

Step-by-Step CCG Parsing with Large Language Models

Tu-Anh Tran¹ Namgi Han¹ Taiga Ishii² Yusuke Miyao¹

¹The University of Tokyo ²NINJAL

{anh, hng88, yusuke}@is.s.u-tokyo.ac.jp taiga.ishii@ninjal.ac.jp

Abstract

We study how step-by-step reasoning should be structured for Combinatory Categorical Grammar (CCG) parsing with Large Language Models (LLMs). Whereas most CCG parsers fix the order of parsing decisions—supertagging every word first, then combining the categories bottom-up into a derivation—for LLMs that can generate the full derivation as a single output sequence, this order is an open design question. We fine-tune three open-weight models on four reasoning paths along two axes: parsing direction (top-down or bottom-up) and step format (lexically grounded partial derivations or symbolic derivation steps). Our results show that not all reasoning paths improve CCG parsing. On English CCGbank, only the top-down lexically grounded path consistently outperforms a direct, no-reasoning baseline across all three models. In a zero-shot cross-lingual transfer evaluation from English to German, Italian, and Dutch, the same path outperforms a strong CCG parser baseline, despite trailing it on English. Our findings suggest that LLMs may benefit from a different parsing strategy than classical CCG parsers.

1 Introduction

Combinatory Categorical Grammar (CCG; Steedman, 2000) is a lexicalized grammar formalism in which most syntactic information is carried by the lexicon, and adjacent constituents combine through a small set of combinatory rules to form a derivation. CCG offers a transparent syntax-semantics interface, which has made it useful in various tasks such as natural language inference (Sugimoto and Yanaka, 2022), compositional semantics (Funakura and Mineshima, 2023), and modeling human sentence processing (Stanojević et al., 2023).

Most CCG parsers follow a bottom-up pipeline: a supertagger first assigns a lexical category to every word, and a separate parser then combines them into a derivation (Clark and Curran, 2007;

Zhang and Clark, 2011; Lewis and Steedman, 2014). These parsers are trained on CCGbanks, which exist for only a handful of languages: methods to build them include converting an existing treebank (Hockenmaier and Steedman, 2007) or projecting derivations from English through parallel data (Abzianidze et al., 2017). Large language models (LLMs) offer a different approach: a single model can be fine-tuned to generate the full derivation in one autoregressive pass and, through cross-lingual transfer, applied to languages without their own CCGbanks. This could lower the barrier to extending CCG parsing to new languages; however, how LLMs should be applied is not obvious.

Recently, step-by-step reasoning has emerged as a way to improve LLM performance on complex tasks (Wei et al., 2022; Kojima et al., 2022; Nye et al., 2021). We ask two questions. First, do intermediate reasoning steps help at all for CCG parsing with LLMs, or can an LLM produce the final derivation directly just as well? Second, if intermediate steps help, how should they be structured? It is not obvious whether an LLM should adhere to the bottom-up pipeline of classical CCG parsers or follow a different approach.

To answer these questions, we evaluate four reasoning paths along two axes: parsing direction (**top-down** or **bottom-up**) and step format (**lexically grounded** partial derivations or **symbolic** derivation steps). We fine-tune three open-weight base models (Llama-3.1-8B (Grattafiori et al., 2024), Qwen3-8B-Base (Yang et al., 2025), and Olmo-3-1025-7B (Ettinger et al., 2026)) on each path with LoRA (Hu et al., 2022), and evaluate on English CCGbank (Hockenmaier and Steedman, 2007) and on the German, Italian, and Dutch portions of the Parallel Meaning Bank (Abzianidze et al., 2017) in a zero-shot cross-lingual setting.¹

¹Our implementation is available at <https://github.com/attnr/llm-ccg-parsing>.

Our main findings are:

- Intermediate reasoning steps do not always help CCG parsing with LLMs. Among four reasoning paths explored, only the lexically grounded top-down path (LexTD) consistently outperforms a no-reasoning baseline; the remaining paths either match the baseline or, in the case of symbolic top-down, fall below it. Notably, LexTD runs counter to the bottom-up direction of classical CCG pipelines.
- In the zero-shot cross-lingual transfer setting, LexTD fine-tuned on Llama-3.1-8B and Qwen3-8B-Base outperforms a strong baseline CCG parser (Hol-CCG; Yamaki et al. 2023). This provides evidence that LLMs may transfer CCG parsing knowledge to new languages more effectively than a specialized parser with a multilingual encoder.

Together, these findings suggest LLMs may benefit from a different parsing strategy than classical parsers, inviting further research into this area.

2 Background

In CCG, each word is assigned a category that determines how it can combine with surrounding words. Categories are either atomic (e.g., S , NP , N) or complex, built recursively using two slash operators: X/Y denotes a function that consumes a Y category on its right and produces an X , and $X\backslash Y$ does the same on the left. A small set of combinatory rules governs how adjacent categories combine into larger ones, such as Forward Application ($X/Y Y \Rightarrow X$) and Backward Application ($Y X\backslash Y \Rightarrow X$). A CCG derivation pairs each word with a lexical category and combines adjacent constituents using the combinatory rules; the resulting derivation is generally binary-branching, though unary rules such as type-raising are also permitted.

