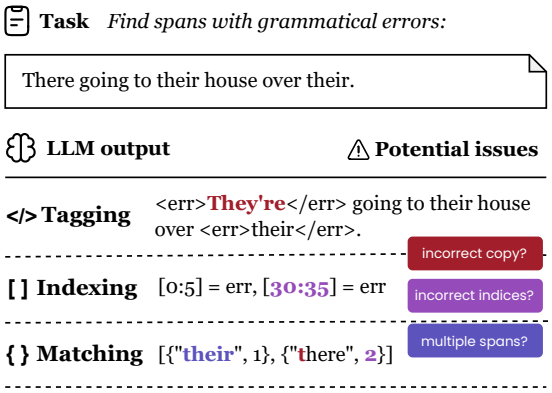

Figure 1: Illustration of main approaches to span labeling with LLMs, along with the associated issues. The issues stem from the fact that LLMs have no explicit mechanism to ground their outputs in the input text.

Why use LLMs for span labeling? First, LLMs can achieve similar performance on span labeling tasks as finetuned encoder models without costly task-specific training (Wang et al., 2025a; Kocmi et al., 2024). Moreover, thanks to their in-context learning abilities, LLMs can deal with low-resource tasks for which encoder models are not readily available, such as evaluating factual accuracy of sports reports (Thomson et al., 2023), identifying rhetorical structures (Mulcaire and Madnani, 2025), or detecting fallacies and propaganda techniques

[1] Our code is available at https://anonymous.4open.science/r/llms-7677.

(Hasanain et al., 2024; Ramponi et al., 2025).

In this paper, we set out to explore how to robustly apply LLMs to span labeling tasks. We first show that there is currently no consensus on how to formulate span labeling tasks for LLMs. This leads to a wide range of ad hoc strategies, often found to deliver unsatisfactory performance on downstream tasks. In Section 3, we cluster these strategies into three groups: tagging the copy of the input text (*tagging*), string matching of the span content (*matching*), and referring to the numerical indices of the input tokens (*indexing*). On top of the base methods for each strategy, we also present tweaks that were introduced to address their drawbacks.

In Section 4, we introduce our novel constrained decoding extension for the *matching* strategies. Our method – LOGITMATCH – provides guarantees on the output format by modifying raw model logits to only allow generating valid spans from the input text. By affecting only the decoding process, our method is applicable to any locally-deployed LLM without costly model finetuning or architecture modifications.

We evaluate a representative set of span labeling methods across four diverse tasks: named entity recognition, grammatical error correction, machine translation error detection, and a synthetic pattern lookup task (see Section 5). In Section 6, we discuss our results, showing that while *tagging* strategies offer a robust baseline, LOGITMATCH successfully addresses the limitations of standard *matching* strategies and offers a highly competitive approach. We also analyze the trade-offs of each method and provide recommendations regarding their applications to specific tasks.

2 Task Definition

In this paper, our goal is to (1) identify contiguous text sequences satisfying given criteria and (2) assign each sequence to one of predefined categories. Both of these subtasks need to be accomplished by generating textual output.

Formally, we define the problem as follows: Given an input text $X = [x_1, x_2, \ldots, x_n]$ consisting of n characters, the objective is to identify and label a set of spans $S = \{(s_i, e_i, c_i)\}_{i=1}^{m}$ in X satisfying given criteria. Each span is characterized by three components: a start index s_i, an end index e_i (exclusive), and a category c_i from the set of span categories C defined for the task. The start and end indices satisfy $1 \leq s_i \leq e_i \leq n$, ensuring that each span corresponds to a contiguous subsequence of tokens in X. We are using a model $\mathcal{M}$ that generates an output string $Y = [y_1, y_2, \ldots, y_{|Y|}]$, out of which S needs to be extracted.

3 Existing Strategies for Span Labeling with Generative Models

We identified three groups of strategies in the literature that are used to formulate span labeling for generative models (see Table 1 for an overview). We categorize these strategies depending on how the spans are marked: *tagging* (Section 3.1), *indexing* (Section 3.2), and *matching* (Section 3.3).

3.1 *Tagging*: Marking Spans in the Input Text

The *tagging* strategies explicitly generate an output for each input token, requiring $O(n)$ output tokens per example. In the more common variant, the model generates a full copy of the input text and surrounds the spans of interest with special tags. Alternatively, the model can generate a sequence of tags, one tag per input token, and include *null* tags for unmarked tokens.

Custom Tags One of the first applications of LLMs for span labeling is the work of Wang et al. (2025b), who apply an LLM to identifying named entities in the text. They prompt the model to surround the entities with special markers `@@` and `##` to denote the tag boundaries. Yan et al. (2024) extend their approach by instructing the model to append the entity category after the closing tag (such as `##LOC`), adding the possibility to classify the spans.

XML-like Tags To detect translation errors, Treviso et al. (2024) instruct an LLM to surround errors with XML-like tags such as `<error1 severity="major">text</error1>`. A similar approach is adopted by Mulcaire and Madnani (2025) to detect shell language (content-free fillers) in English language exams.

BIO Tags The BIO scheme (Ramshaw and Marcus, 1995) classifies each token as a beginning of a span (`B`), inside a span (`I`), or outside a span (`O`). Obeidat et al. (2025) use the BIO scheme for biomedical NER by making the LLM generate a pair `word:tag` for each word in the text. The BIO tagging scheme was also initially tested by Ramponi et al. (2025) to detect errors in social media texts, but it was discarded due to unsatisfactory results. An alternative approach proposed by Wang

Target output: Turing PER was born in London LOC.