The standard approach to CCG parsing decomposes this task into two stages. The first stage, supertagging (Bangalore and Joshi, 1999; Clark, 2002), assigns a small set of likely categories to each word in the sentence; this resolves much of the local syntactic ambiguity and is sometimes referred to as “almost parsing” (Bangalore and Joshi, 1999). The second stage uses these categories to build a derivation, using algorithms such as chart parsing (Clark and Curran, 2007), transition-based parsing (Zhang and Clark, 2011), and A* search

(Lewis and Steedman, 2014). The pipeline as a whole is bottom-up: supertags are decided before any combination takes place.

3 Related Work

3.1 Step-by-Step Reasoning in LLMs

LLMs have been shown to perform better on many tasks when prompted to produce intermediate reasoning steps before a final answer, an approach known as Chain-of-Thought prompting (Wei et al., 2022; Kojima et al., 2022) or scratchpads (Nye et al., 2021). These intermediate steps may take different forms depending on the task: free-form natural language for arithmetic and commonsense reasoning (Wei et al., 2022), or symbolic states for tasks that simulate program execution or other deterministic procedures (Nye et al., 2021).

3.2 Syntactic Parsing with Generative Models

For dependency parsing, Hromei et al. (2024) fine-tune LLaMA-based models in a sequence-to-sequence setup and report results on par with state-of-the-art parsers across 26 languages, while Matsuda et al. (2025) fine-tune LLMs with step-by-step prompts (POS, heads, then labels) and a simplified tabular output format, achieving state-of-the-art accuracy on UD across 17 languages. Without fine-tuning, however, Ezquerro et al. (2025) find that most open-weight LLMs cannot outperform simple uninformed baselines such as right-branching trees. For constituency parsing, Bai et al. (2025) evaluate LLMs in zero-shot, few-shot, and fine-tuning settings, finding that they often produce structurally invalid or unfaithful trees in all three. Tian et al. (2024) report a similar tendency for prompted LLMs to produce overly flat trees, and propose a three-stage chunking pipeline that first identifies constituents and then assembles them into trees. Zhang et al. (2025) take a different angle, using grammar rules from existing treebanks to guide LLMs in correcting their errors.

Several other works are relevant to our specific design choices. Drozdov et al. (2023) use syntactic decomposition as a scaffold for least-to-most prompting in compositional semantic parsing. For compositional syntactic language models, Zhao et al. (2025) compare top-down and bottom-up linearizations of parse trees, reporting differences across language modeling, syntactic generalization, and downstream tasks. On CCG specifically, Zhao and Penn (2024) reach state-of-the-art performance

	Lexically grounded	Symbolic
Top-down	<pre> <STEPS> <STEP>{S A river flows}</STEP> <STEP>{S {NP A river} {S\NP flows}}</STEP> <STEP>{S {NP {NP/N A} {N river}} {S\NP flows}}</STEP> </STEPS> <FINAL> {S {NP {NP/N A} {N river}} {S\NP flows}} </FINAL> </pre>	<pre> <STEPS> C1 := combine(C2, C3) : S C2 := combine(C4, C5) : NP C3 := flows : S\NP C4 := A : NP/N C5 := river : N </STEPS> <FINAL> {S {NP {NP/N A} {N river}} {S\NP flows}} </FINAL> </pre>
Bottom-up	<pre> <STEPS> <STEP>{NP/N A} {N river} {S\NP flows}</STEP> <STEP>{NP {NP/N A} {N river}} {S\NP flows}</STEP> <STEP>{S {NP {NP/N A} {N river}} {S\NP flows}}</STEP> </STEPS> <FINAL> {S {NP {NP/N A} {N river}} {S\NP flows}} </FINAL> </pre>	<pre> <STEPS> C1 := A : NP/N C2 := river : N C3 := flows : S\NP C4 := combine(C1, C2) : NP C5 := combine(C4, C3) : S </STEPS> <FINAL> {S {NP {NP/N A} {N river}} {S\NP flows}} </FINAL> </pre>

Table 1: The four reasoning paths for the sentence *A river flows*. The final block is identical across all four paths.

on supertagging with LLMs, but cover only lexical category assignment and leave derivation construction to a separate parser. Beyond the pipeline setting, joint approaches to CCG supertagging and parsing also exist: [Auli and Lopez \(2011\)](#) combine supertagging and parsing features in a single model, using loopy belief propagation and dual decomposition for inference.

4 Methodology

We frame CCG parsing as a non-incremental parsing task where the LLM, given a sentence, generates intermediate reasoning steps followed by the final derivation as a single output sequence. This differs from the two-stage pipeline in Section 2. Two possible design choices then follow: the direction in which the derivation is built, and the format of each intermediate step.

For direction, we contrast **top-down** and **bottom-up**, the two strategies that have dominated syntactic parsing in both classical ([Collins, 1997](#); [Charniak, 2000](#)) and neural ([Stern et al., 2017](#); [Kitaev and Klein, 2018](#)) settings. For step format, we consider two ways of representing the partial derivation: a **lexically grounded** format that shows the partial tree explicitly at each step, and a **symbolic** format that records the derivation as a sequence of variable assignments resembling programming code.

Intuitively, these formats reflect two reasoning paradigms tied to different aspects of LLM pretraining. Because LLMs are trained on natural language, they may benefit from a grounded format that pre-

serves surface tokens to guide syntactic decisions through lexical context. Conversely, because they are also trained on code, they may benefit from a symbolic format that frames the derivation as a sequence of algebraic steps. These formats probe whether the model requires surface anchoring or can build derivations through only variables.

Crossing these two axes gives four reasoning paths (Table 1), described below.

4.1 Parsing Direction

In the **top-down** setting, the model starts from the category of the full sentence and recursively expands each constituent into the children that combine to produce it, until every leaf is a single token paired with its lexical category. In the **bottom-up** setting, the model first assigns a lexical category to each token, then combines adjacent constituents using CCG combinatory rules, and continues until a single constituent spans the sentence. Both directions reach the same derivation; they differ only in the order in which it is built.

4.2 Step Format

The model’s output for each input sentence consists of two blocks: a sequence of intermediate steps marked by `<STEPS>` and `</STEPS>`, followed by the final derivation marked by `<FINAL>` and `</FINAL>`. The two formats differ only in how the steps inside `<STEPS>` refer to the constituents being built.

Lexically grounded. Each step is wrapped in `<STEP>` tags and shows the partial derivation so far: a bracketed structure containing the surface tokens and the categories built so far.² The model adds brackets and a category for each new constituent at each step as the partial derivation grows, with the final step containing the complete tree. The bracketing convention follows the HTML version of CCGbank (Hockenmaier and Steedman, 2007), in which each constituent is a category followed by its children inside curly braces, of the form `{category child1 child2 ...}`.

Symbolic. Each constituent is defined as a variable and referenced by name in other steps. A step takes one of three forms: a lexical assignment `C1 := w : c`, a binary combination `C4 := combine(C1, C2) : c`, or a unary rule application `C5 := unary(C3) : c`, where the lowercase `c` is a CCG category, `w` is a token, and `C1, C2, ...` are variable names. Each step is thus a single operation that defines one new constituent. Symbolic sequences typically contain fewer tokens overall than lexically grounded ones.

The final derivation inside the `<FINAL>` block is in the same bracketed form for all four paths. The two formats encode the same derivations and are interconvertible, so any difference in parsing accuracy reflects how the format interacts with the model rather than what it expresses.

4.3 Reasoning Paths

Crossing the two axes yields four reasoning paths: lexically grounded top-down (**LexTD**), lexically grounded bottom-up (**LexBU**), symbolic top-down (**SymTD**), and symbolic bottom-up (**SymBU**). Table 1 shows all four for the sentence *A river flows*. For each path, the training target is constructed from the gold CCG derivation by traversing it in the path’s direction and rendering each constituent in the path’s format. The model is fine-tuned to generate the resulting sequence, including the `<STEPS>` and `<FINAL>` blocks. The contents of the `<FINAL>` block are used for evaluation in Section 6; the intermediate steps are used for analysis in Section 7.

The four paths differ in the number of output tokens generated per sentence. Relative to a no-reasoning baseline that emits only the final deriva-

tion, SymTD and SymBU produce around $3.2\times$ as many tokens, LexTD around $9\times$, and LexBU around $14\times$. The lexically grounded format repeats large portions of the partial derivation across steps, which accounts for most of this gap. Full counts by tokenizer are in Table 8 of the Appendix.

4.4 Prompts

The prompt for each reasoning path has two parts: a task description that specifies the required output format, and the input sentence to parse. We use a different task description for each of the four reasoning paths, since each one expects a different output format. The same task description is used at training and inference time. The full prompts are provided in Appendix D.

5 Experimental Setup

We run two sets of experiments: English CCG parsing on the CCGbank, and cross-lingual transfer on the Parallel Meaning Bank (PMB).

5.1 Data

English CCGbank. CCGbank (Hockenmaier and Steedman, 2007) is a corpus of CCG derivations extracted from the Wall Street Journal portion of the Penn Treebank. We use the standard splits: Sections 02-21 for training, Section 00 for development, and Section 23 for testing.

Cross-lingual PMB. The Parallel Meaning Bank (Abzianidze et al., 2017) is a corpus of CCG derivations annotated for English, German, Italian, and Dutch. Since the same document may appear in multiple languages, to preserve a strict zero-shot setting, we remove from the English training set any document that also appears in the development or test sets of other languages. We then train on the cleaned English training set and evaluate zero-shot on the development and test sets of all four languages. We strip all features, such as `[dc1]` or `[ng]`, from the categories, since the feature inventory is not consistent across the four languages. We use the gold-annotated portion of PMB 4.0 only. PMB sentences are generally shorter than English CCGbank sentences (around 5-6 words on average, against 23).

Tables 6 and 7 in Appendix A show the number of sentences in the training, development, and test splits of each treebank.

²The `<STEP>` tags delimit individual steps because each lexically grounded step is itself a forest of bracketed trees, and the boundaries between successive forests would otherwise blur. The symbolic format needs no such delimiters because each step is a single assignment on its own line.

5.2 Models

We run our experiments on three open-weight base models: Llama-3.1-8B, Qwen3-8B-Base, and Olmo-3-1025-7B.

5.3 Baselines

Our experiments include five baselines.

The **Direct** baseline fine-tunes the same models to generate the final CCG derivation tree without any intermediate steps. This isolates the contribution of step-by-step generation from that of fine-tuning itself.

The **Two-step** baseline mirrors the pipeline of Section 2: the model first generates the lexical category for every token, formatted as {c1 w1} {c2 w2} . . . , and then the final derivation tree.