Str.	Method	Output	Used in
tagging	Custom tags	`@@Turing## was born in @@London##.`	Wang et al. (2025b)
		`@@Turing##PER was born in @@London##LOC.`	Yan et al. (2024)
	XML tags	`<entity type="PER">Turing</entity> was born in <entity type="LOC">London</entity>.`	Treviso et al. (2024), Mulcaire and Madnani (2025)
	BIO tags	`(Turing, B-PER) (was, O) (born, O) (in, O) (London, B-LOC)`	Obeidat et al. (2025), Ramponi et al. (2025)
		`B-PER O O O B-LOC`	Wang et al. (2025b)
indexing	Char-level	`[1:7] = PER` `[20:26] = LOC`	Hasanain et al. (2024)

	Word+num	*Input: [1]Turing [2]was [3]born [4]in [5]London*
		`[1:1] = PER`
		`[5:5] = LOC`
			Ramponi et al. (2025)

matching	Bullet points	`- PER - Turing` `- LOC - London`	Kocmi and Federmann (2023)
	JSON output	`[{"type": "PER", "text": "Turing"}, {"type": "LOC", "text": "London"}]`	Kasner and Dušek (2024), Kasner et al. (2025)
		`[{"type": "PER", "text": "Turing", "index": 0}, {"type": "LOC", "text": "London", "index": 0}]`	Klesnilová (2025), Hasanain et al. (2024)

Table 1: Overview of span labeling strategies with generative models used in related work. As an example, we present the methods applied on the NER task for the sentence "*Turing was born in London.*" with two span categories: PER (person) and LOC (location).

et al. (2025b) also relies on BIO tags but only generates the sequence of tags without generating the words themselves. However, this underperforms their aforementioned custom tag approach.

3.2 *Indexing*: Referring to Character Indices

The *indexing* strategies focus on the spans to be labeled, which requires $O(m)$ tokens per example. The model is explicitly asked for the numerical start and end indices (s_i, e_i) of each span.

Character-level For propaganda span detection, Hasanain et al. (2024) ask for the start and end character indices of the span. They report that the method led to inaccuracies since the model did not have explicit access to character indices. The authors eventually resorted to string matching of the span content (cf. Section 3.3).

Word-Level+Numbering For the task of detecting argumentation falacies, Ramponi et al. (2025) provide an LLM with explicit access to word indices by appending an index to each word in the input text. The output they require is in the format `[start_word-end_word = category]`. This approach was found to perform better than using BIO tags (cf. Section 3.1).

3.3 *Matching*: String Matching of the Span Content

The *matching* strategies also focus on the content of the spans to be labeled and thus require generating at least $O(m)$ tokens per example.[2] Unlike in *indexing* strategies, the model generates the textual content of the spans. The start and end indices (s_i, e_i) are determined post-hoc by string matching.

Unstructured List A basic version of this approach is adopted by Kocmi and Federmann (2023), who detect translation errors by asking the model to generate a bulleted list with items `{error_category} - {span_text}`.

JSON Output Kasner and Dušek (2024) use a JSON-like format for evaluating output accuracy in data-to-text generation. The approach is also applied by Kasner et al. (2025) to identify errors in machine translation and detect propaganda techniques. Similarly, Hasanain et al. (2024) and Klesnilová (2025) use the JSON format to detect propaganda techniques. To disambiguate spans with the same content, Klesnilová proposes adding

[2] Since the content of the span gets generated, the upper bound is $O(n)$. However, the spans tend to be much shorter than the input text in practice.

a JSON field `occurence_index` with the sequential number of the span.

3.4 Issues with Existing Methods

We now turn to the drawbacks of the strategies. Some of the drawbacks are inherent, while others are related to specific limitations of LLMs.

LLMs cannot copy reliably. Both the *tagging* and *matching* strategies rely on the model's ability to copy its input. For *tagging*, the model must reproduce the entire input text exactly to preserve alignment. For *matching*, the generated span must match the input substring character-for-character. In practice, however, LLMs often "fix" the text by correcting typos, changing capitalization, or correcting factually incorrect sentences (Liu et al., 2024; Li et al., 2024; Jarolím et al., 2025). It is possible to partially mitigate this issue with fuzzy matching during post-processing, but that may introduce a new source of errors and does not solve the fundamental lack of grounding in the input text.

LLMs cannot count reliably. Although positional embeddings allow LLMs to operate with the relative position of input tokens, the embeddings do not provide the model a straightforward access to the explicit token indices. The knowledge of number of characters in each token is learned completely indirectly (Edman et al., 2024; Fu et al., 2024). Therefore, when *indexing* strategies attempt to bypass text generation by predicting numerical positions directly, the indices are only "informed guesses". One can mitigate this by inserting indices into the text explicitly (Ramponi et al., 2025). However, that disrupts the natural flow of text and potentially degrades model performance.

Span content only is ambiguous. For *matching* strategies, the lack of grounding in the input text can also introduce ambiguities. If the model correctly identifies a span (e.g., "the"), but that exact span appears many times across the document, string matching alone cannot determine which specific occurrence the model intended to label. Disambiguating by asking the model to generate an occurrence index (where the model can suggest that it refers to the *second* instance of word "the"; Klesnilová, 2025) once again puts the burden on the model's counting abilities.

4 LogitMatch: Making *matching* Methods More Robust

We introduce a new method – LogitMatch – that addresses some of the weaknesses of the *matching* strategies outlined in Section 3.3. To determine the start and end indices of the span, the textual content of the span generated by the model needs to match a specific span of the input text. The main issue is that the model may not perfectly reproduce the input span. Our method ensures that each decoded span is a valid input span while making only minimal interventions to the model's decoding process.

4.1 Preliminaries

LogitMatch is based on decoding a list of spans in a JSON format (cf. Section 3.3). We ask the model to provide a list of JSON objects, where each object contains the field `text` with the span content.[3] The main idea of LogitMatch is to **limit the set of permissible tokens while the `text` field is generated** to ensure that the decoded tokens are a continuous span from the input text.[4]

4.2 Algorithm

Our algorithm (see Appendix B.2 for code) uses three modes: `DEFAULT`, `SELECT`, and `COPY`.

Default Mode Our algorithm starts in `DEFAULT` mode, where uses standard autoregressive decoding, sampling the next token from the model distribution: $y_t \sim P_{\mathcal{M}}(y_t \mid y_{<t}, X)$.

Select Mode The `SELECT` mode is activated once the model starts decoding the value of the `text` field containing the span content. In `SELECT` mode, we limit the model's vocabulary to the tokens from the input: $[x_1, x_2, \ldots, x_n]$ and let the model generate a *single token* x_i starting a span.

Copy Mode After the token x_i is generated, we switch to the `COPY` mode, where the model can generate the following input token x_{i+k} (where initially $k = 1$) or a closing quote (`"`). If the model generates the token x_{i+k}, we increment k and continue in the `COPY` mode. In the latter case, we exit the `COPY` mode and continue in `DEFAULT` model.

[3] The objects can optionally contain other fields, e.g. for span categorization.

[4] The field naming we selected is arbitrary and does not affect the generality of our method.