The **5-shot** baseline runs the same base models with the Direct prompt in a 5-shot setting *without parameter updates*. The five in-context demonstrations are randomly sampled from the English training set and fixed across test sentences. This isolates the contribution of fine-tuning from that of model capacity.

The **GPT-5.4** baseline runs OpenAI’s GPT-5.4 (OpenAI, 2026) (version 2026-03-05) in the same 5-shot Direct setting, with reasoning effort set to “none” or “high”.

The **Hol-CCG** baseline is the parsing model of Yamaki et al. (2023), which composes phrase representations using circular correlation in a continuous vector space and reaches state-of-the-art performance on English CCGbank. Yamaki et al. (2023) also provide their implementation. For the English CCGbank, we report their score directly. For the cross-lingual PMB, we replace its RoBERTa-large encoder (Liu et al., 2019) with XLM-R-large (Conneau et al., 2020)³ to allow cross-lingual application, and train only on the English portion of PMB, like the LLM-based methods.

5.4 Training and Inference

We fine-tune all LLMs using LoRA (Hu et al., 2022) on the training set of each experiment. For each LLM-based configuration, we train with three random seeds and report mean and standard deviation across seeds. The same LoRA configuration and training schedule are used across all three models, with early stopping on development-set loss; full hyperparameters are reported in Appendix B.

³XLM-R is the multilingual counterpart of RoBERTa, sharing the same architecture, and remains one of the strongest open multilingual encoders for cross-lingual transfer.

Model	Variant	SV	Cat	Dep LF
Yoshikawa et al. (2017)		–	88.80	
Tian et al. (2020)		96.25	90.58	
Clark (2021)		96.50	92.90	
Yamaki et al. (2023)		96.60	92.12	
Zhao and Penn (2024)		96.64	92.05	
Llama-3.1-8B	Direct	96.08	90.02	85.66
	Two-step	96.20	90.00	84.65
	LexTD	99.71	95.48	89.48
	LexBU	97.87	92.60	85.31
	SymTD	98.24	89.61	84.15
	SymBU	98.95	93.73	86.48
Olmo-3-7B	Direct	82.33	73.48	75.15
	Two-step	91.44	83.36	80.40
	LexTD	94.97	88.82	85.44
	LexBU	92.80	85.12	79.77
	SymTD	74.02	65.92	67.67
	SymBU	92.11	83.16	78.55
Qwen3-8B	Direct	96.48	89.75	84.54
	Two-step	96.12	90.28	84.84
	LexTD	99.56	94.90	88.43
	LexBU	98.73	93.50	85.61
	SymTD	98.49	89.35	83.68
	SymBU	98.83	93.36	85.63
Llama-3.1-8B (5-shot)		3.61	0.77	0.03
Olmo-3-7B (5-shot)		4.74	0.82	0.02
Qwen3-8B (5-shot)		8.10	1.52	0.06
GPT-5.4 (none, 5-shot)		11.59	4.60	1.31
GPT-5.4 (high, 5-shot)		71.04	51.98	39.81

Table 2: Results on CCGbank Section 23. Variant abbreviations defined in Section 4.3. **SV** is the percentage of structurally valid outputs; **Cat** is supertagging accuracy; **Dep LF** is labeled dependency F1. The 5-shot rows use the Direct prompt without fine-tuning. Values for LLM-based rows are means across 3 random seeds; standard deviations are reported in Appendix E. Best variant per column within each fine-tuned model is in bold.

For inference, we use greedy decoding (see Appendix F for a preliminary experiment with constrained decoding). For the Hol-CCG baseline on PMB, we train with ten random seeds and the same hyperparameter settings as Yamaki et al. (2023).

5.5 Evaluation Metrics

For English CCGbank, we evaluate using the standard CCG **dependency F1** score, which measures the agreement between the predicate-argument dependencies extracted from the predicted derivation and those extracted from the gold derivation, following Clark and Curran (2007).

For the cross-lingual PMB evaluation, no equivalent dependency-extraction script is available. We therefore use EVALB,⁴ a constituent-based parser evaluation tool that measures **bracketed F1** be-

⁴<http://nlp.cs.nyu.edu/evalb/>

tween the predicted and gold tree structures. The bracketed F1 score is an approximate measure of parsing quality: it does not check whether each combination follows a valid CCG combinatory rule, which the dependency F1 implicitly does. The models are trained to produce derivations that include punctuation, but we exclude it from the evaluation, since its annotation conventions differ across the four languages. For example, punctuation attaches at a different depth in English than in the other three languages, which would inflate cross-lingual differences.

In addition to F1, we report **structural validity** (SV): the percentage of outputs whose tree structure can be read by our bracketed-tree reader. This requires properly nested brackets, well-formed CCG categories,⁵ and at most two children per internal node. Outputs that are structurally invalid count as parsing failures and contribute zero to F1, rather than being excluded from evaluation, which ensures comparability with prior work that achieves 100% or near-100% parsing coverage.

Since neither bracketed F1 nor structural validity verifies whether each combination is licensed, for PMB we additionally report **application validity** (AppV), the percentage of individual rule applications that are valid, and **derivation validity** (DerivV), the percentage of derivations in which every application is valid. We validate each binary node against the CCG combinatory rule schemas, and each unary node against the type-raising and type-changing rules attested in the PMB training data of the four languages. Both measures are computed over structurally valid outputs.