4.3 Tokenization Issues

Although our algorithm is conceptually simple, we found that input tokenization introduces additional issues that need to be handle with care so that we do not accidentally disallow valid tokens.

Different input and output tokenization. We need to account for the fact that tokenization of the output text does *not* necessarily correspond to the input text tokenization. Consider, for example, the input text "`Hello.`". On the input, the text may be tokenized as `["Hello", "."]`. However, the model might first decode the token "`Hel`", which guides it towards the tokens "`lo`" and "`.`". Therefore, in the `COPY` mode, we need to allow not only the following input token, but all the tokens that are valid prefixes of the subsequent text.

Handling the field boundaries. The value of the JSON field is a string enclosed in quotes. However, the quotes may be decoded not only as standalone tokens, but also as parts of a token which includes content inside or outside of the field. When entering the `SELECT` mode (i.e., in the step where the expression "`text"\s:\s"` would be decoded), we need to consider whether the token containing the opening quote also has a suffix (e.g. `_"London`) and handle the suffix (in this case `London`) in the context of the `SELECT` mode. Accordingly, in the `COPY` mode, we need to whitelist all the tokens that satisfy the following: they start with the valid prefix, i.e., continuation of the decoded text, followed by a quotation mark, and optionally followed by a suffix (e.g., a comma followed by a newline). All these tokens need to be considered a signal for exiting the `COPY` mode.

5 Experiments

5.1 Methods

Based on our survey in Section 3 (see also Table 1), we implement and evaluate what we consider a representative set of methods for each strategy:

- TAG: *tagging* with XML-like tags.
- INDEX: *indexing* with character-level indices.
- INDEX-ENRICHED: *indexing* with character-level indices, with the character indices inserted before each word on the input.
- MATCH: *matching* with JSON output.
- MATCH-OCC: *matching* with JSON output and occurrence index.

Task	# Ex.	# Lang.	# Cat.	Word tok.
NER	7,523	13	3	✓
GEC	504	1	3	✓
ESA-MT	867	9	2	✗
CPL	1,000	1	1	✗

Table 2: Statistics of the datasets used in our experiments. The table shows the number of examples, languages, and categories used for each task. The inputs for NER and GEC use NLTK-style word tokenization.

We also implement our novel method from Section 4 analogically to the *matching* methods:

- LOGITMATCH: *matching* with JSON output and constrained decoding.
- LOGITMATCH-OCC: *matching* with JSON output, constrained decoding, and occurrence index.

For JSON-based outputs, we additionally test a structured output variant (denoted with -S) in which we enforce the JSON schema.[5] In this variant, the model output is guaranteed to contain all the required fields and a valid category label.

We provide more details on individual methods, including model prompts and details on output parsing, in Appendix B.

5.2 Datasets

For our experiments, we select four span labeling tasks. First, we use two well-established high-resource NLP tasks: named entity recognition and grammatical error correction. We also adopt span-level detection of errors in machine translation as a low-resource span labeling task. Finally, we create a synthetic task called *conditional pattern lookup* to stress test robustness of all approaches.[6] See Table 2 for an overview of our datasets.

Named Entity Recognition (NER) The goal in the NER task is to localize named entities and classify their type. We use the UniversalNER benchmark (Mayhew et al., 2024) that covers three coarse-grained entity types: Person (`PER`), Organization (`ORG`), and Location (`LOC`). The dataset contains 7,523 examples across thirteen languages.

Grammatical Error Correction (GEC) In the GEC task, the goal is to identify spans requiring

[5] For LOGITMATCH, enforcing the JSON schema happens *alongside* our custom constrained decoding. See Appendix B.2 for details.

[6] Note that the inputs for NER and GEC use NLTK-style word tokenization (Bird et al., 2009) for separating punctuation from words, which we keep in its original form.

edits in order to make the text grammatically correct. We adapt the English Write & Improve corpus from the MultiGEC (Masciolini et al., 2025) dataset that contains 504 examples with a list of edits between the original and corrected sentences from the ERRANT tool (Bryant et al., 2017). We use the categorization of grammatical errors into three high-level categories: replacement (`R`), missing (`M`), and unnecessary (`U`).[7]

Detecting Errors in Machine Translation (ESA-MT) In the ESA-MT task, the goal is to find error spans in translated texts given their source. We use data from the error span annotation (ESA) task at WMT24 (Kocmi et al., 2024) that contains translation of texts from the news, social and literary domains from English into nine languages. Specifically, we select the examples containing errors from a test set of Kasner et al. (2025), resulting in 867 examples. Following the annotation scheme in the dataset, we classify the errors as `MINOR` or `MAJOR`.

Conditional Pattern Lookup (CPL) In real-world tasks, multiple occurrences of the same span are relatively rare. To test how the approaches behave with multiple occurrences of a specific span, we devise a synthetic task called *conditional pattern lookup*. The goal of the task is to find occurrences of a given regular expression in the input sequence. We use constraints `not_preceded_by`, `not_followed_by`, `preceded_by`, and `followed_by` to make the lookup conditional (i.e., only some occurrences of a specific pattern should be identified). For example, *Find all sequences matching* '`\w+ dry`' *that are not preceded by* '`house`'.[8] As an input, we generate 1000 random sequences of approximately 100 English words. We ensure that the pattern occurs at least once in the input sequence.

5.3 Models

We test all approaches with state-of-the-art open LLMs that we selected to cover multiple providers and model sizes: Qwen3-8B (Yang et al., 2025), Mistral-Small-24B-Instruct (Mistral AI Team, 2025), and Llama-3.3-70B-Instruct (Grattafiori et al., 2024). We also evaluate the proprietary GPT-5-mini model on the methods that do not require access to logits (i.e., excluding LOGITMATCH). By default, we disable reasoning settings of the models. To test how reasoning may help, we performed additional experiments with Qwen3-8B with the reasoning effort set to `high`, that we denote as Qwen3-8B-Think. The details can be found in the Appendix A.