6 Results

6.1 CCGbank

LexTD is the best path for every model on all three metrics. Table 2 reports parsing results on CCGbank Section 23. Olmo-3-1025-7B has an unusually weak Direct baseline (75.15 LF), against which most reasoning paths gain; on Llama-3.1-8B and Qwen3-8B-Base, where Direct is already competitive, LexTD is the only path that consistently improves by more than 1 point over the baseline. All four reasoning paths produce a higher percentage of structurally valid derivations than either fine-tuned baseline, with one exception (SymTD on

⁵Concretely: atomic categories from a fixed inventory, or X/Y or $X \setminus Y$ where X and Y are themselves well-formed. This rules out ill-formed strings such as $S/$ or $(NP \setminus NP).B$.

Model	Variant	Labeled Bracketed F1			
		EN	DE	IT	NL
Llama-3.1-8B	Direct	89.26	64.74	68.70	66.82
	Two-step	89.02	67.75	68.36	68.96
	LexTD	90.36	69.50	72.86	71.92
	LexBU	88.47	68.18	67.97	68.91
	SymTD	88.47	62.28	66.18	62.23
	SymBU	89.46	68.88	70.10	70.16
Olmo-3-7B	Direct	83.61	51.50	54.47	53.28
	Two-step	83.32	58.81	61.54	57.82
	LexTD	88.23	62.46	67.00	63.91
	LexBU	87.28	61.31	64.42	60.11
	SymTD	77.95	51.70	56.88	51.47
	SymBU	78.82	59.48	65.15	57.78
Qwen3-8B	Direct	87.28	65.34	66.10	65.50
	Two-step	89.61	68.15	69.71	66.82
	LexTD	90.10	71.35	73.93	72.79
	LexBU	88.22	65.36	69.05	66.01
	SymTD	88.24	66.23	68.83	66.22
	SymBU	89.40	69.43	71.59	71.78
GPT-5.4 (none, 5-shot)		65.74	61.56	62.74	59.01
GPT-5.4 (high, 5-shot)		82.36	72.76	74.76	73.35
Hol-CCG (XLM-R-large)		90.81	67.36	69.51	71.14
Model	Variant	Supertagging accuracy			
		EN	DE	IT	NL
Llama-3.1-8B	Direct	90.87	69.99	75.00	72.62
	Two-step	91.96	77.50	75.74	78.73
	LexTD	93.48	78.55	83.94	82.16
	LexBU	92.16	78.33	78.22	79.87
	SymTD	90.10	65.03	71.59	63.85
	SymBU	92.61	81.91	85.11	83.65
Olmo-3-7B	Direct	82.64	46.15	50.27	48.10
	Two-step	83.47	60.93	64.83	57.08
	LexTD	89.75	65.38	71.54	65.87
	LexBU	90.33	67.12	68.37	63.21
	SymTD	74.18	49.44	54.98	48.05
	SymBU	76.20	64.90	71.17	59.89
Qwen3-8B	Direct	88.17	71.04	69.29	72.51
	Two-step	91.98	77.84	78.17	75.18
	LexTD	92.80	80.69	85.69	83.32
	LexBU	92.08	75.74	77.18	75.12
	SymTD	89.88	71.20	73.15	70.60
	SymBU	92.99	82.88	85.74	83.92
GPT-5.4 (none, 5-shot)		61.51	59.20	63.05	55.71
GPT-5.4 (high, 5-shot)		79.65	79.73	86.10	80.26
Hol-CCG (XLM-R-large)		95.10	74.50	79.12	78.94

Table 3: Results on the PMB test set, evaluated zero-shot on German (DE), Italian (IT), and Dutch (NL) after training on English (EN). Variant abbreviations as in Table 2. Values are means across 3 random seeds for fine-tuned LLMs and 10 seeds for Hol-CCG. Best variant per column within each fine-tuned model in bold.

Olmo-3-1025-7B, which we discuss in Section 7).

Comparison with prior work. Our best labeled F1 of 89.48, achieved by LexTD with Llama-3.1-8B, sits below the current state of the art of 92.90

Model	Variant	SV				AppV				DerivV			
		EN	DE	IT	NL	EN	DE	IT	NL	EN	DE	IT	NL
Llama-3.1-8B	Direct	99.78	97.80	98.12	97.90	99.49	97.74	98.90	97.69	97.37	88.54	94.91	89.59
	Two-step	99.18	95.37	98.26	97.22	98.35	93.45	94.98	92.81	93.33	72.22	79.18	71.50
	LexTD	99.75	96.89	99.20	97.83	99.70	98.56	98.75	98.67	98.51	93.01	94.97	93.75
	LexBU	99.21	97.20	99.13	98.37	98.30	92.38	93.88	92.13	92.75	69.72	75.13	68.95
	SymTD	99.47	98.41	96.89	97.01	99.19	98.01	98.16	97.95	96.60	90.52	91.79	89.89
	SymBU	99.68	99.51	99.93	99.39	98.53	93.42	96.05	93.03	94.44	77.04	83.79	77.26
Olmo-3-7B	Direct	95.09	90.80	91.90	91.86	98.98	95.96	97.26	96.62	95.59	81.74	88.61	84.93
	Two-step	97.22	92.50	95.30	93.49	97.49	89.80	90.08	89.86	89.74	60.29	59.33	57.97
	LexTD	98.07	94.15	97.61	94.84	99.41	96.62	97.47	97.01	97.46	85.29	90.16	87.41
	LexBU	98.96	97.44	97.11	97.49	97.30	88.16	87.93	86.76	89.44	55.72	53.38	50.98
	SymTD	92.21	90.61	92.91	90.22	98.05	94.73	95.14	94.64	91.85	76.70	79.92	77.65
	SymBU	92.08	89.76	94.00	87.64	96.81	92.14	94.37	90.83	90.00	73.27	78.42	70.75
Qwen3-8B	Direct	98.89	96.83	96.17	97.76	99.16	96.21	98.13	97.51	95.93	83.93	91.68	87.46
	Two-step	99.46	98.54	98.63	98.10	98.52	92.51	94.48	92.52	93.67	69.76	76.32	68.56
	LexTD	99.68	99.27	99.50	98.64	99.55	98.22	98.94	98.62	97.65	91.66	94.98	94.02
	LexBU	99.59	98.30	98.84	98.23	97.84	90.44	91.82	91.54	91.58	62.29	69.78	64.63
	SymTD	99.56	98.96	98.92	98.92	99.46	98.69	98.46	98.09	97.65	93.66	92.77	91.36
	SymBU	99.31	98.84	99.42	99.32	98.46	94.73	96.82	94.15	94.04	81.08	88.94	77.93

Table 4: Structural validity (SV), application validity (AppV), and derivation validity (DerivV) on the PMB test set. AppV and DerivV are computed over structurally valid outputs. Values are means across three random seeds; standard deviations are reported in Appendix E. Best variant per column within each model in bold.

(Clark, 2021) and below the LLM-based result of 92.05 from Zhao and Penn (2024). Two factors account for this gap. First, training technique: Zhao and Penn (2024) repeat the input sentence five times during training, which we do not adopt to keep our experiments simple and our comparison across reasoning paths clean. Second, task scope: in both prior systems, the final derivation is constructed by a separate grammar-based parser, while our LLM generates both the lexical categories and the derivation in a single autoregressive pass.

Without fine-tuning, the three base models struggle to produce valid CCG derivations, and GPT-5.4 falls well short of the fine-tuned models even with high reasoning effort.

6.2 Cross-lingual PMB

LexTD transfers best among reasoning paths.

Table 3 reports test-set results on PMB. LexTD is the best path on labeled bracketed F1 for every model on every language. On English, it comes within roughly 1 point of Hol-CCG (90.81 LF), but does not surpass it on any of the three base models. On the zero-shot languages, LexTD with Llama-3.1-8B and Qwen3-8B-Base improves over Hol-CCG by up to 4 points, with the largest gap on Italian. With these two base models, several other LLM paths also outperform Hol-CCG on individual non-English languages, though by smaller margins. The pattern with Llama-3.1-8B and Qwen3-8B-

Base is consistent: stronger English performance with Hol-CCG, stronger cross-lingual transfer with LLM-based methods, led by LexTD.

Averaged across reasoning paths, the gap from English to non-English performance is consistent with the models’ multilingual pretraining: Olmo-3-1025-7B, pretrained on a predominantly English corpus, trails by around 25 LF points on average across the three non-English languages, while Llama-3.1-8B and Qwen3-8B-Base trail by around 21 and 20 points respectively.

Supertagging accuracy. Supertagging accuracy is less consistent across paths, with LexTD, LexBU, and SymBU each leading on different model-language combinations. On English, Hol-CCG again leads at 95.10 against the strongest fine-tuned LLM (Llama-3.1-8B with LexTD at 93.48).

GPT-5.4 without fine-tuning. GPT-5.4 with high reasoning is competitive on both metrics for non-English languages, outperforming Hol-CCG by 1 to 7 supertagging points and exceeding the best fine-tuned LLM path on labeled F1 across all three target languages. On English, however, it trails the fine-tuned LLMs and lags Hol-CCG by 8 labeled F1 points.

Validity measures. Table 4 reports the three validity measures. The cross-lingual outputs are largely valid CCG derivations, not merely struc-