5.4 Evaluation Metric

Our evaluation metric is the F1-score over annotated spans, adjusted for span overlaps (Da San Martino et al., 2019; Kasner et al., 2025). This metric extends the standard F1-score by counting matches proportionally based on their character-level overlap. Formally, we denote the set of gold and hypothesis spans as $s \in S$ and $\hat{s} \in \hat{S}$, respectively. We use $s_i \cap s_j$ to denote character-level overlap of spans and $|s|$ to denote the span length in characters. The F1-score is calculated as the harmonic mean of precision (P) and recall (R): $F_1 = (2 \cdot P \cdot R)/(P + R)$, where:

- **Precision** is the average overlap ratio for each predicted span:

$$P(S, \hat{S}) = \frac{1}{|\hat{S}|} \sum_{\hat{s} \in \hat{S}} \frac{|s \cap \hat{s}|}{|\hat{s}|} \quad (1)$$

- **Recall** is the average coverage of gold spans by predictions:

$$R(S, \hat{S}) = \frac{1}{|S|} \sum_{s \in S} \frac{|s \cap \hat{s}|}{|s|} \quad (2)$$

We evaluate both *hard* and *soft* variants of these metrics. The hard variant requires exact category matching for overlaps to contribute to the score, while the soft variant ignores category labels and only considers positional overlap.

6 Results

We primarily base our observations on the overlap-adjusted *hard* F1-score in Table 3. To provide more insights into the failures of individual strategies, we also compute the following error rates (see Figure 2):

- **Parsing errors**: The model output could not be parsed (e.g., invalid JSON).
- **Span content errors**: The span content generated by the model could not be matched to any substring in the input text.

[7] For more details on how we adapted GEC for span labeling, see Appendix D.

[8] Note that the task can be easily solved by any regular expression matcher. We apply LLMs to it only for benchmarking purposes.

Method	NER 8B	NER 24B	NER 70B	NER 5-m	GEC 8B	GEC 24B	GEC 70B	GEC 5-m	ESA-MT 8B	ESA-MT 24B	ESA-MT 70B	ESA-MT 5-m	CPL 8B	CPL 24B	CPL 70B	CPL 5-m
TAG	63.4	63.3	70.4	**75.5**	**24.0**	**31.4**	**30.4**	**36.5**	11.0	11.0	**11.6**	12.8	52.9	71.5	**82.2**	**80.6**
INDEX	17.8	16.7	23.3	33.8	9.5	7.5	5.2	6.8	7.8	8.2	7.7	7.0	14.3	17.3	14.9	19.9
INDEX-ENRICHED	39.6	61.6	59.3	69.1	10.0	12.0	12.9	23.2	9.0	9.5	8.7	7.8	49.1	**78.1**	68.1	79.9
MATCH	70.9	53.5	**72.7**	59.0	15.2	19.3	22.1	24.3	13.6	12.3	10.0	**13.7**	30.6	36.9	39.8	44.1
MATCH-S	71.0	49.1	72.4	71.3	16.9	21.2	22.6	24.2	12.9	**13.3**	10.7	12.9	30.4	37.0	40.1	42.3
MATCH-OCC	71.1	63.0	68.2	66.7	16.6	16.9	23.8	27.6	13.5	12.9	11.2	12.4	**73.4**	64.1	74.9	77.4
MATCH-OCC-S	71.5	53.6	67.3	65.0	16.4	19.0	24.6	26.4	13.4	12.9	11.0	12.5	73.2	66.6	75.1	79.9
LOGITMATCH	71.4	53.7	72.4	–	15.8	22.0	22.6	–	13.0	12.3	10.4	–	31.0	36.9	40.0	–
LOGITMATCH-S	**71.7**	48.6	72.3	–	15.8	22.4	23.1	–	13.4	12.5	11.2	–	31.0	36.8	40.3	–
LOGITMATCH-OCC	71.5	**63.4**	68.3	–	16.9	19.7	24.1	–	13.9	12.7	11.4	–	73.2	63.6	75.0	–
LOGITMATCH-OCC-S	71.3	53.0	68.3	–	17.9	19.6	24.3	–	**14.2**	13.0	11.3	–	73.2	66.3	75.2	–

Table 3: Per-method **hard** F1 score in %. The best score for each model per dataset is bold.

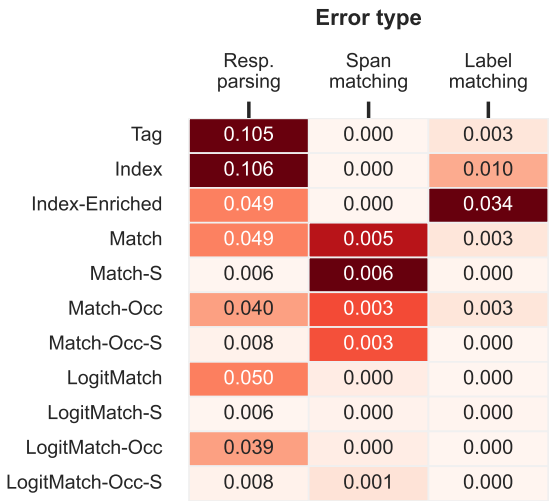

Figure 2: Error rates for each method (highlighted for each column separately)across all datasets. We report three kinds of errors: error in parsing the model response, error in matching the span content, and error in matching the category label to one of the existing labels.

- **Category label errors**: The category label generated by the model did not match any of the expected labels.

We also provide the *soft* F1-score results in Table 8 in Appendix C.

LOGITMATCH improves matching performance in non-standard inputs. Our method improves the results on the NER and GEC tasks where the input text is tokenized using traditional NLP conventions (e.g., spaces around punctuation). With vanilla *matching* methods, LLMs tend to normalize the text during generation, leading to mismatches (see Table 4 for an example of such behavior).

Tagging is robust across tasks. The TAG method seems to be the most universal, having consistent performance across tasks. Notably, it consistently outperforms other methods on GEC. We hypothetize that tags can be more naturally wrapped around the exact spot with the grammatical errors, while for a JSON value, it is more natural to output the whole sentence segment. This is supported by the fact that the spans on GEC have an average length in characters similar to the reference (42.4 vs. 59.2), while the spans of *matching* models are much longer (198.0). We also noticed this trend on other tasks (see Table 6 in Appendix C.2 for more details). However, note that the performance of the TAG method is also partially aided by the fact that we match the content of the span to its closest occurrence in the input text, which allows for minor discrepancies in the surrounding text and minimizes parsing errors (see Appendix B.2).