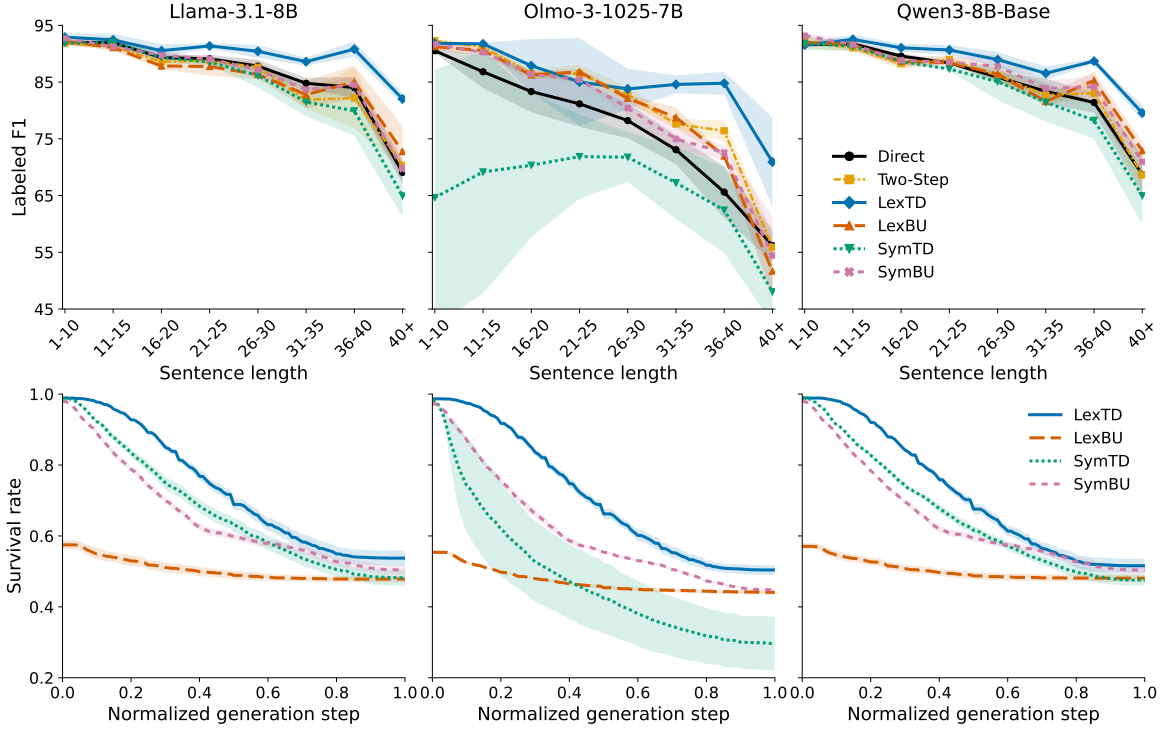


Figure 1: Diagnostics on CCGbank Section 00. **Top:** labeled F1 by sentence length, in eight bins from 1-10 to 40+ tokens. **Bottom:** survival rate by normalized generation step, for the four reasoning paths across three base models. Shaded bands show standard deviation across three seeds.

tures that match the gold trees. LexTD has the highest application and derivation validity in most model-language combinations, with 91.7-98.5% of its derivations free of invalid applications on Llama-3.1-8B and Qwen3-8B-Base. Validity also degrades far less under transfer than labeled bracketed F1 does: for LexTD on these two models, labeled F1 falls by 17-19 points from English to the three transfer languages on average, while structural and application validity fall by around 1-2 points and derivation validity by 4-5.

7 Analysis

This section examines three diagnostics that investigate where and why parsing performance differs across the four reasoning paths. All analyses use the CCGbank development set. Llama-3.1-8B and Qwen3-8B-Base exhibit similar diagnostic trends, while Olmo-3-1025-7B exhibits higher seed variance and several path-specific instabilities; our discussion follows the trends common to the first two, noting Olmo-specific behavior where relevant.

7.1 Parsing accuracy by sentence length

Figure 1 (top) shows labeled F1 on CCGbank Section 00 by sentence length. On all three base mod-

els, labeled F1 declines as sentences grow longer, but at different rates across paths. **LexTD degrades least with length:** its F1 on the longest sentences (40+) stays roughly 7-10 points above the next-best path on Llama-3.1-8B and Qwen3-8B-Base, and the gap widens as length increases. SymTD, in contrast, falls below the Direct and Two-step baselines at the longest length bin on Llama-3.1-8B and Qwen3-8B-Base, and on Olmo-3-1025-7B it sits substantially below every other path across the full length range, including the shortest sentences.

7.2 Where in generation do paths break down?

We measure **survival rate**: for each sentence, we compare the model’s predicted intermediate steps against the gold derivation step by step, similar in spirit to Baumann et al. (2011) and Köhn (2018) for incremental processors. A sentence *fails* at step t if the model’s t -th predicted step does not match the t -th gold step exactly, or if the model has produced fewer than t steps in total. The survival rate at step t is the fraction of sentences that have not yet failed. Figure 1 (bottom) plots survival rate against generation step, normalized to $[0, 1]$. We restrict this analysis to the four reasoning paths, which produce

Model	Path	No oracle	+ Oracle
Llama-3.1-8B	LexBU	84.90 $\pm$ 1.63	92.58 $\pm$ 0.83
	SymBU	85.32 $\pm$ 0.36	91.40 $\pm$ 0.84
Olmo-3-1025-7B	LexBU	79.59 $\pm$ 0.65	71.26 $\pm$ 9.01
	SymBU	78.68 $\pm$ 1.32	78.54 $\pm$ 3.19
Qwen3-8B-Base	LexBU	85.02 $\pm$ 0.45	92.86 $\pm$ 0.20
	SymBU	85.39 $\pm$ 0.98	91.28 $\pm$ 0.50

Table 5: Labeled F1 on the CCGbank development set for the two bottom-up reasoning paths, with and without oracle supertags. Mean $\pm$ standard deviation across three random seeds.

a sequence of intermediate derivation steps; Direct and Two-step do not.

LexTD maintains the highest survival rate throughout generation on all three base models. Its curve begins with a brief plateau (roughly the first 0.1-0.2 normalized steps); the early steps in this path are decisions about how the sentence splits into top-level constituents, which the model gets right on most sentences.