Indexing is not reliable without indices. Directly asking models to predict numerical indices (INDEX) leads to low performance on all tasks (below 24% for open LLMs). As illustrated in Table 4, models frequently hallucinate indices that ignore word boundaries or fall entirely outside of the text. Adding explicit numbering to the input (INDEX-ENRICHED) can sometimes significantly improve the situation (e.g., improvement by 21-46% points on NER). However, this method still falls short on GEC and ESA-MT. A plausible explanation is that the inserted indices interfere with the ability of the model to detect the errors.

Task: NER	**Model: Llama-3.3-70B**
Input	On Thursday , the 3rd of November , the mayor of **Saint - Gaudens** (in the **Haute - Garonne** region) decided to suspend the urban development agreement with the government formed in 2014 .
Match	`{"text": "Saint-Gaudens", "label": "LOC"}, {"text": "Haute-Garonne", "label": "LOC"}`
LogitMatch	`{"text": "Saint - Gaudens", "label": "LOC"}, {"text": "Haute - Garonne", "label": "LOC"}`
Task: ESA-MT	**Model: gpt-5-mini**
Source	anatomically accurate green M&M
Target (orig)	anatomicky správný zelený M&M
Target (enriched)	0::anatomicky 11::správný 19::zelený 26::M&M
Index	`[0:22] = MINOR` *anatomicky správný zelený M&M*
Index-Enriched	`[11:19] = MAJOR` *anatomicky správný zelený M&M*

Table 4: Qualitative examples comparing different span labeling methods. In the NER task, the **Match** method fails to reproduce the input tokenization (spaces around hyphens), while **LogitMatch** correctly adheres to it. In the ESA-MT task, the **Index** method generates indices that are ignoring word boundaries, whereas **Index-enriched** helps the model locate the span correctly. Problematic outputs are highlighted in red.

Using the `occurence_index` field helps with matching multiple occurences. In the CPL task, the performance of *matching* methods improves dramatically (by ~30-40% points) by using the `occurrence_index` for indexing the span content. This makes sense as the *matching* methods otherwise have no way to refer to additional occurrences of the pattern on the input. However, for tasks where the spans are mostly unique and unambiguous, such as NER and ESA-MT, this feature does not provide a clear benefit.

Structured outputs do not uniformly improve performance While structured outputs expectedly minimize parsing errors[9] (see Figure 2), restricting an output format naively may hurt model performance. Specifically, we found that the models *without* structured outputs sometimes produce a "spontaneous" chain-of-thought before generating the output, which can help the model to improve its answer.

Explicit reasoning helps, but slows down inference. The issue with restricting the output does not arise with reasoning models that explicitly enclose their reasoning trace between `<think>` tags, as this reasoning trace is handled separately in LLM frameworks. Our additional experiments (see Appendix C.3) show that reasoning may boost the model performance – for example, by explicitly indexing individual characters in the reasoning trace. However, this comes at the cost of a much higher number of output tokens during inference, which in our case sometimes lead to hitting the maximum output token limit before producing the answer.

Smaller models can be competitive with larger ones. Since our primary goal was to explore span labeling strategies rather than benchmark models on the tasks, comparisons of model performance should be interpreted with caution. Our results suggest that Qwen3-8B is competitive with the much larger Llama-3.3-70B. However, the factors such as quantization and release date of the models (Qwen3 is newer than Llama 3.3) might be influencing these results. GPT-5-mini (with unknown size) outperforms open models in many setups, but LOGITMATCH cannot be straightforwardly applied to it due to API limitations.

7 Conclusion

We identified and tested three major strategies for span labeling with LLMs and tested them on four span labeling tasks using state-of-the-art LLMs. Overall, the straightforward strategy of *tagging* the spans with XML-like tags has the most consistent results. However, *matching* strategies are more token-efficient and often competitive. We proposed a new method, LOGITMATCH, that solves the alignment issue of *matching* methods. On the tasks with multiple occurrences of same spans, *indexing* with indexed input can minimize output post-processing while providing competitive performance. Future work might explore how to combine the advantages of individual methods and how prompting with different examples and instructions influences the efficiency of each method.

[9]The error rate is non-zero since the models may still fail to return a valid JSON, e.g. if they gets stuck in an infinite loop and reach the maximum number of output tokens.

Limitations

We focused only on the span labeling approaches that involve parsing output text, as we find these the most generally applicable. Therefore, we did not explore alternative approaches that are more resource intensive, such as LLM2Vec (BehnamGhader et al., 2024) that transforms decoder-only LLMs into encoders with additional finetuning, or the approach of Dukić and Šnajder (2024) that allows the LLM to join span labeling with input processing by means of architecture modifications.

Our proposed method does not deal with the disambiguation of multiple spans. As we have shown, our method can be combined with `occurence_index` to achieve that effect. However, this approach is not perfect. Future work might thus explore other approaches, for example, disambiguating the span using attention values.

In principle, a variant of LOGITMATCH is also applicable to *tagging* methods, where it may constrain the decoded text to correspond to the input text. However, our preliminary experiments ran into severe issues with tokenization handling, so we abandoned this approach. Future attempts should carefully handle tokens at the boundaries between the outer text, the tag itself, and the inner text to avoid performance degradation.

Ethical Considerations

To the best of our knowledge, our work does not pose any immediate ethical risks. Our proposed method is designed to improve the reliability of text analysis tools, which we believe is a helpful contribution to the field. For our experiments, we relied solely on existing, publicly available datasets. We did not collect any new data or employ human annotators for this project. We used AI assistants to help with writing experimental code and to improve the clarity of the text. All generated content was manually reviewed and verified by the authors.

References

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. Llm2vec: Large language models are secretly powerful text encoders. In *First Conference on Language Modeling*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017 4, Volume 1: Long Papers*, pages 793–805, Vancouver, Canada.

Giovanni Da San Martino, Seunghak Yu, Alberto Barrón-Cedeño, Rostislav Petrov, and Preslav Nakov. 2019. Fine-Grained Analysis of Propaganda in News Article. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5636–5646. Association for Computational Linguistics.

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, NAACL-HLT 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, MN, USA.

David Dukić and Jan Šnajder. 2024. Looking right is sometimes right: Investigating the capabilities of decoder-only llms for sequence labeling. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 14168–14181.

Lukas Edman, Helmut Schmid, and Alexander Fraser. 2024. CUTE: Measuring LLMs' Understanding of Their Tokens. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL*, pages 3017–3026, USA.

Tairan Fu, Raquel Ferrando, Javier Conde, Carlos Arriaga, and Pedro Reviriego. 2024. Why Do Large Language Models (LLMs) Struggle to Count Letters? *CoRR*, abs/2412.18626.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

guidance-ai. 2025. Guidance: A language for constraining large language models. `https://github.com/guidance-ai/guidance`. MIT License.

Maram Hasanain, Fatema Ahmad, and Firoj Alam. 2024. Large Language Models for Propaganda Span Annotation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 14522–14532, Miami, Florida, USA.

Antonín Jarolím, Martin Fajčík, and Lucia Makaiová. 2025. Can LLMs extract human-like fine-grained

evidence for evidence-based fact-checking? *arXiv preprint arXiv:2511.21401*.

Zdeněk Kasner and Ondřej Dušek. 2024. Beyond Traditional Benchmarks: Analyzing Behaviors of Open LLMs on Data-to-text Generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024*, pages 12045–12072, Bangkok, Thailand.

Zdeněk Kasner, Vilém Zouhar, Patrícia Schmidtová, Ivan Kartáč, Kristýna Onderková, Ondřej Plátek, Dimitra Gkatzia, Saad Mahamood, Ondřej Dušek, and Simone Balloccu. 2025. Llms as span annotators: A comparative study of llms and humans. *Preprint*, arXiv:2504.08697.

Kristýna Klesnilová. 2025. Multilingual Detection of Persuasion Techniques in Text using Large Language Models.

Tom Kocmi and Christian Federmann. 2023. GEMBA-MQM: Detecting Translation Quality Error Spans with GPT-4. In *Proceedings of the Eighth Conference on Machine Translation, WMT 2023*, pages 768–775, Singapore.

Tom Kocmi, Vilém Zouhar, Eleftherios Avramidis, Roman Grundkiewicz, Marzena Karpinska, Maja Popovic, Mrinmaya Sachan, and Mariya Shmatova. 2024. Error Span Annotation: A Balanced Approach for Human Evaluation of Machine Translation. In *Proceedings of the Ninth Conference on Machine Translation, WMT 2024, Miami, FL*, pages 1440–1453, USA.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz*, pages 611–626, Germany.

Yongqi Li, Mayi Xu, Xin Miao, Shen Zhou, and Tieyun Qian. 2024. Prompting Large Language Models for Counterfactual Generation: An Empirical Study. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024*, pages 13201–13221, Torino, Italy.

Michael Xieyang Liu, Frederick Liu, Alexander J. Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J. Cai. 2024. "We Need Structured Output": Towards User-centered Constraints on Large Language Model Output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, CHI EA 2024*, pages 10:1–10:9, Honolulu, HI, USA.

Arianna Masciolini, Andrew Caines, Orphée De Clercq, Joni Kruijsbergen, Murathan Kurfal, Ricardo Muñoz Sánchez, Elena Volodina, Robert Östling, Kais Allkivi, Špela Arhar Holdt, Ilze Auzina, Roberts Darģis, Elena Drakonaki, Jennifer-Carmen Frey, Isidora Glišić, Pinelopi Kikilintza, Lionel Nicolas, Mariana Romanyshyn, Alexandr Rosen, and 11 others. 2025. Towards Better Language Representation in Natural Language Processing: A Multilingual Dataset for Text-level Grammatical Error Correction. *International Journal of Learner Corpus Research*, 11:309–335.

Stephen Mayhew, Terra Blevins, Shuheng Liu, Marek Suppa, Hila Gonen, Joseph Marvin Imperial, Börje Karlsson, Peiqin Lin, Nikola Ljubesic, Lester James V. Miranda, Barbara Plank, Arij Riabi, and Yuval Pinter. 2024. Universal NER: A Gold-standard Multilingual Named Entity Recognition Benchmark. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024*, pages 4322–4337, Mexico City, Mexico.

Mistral AI Team. 2025. Mistral Small 3. `https://mistral.ai/news/mistral-small-3`. Accessed: 2026-01-02.

Phoebe Mulcaire and Nitin Madnani. 2025. Span Labeling with Large Language Models: Shell vs. Meat. In *Proceedings of the 20th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2025)*, pages 850–859, Vienna, Austria.

Motasem S. Obeidat, Md Sultan Al Nahian, and Ramakanth Kavuluru. 2025. Do LLMs Surpass Encoders for Biomedical NER? In *13th IEEE International Conference on Healthcare Informatics, ICHI 2025*, pages 352–358, Rende, Italy.

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

Alan Ramponi, Agnese Daffara, and Sara Tonelli. 2025. Fine-grained Fallacy Detection with Human Label Variation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque*, pages 762–784, New Mexico, USA.

Lance A. Ramshaw and Mitch Marcus. 1995. Text Chunking using Transformation-based Learning. In *Third Workshop on Very Large Corpora, VLC at ACL 1995*, Cambridge, Massachusetts, USA.

Craig Thomson, Ehud Reiter, and Barkavi Sundararajan. 2023. Evaluating factual accuracy in complex data-to-text. *Comput. Speech Lang.*, 80:101482.

Marcos V. Treviso, Nuno Miguel Guerreiro, Sweta Agrawal, Ricardo Rei, José Pombal, Tânia Vaz, Helena Wu, Beatriz Silva, Daan van Stigt, and André F. T. Martins. 2024. xTower: A Multilingual LLM for Explaining and Correcting Translation Errors. In

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 15222–15239, Miami, Florida, USA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, Guoyin Wang, and Chen Guo. 2025a. GPT-NER: Named Entity Recognition via Large Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4257–4275, Albuquerque, New Mexico, USA.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, Guoyin Wang, and Chen Guo. 2025b. GPT-NER: Named Entity Recognition via Large Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4257–4275, Albuquerque, New Mexico, USA.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. 2025. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025*, pages 2526–2547, Vienna, Austria.

Brandon T. Willard and Rémi Louf. 2023. Efficient Guided Generation for Large Language Models. *CoRR*, abs/2307.09702.

Faren Yan, Peng Yu, and Xin Chen. 2024. LTNER: Large Language Model Tagging for Named Entity Recognition with Contextualized Entity Marking. In *Pattern Recognition - 27th International Conference, ICPR 2024*, volume 15319 of *Lecture Notes in Computer Science*, pages 399–411, Kolkata, India.

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. *CoRR*, abs/2505.09388.

A Experimental Setup

Table 5 presents the overview of our models and their decoding parameters. We run the open models using vLLM (Kwon et al., 2023) in 16-bit precision, except for Llama 3.3 70B that we quantize to 4-bit precision. For structured outputs, we use the vLLM structured output feature.[10] The decoding parameters are the default parameters in the Huggingface configuration file. We set only the `max_tokens` parameter manually to 4,096. For each experiment, we did a single run with the random seed set to 0.

B Implementation Details

B.1 Prompts

The overview of the prompts we use for individual methods and tasks is provided in Table 7.

B.2 Methods

Below, we provide extra implementation details for individual methods.

Tagging For the *tagging* strategy, we parse XML-style tags using regular expressions. To map these predictions back to the original text, we estimate the span's location by stripping tags from the preceding output to calculate a running offset. Since models occasionally alter whitespace or hallucinate, we treat this calculated position as a heuristic rather than an absolute truth. If the text at the expected location does not match the tag content, we search the full text for the span and resolve ambiguities by selecting the occurrence closest to our initial estimate. This allows us to recover correct boundaries even when the model's output deviates slightly from the input.

LogitMatch The pseudocode for the LogitMatch algorithm (Section 4) is described in Algorithm 1. Note that this is a simplified version of the algorithm that does not account for tokenization. We implement our method as a vLLM `LogitsProcessor`.[11]

C Additional Results

C.1 Soft F1

In Table 8, we present the *soft* F1-score results of our models, supplementing Table 3 with the *hard* F1-score. The *soft* F1 score does not penalize differences in category labels, only in span positions.

[10] https://docs.vllm.ai/en/latest/features/structured_outputs/

[11] https://docs.vllm.ai/en/latest/design/logits_processors/

Model	Model ID	temp.	top-p	top-k	Max tok.
Qwen3-8B	`Qwen/Qwen3-8B`	0.6	0.95	20	4,096
Mistral-Small-24B-Instruct	`mistralai/Mistral-Small-24B-Instruct-2501`	0.15	1.0	-	4,096
Llama-3.3-70B-Instruct	`meta-llama/Llama-3.3-70B-Instruct`	0.6	0.9	-	4,096
GPT-5-mini	`gpt-5-mini-2025-08-07`	1.0	1.0	-	4,096

Table 5: Model identifiers and their default decoding parameters that we used for our experiments (temperature, top-p, top-k). We set the maximum number of output tokens to 4,096 for all the experiments.

Algorithm 1 LogitMatch decoding algorithm

Require: Input $X = [x_1, x_2, \ldots, x_n]$, model $\mathcal{M}$

Initialize output sequence Y
while decoding is not finished **do**
  $t \leftarrow$ next token predicted by $\mathcal{M}(Y)$ ▷ **Decoding loop**
  Append t to Y
  if `'"text": "'` was decoded **then** ▷ **Select mode**
    $V_{\text{select}} \leftarrow \{x \mid x \in X\}$ ▷ Limit vocabulary to tokens present in input
    $x_i \leftarrow$ sample from $\mathcal{M}(Y)$ constrained to V_{select}
    Append x_i to Y
    $k \leftarrow i$ ▷ Set current pointer to index of generated token
    loop ▷ **Copy mode**
      $V_{\text{copy}} \leftarrow \{x_{k+1}\} \cup \{\text{"}\}$ ▷ Allow next input token or closing quote
      $t_{\text{next}} \leftarrow$ sample from $\mathcal{M}(Y)$ constrained to V_{copy}
      Append t_{next} to Y
      if t_{next} is " **then**
        break ▷ Exit copy mode
      else
        $k \leftarrow k + 1$ ▷ Advance input pointer
      end if
    end loop
  end if
end while

Method	NER	GEC	ESA-MT	CPL
Reference	16.7	59.2	30.4	75.8
Tagging	16.5	42.4	30.5	38.2
Matching	18.1	198.0	108.7	60.8
LogitMatch	18.2	201.1	107.3	60.9
Indexing	29.1	271.8	117.2	87.5

Table 6: Average number of characters within spans per example.

C.2 Span Lengths

Table 6 shows the average length of the spans predicted by individual methods compared to the average length of the reference spans in each dataset.

C.3 Experiments with Qwen3 Reasoning

Table 9 summarizes our experiments with Qwen-8B with reasoning disabled (our default model) vs. Qwen-8B with reasoning enabled. We note that while in some cases the reasoning helped significantly (e.g. 22% improvement in soft-F1 score on NER with the INDEX method), in some cases the results got worse. Our manual inspection of results has shown that the reasoning trace sometimes got very long, exceeding the 4,096 output `max_token` limit (e.g, when the model started to sequentially enumerate all the characters in the input sequence to get their indices), so it was not able to generate the final output. We were unfortunately not able to re-generate the outputs with a higher token limit due to time constraints.

D Data Preprocessing

We made several adjustments to the MultiGEC dataset (Masciolini et al., 2025) that we used for the GEC task:

- The original MultiGEC dataset contains texts in

12 languages in total. We initially wanted to include all the languages in our experiments. However, we found that for the majority of languages, the inputs (text with errors) and the outputs (corrected texts) are not aligned. Only the *English* subset (Write & Improve corpus) contained the list of edits between the original and corrected sentences, which corresponds directly to the span labeling task. Therefore, we limited our dataset only to the English subset.

- Since we focus mainly on span identification, we ignore the corrected text for the replacement and missing edits. For the same reason, we also use only the high-level categories (replacement, missing, and unnecessary) and ignore the more fine-grained categories.
- The *missing* spans have zero length: their `start_index` is equal to `end_index`, which is the index of the token *before* which the missing text should be inserted. That would be a severe disadvantage for the *matching* methods as they cannot efficiently refer to an empty span. Therefore, we instruct the models to mark the span *before* which the missing text should appear. We then manually set the `end_index` equal to `start_index`.

Prompt template	

Structure	`[Task description]`
	`Output Format: [Format]`
	`Labels: [Label list]`
	`Examples:`
	`1. [Example input] -> [Example output]`
	`...`
	`[Notes]`
	`[Input text]`

Task definitions	
NER	`Extract named entities (PER, ORG, LOC) from the text.`
GEC	`Identify grammatical errors in learner-written text.`
ESA-MT	`Identify translation errors by comparing the translation to the source text. The errors are words, phrases or punctuation marks in the translation that are incorrect.`
CPL	`Find all text spans that match the given pattern queries.`
Tagging strategies	
Format	`<entity type="LABEL">text</entity>`
Note	`IMPORTANT: You must output the entire text, including non-tagged parts. Rewrite the whole input text with XML tags.`
Example output	`<entity type="PER">Lina Berg</entity> joined <entity type="ORG">AstraTech</entity> in <entity type="LOC">Stockholm</entity> after finishing her studies at <entity type="ORG">Northvale University</entity>.`
Matching strategies	
Format	`[{"text": "exact text span from input", "label": "category (PER, ORG, LOC)"}]`
Note	`Return a valid JSON array only.`
Example output	`[ {"text": "Lina Berg", "label": "PER"}, {"text": "AstraTech", "label": "ORG"}, {"text": "Stockholm", "label": "LOC"}, {"text": "Northvale University", "label": "ORG"} ]`
Variant: Matching with Occurrence	
Format	`[{"text": "exact span from input", "label": "category (PER, ORG, LOC)", "occurrence": "which occurrence (1, 2, 3...)"}]`
Example output	`[ {"text": "Lina Berg", "label": "PER", "occurrence": 1}, {"text": "AstraTech", "label": "ORG", "occurrence": 1}, {"text": "Stockholm", "label": "LOC", "occurrence": 1}, {"text": "Northvale University", "label": "ORG", "occurrence": 1} ]`
Indexing strategies	
Format	`[start:end] = LABEL`
Note	`IMPORTANT: Character positions are 0-indexed. - First character is at position 0 - Spaces count as characters - start is inclusive, end is exclusive`
Example output	`[0:9] = PER [17:26] = ORG [30:39] = LOC [71:91] = ORG`

Table 7: Overview of prompts used for each strategy.

Method	NER				GEC				ESA-MT				CPL			
	8B	24B	70B	5-m	8B	24B	70B	5-m	8B	24B	70B	5-m	8B	24B	70B	5-m
TAG	68.1	**66.9**	74.8	**79.5**	**30.4**	**41.0**	**38.2**	**47.4**	19.9	17.9	21.8	26.5	52.9	71.5	**82.2**	**80.6**
INDEX	24.1	19.1	27.7	37.5	11.7	11.6	6.6	9.1	17.6	14.9	14.2	16.3	14.3	17.3	14.9	19.9
INDEX-ENRICHED	45.6	**66.9**	65.4	75.4	16.7	15.2	17.2	31.8	20.1	16.3	15.9	18.4	49.1	**78.1**	68.1	79.9
MATCH	75.4	56.4	**76.2**	61.2	20.2	27.3	31.4	34.2	26.1	26.7	25.7	28.9	30.6	36.9	39.8	44.1
MATCH-S	75.5	51.8	76.0	74.1	21.7	26.8	31.7	33.0	25.9	27.3	25.8	29.0	30.4	37.0	40.1	42.3
MATCH-OCC	75.6	66.1	71.8	69.4	21.3	23.3	33.1	36.3	26.3	27.3	26.4	**29.5**	**73.4**	64.1	74.9	77.4
MATCH-OCC-S	76.1	56.4	70.9	67.8	21.7	24.0	33.7	35.7	**27.0**	27.1	26.3	28.8	73.2	66.6	75.1	79.9
LOGITMATCH	76.4	56.5	76.1	–	20.2	27.1	31.4	–	26.1	26.4	25.3	–	31.0	36.9	40.0	–
LOGITMATCH-S	**76.5**	51.3	76.1	–	20.6	27.9	32.6	–	26.2	26.4	**26.7**	–	31.0	36.8	40.3	–
LOGITMATCH-OCC	76.2	66.5	72.0	–	21.4	24.8	33.8	–	26.8	26.7	**26.7**	–	73.2	63.6	75.0	–
LOGITMATCH-OCC-S	76.0	55.9	72.0	–	22.9	24.6	33.8	–	26.5	**27.4**	26.4	–	73.2	66.3	75.2	–

Table 8: Per-method **soft** F1 score in %. The best score for each model per dataset is bold.

Method	NER				GEC				ESA-MT				CPL			
	8B		8B-T		8B		8B-T		8B		8B-T		8B		8B-T	
	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
TAG	63.4	68.1	82.6	86.6	**24.0**	**30.4**	**24.4**	**37.6**	11.0	19.9	8.4	19.7	52.9	52.9	**77.1**	**77.1**
INDEX	17.8	24.1	59.6	66.7	9.5	11.7	4.1	5.0	7.8	17.6	4.3	10.7	14.3	14.3	5.6	5.6
INDEX-ENRICHED	39.6	45.6	60.2	68.0	10.0	16.7	20.1	24.2	9.0	20.1	4.5	11.0	49.1	49.1	60.0	60.0
MATCH	70.9	75.4	**83.4**	**87.4**	15.2	20.2	22.3	29.3	13.6	26.1	9.6	22.0	30.6	30.6	33.3	33.3
MATCH-S	71.0	75.5	71.4	75.9	16.9	21.7	13.4	17.6	12.9	25.9	12.8	25.9	30.4	30.4	32.8	32.8
MATCH-OCC	71.1	75.6	77.3	81.1	16.6	21.3	20.8	26.0	13.5	26.3	9.9	21.9	**73.4**	**73.4**	69.5	69.5
MATCH-OCC-S	71.5	76.1	68.3	72.4	16.4	21.7	14.6	19.5	13.4	**27.0**	**13.6**	**27.1**	73.2	73.2	73.4	73.4
LOGITMATCH	71.4	76.4	–	–	15.8	20.2	–	–	13.0	26.1	–	–	31.0	31.0	–	–
LOGITMATCH-S	**71.7**	**76.5**	–	–	15.8	20.6	–	–	13.4	26.2	–	–	31.0	31.0	–	–
LOGITMATCH-OCC	71.5	76.2	–	–	16.9	21.4	–	–	13.9	26.8	–	–	73.2	73.2	–	–
LOGITMATCH-OCC-S	71.3	76.0	–	–	17.9	22.9	–	–	**14.2**	26.5	–	–	73.2	73.2	–	–

Table 9: Qwen3-8B F1-score (hard, soft) in % with vs. without reasoning. Qwen3-8B (`8B`) has reasoning capabilities turned off, while Qwen3-8B-Think (`8B-T`) has them turned on. The best score for each model per dataset is bold.