The two bottom-up paths follow different trajectories. LexBU starts low: its first step is a single bracketed state containing the lexical category for every token, so any incorrect category fails here, and the survival rate starts at roughly 0.55-0.57. SymBU emits one lexical assignment per step, so survival starts near 1.0 and declines steeply over the first 0.4 normalized steps, after which the decline slows visibly. This inflection appears on all three base models, and is close to the average position where bottom-up supertagging completes ($\sim$ 0.48). After supertagging, both paths produce invalid combination steps at a much lower rate, consistent with the “almost parsing” view of supertagging.

SymTD has a similar shape to LexTD on Llama-3.1-8B and Qwen3-8B-Base, but without the early plateau, and ends slightly below; on Olmo-3-1025-7B, its survival drops sharply throughout generation and ends well below the other three paths. We hypothesize that the combination of top-down generation with variable references, where constituents are referenced before their surface tokens appear, is more challenging for LLMs than other paths. We leave a closer investigation to future work.

7.3 Parsing with gold supertags

The previous diagnostic identified supertagging as a hard part of bottom-up generation. To examine what happens to the rest of the derivation once supertagging is given, we run an oracle experiment

on the bottom-up paths: at inference, we replace the supertagging portion of the model’s output with the gold lexical categories, then let it generate the remaining steps autoregressively. This is possible for bottom-up paths but not for top-down paths, where lexical categories appear at the leaves rather than at the start of the output.

Table 5 shows labeled F1 on the development set for LexBU and SymBU, with and without gold supertags. On Llama-3.1-8B and Qwen3-8B-Base, both paths gain 6-8 points from gold supertags, but LexBU gains more and ends higher in both cases, despite the two paths being roughly tied without them. The result suggests that with the same correct supertags, the **lexically grounded format builds derivations more reliably than the symbolic format**. On Olmo-3-1025-7B, LexBU with gold supertags shows a substantial drop in mean labeled F1 and a high standard deviation ($\pm$ 9.01), consistent with the unstable training pattern noted above.

8 Conclusion

We compared four reasoning paths for CCG parsing with LLMs. Only the top-down direction with lexically grounded steps consistently improved on a no-reasoning baseline. This runs counter to the bottom-up pipeline of most classical CCG parsers, suggesting that LLMs may benefit from a different parsing strategy. The same path outperformed a strong CCG parser when transferred to other languages, providing evidence that LLMs can extend CCG parsing to languages without CCGbanks.

Limitations

Our comparison covers four reasoning paths and three open-weight base models in the 7-8B parameter range. Smaller or larger models, instruction-tuned variants, and closed models other than GPT-5.4 are outside the scope of this work, and our findings on which reasoning path works best may not extend to those settings. We fine-tune on English CCGbank only, since English is the dominant language for CCG resources and the English portion is also the largest in PMB.

The four reasoning paths are not intended to cover every possible design for CCG derivation generation. The same holds for our task scope: we generate reasoning steps and the final derivation in a single autoregressive pass, and do not explore multi-turn setups in which intermediate steps are produced through separate model calls. The current

experimental setup already involves over 100 LLM fine-tuning runs, and adding another design axis would quickly become prohibitive.

We use a single LoRA configuration with the same hyperparameter settings across all models and reasoning paths. This keeps the comparison across paths uniform but leaves open how each path would behave under a more carefully tuned configuration. Constrained decoding is not part of our experimental design, though we report a preliminary experiment in Appendix F. We also do not measure inference time.

Ethical Considerations

We use existing public artifacts (CCGbank, PMB, open-weight models, EVALB) in accordance with their intended research use. We do not collect or annotate new data, and identify no specific risks arising from this work. We do not observe personally identifying information or offensive content requiring filtering in the treebanks.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Number JP24H00087. We thank Junjie Chen for helpful comments on an earlier draft of this paper. We also thank the anonymous reviewers for their constructive feedback. We acknowledge the use of AI tools for proofreading and coding assistance. All results are validated by the authors.

References

- Lasha Abzianidze, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos. 2017. [The Parallel Meaning Bank: Towards a multilingual corpus of translations annotated with compositional meaning representations](#). In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 242–247, Valencia, Spain. Association for Computational Linguistics.
- Michael Auli and Adam Lopez. 2011. [A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing](#). In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 470–480, Portland, Oregon, USA. Association for Computational Linguistics.
- Xuefeng Bai, Jialong Wu, Yulong Chen, Zhongqing Wang, Kehai Chen, Min Zhang, and Yue Zhang. 2025. Constituency parsing using LLMs. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*.
- Srinivas Bangalore and Aravind K. Joshi. 1999. [Supertagging: An approach to almost parsing](#). *Computational Linguistics*, 25(2):237–265.
- Timo Baumann, Okko Buß, and David Schlangen. 2011. [Evaluation and optimisation of incremental processors](#). *Dialogue & Discourse*, 2(1):113–141.
- Aditya Bhargava and Gerald Penn. 2020. [Supertagging with CCG primitives](#). In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, Online. Association for Computational Linguistics.
- Eugene Charniak. 2000. [A maximum-entropy-inspired parser](#). In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*.
- Stephen Clark. 2002. [Supertagging for Combinatory Categorical Grammar](#). In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 19–24, Università di Venezia. Association for Computational Linguistics.
- Stephen Clark. 2021. [Something old, something new: Grammar-based CCG parsing with transformer models](#). Technical Report arXiv:2109.10044, Cambridge Quantum Computing.
- Stephen Clark and James R. Curran. 2007. [Wide-coverage efficient statistical parsing with CCG and log-linear models](#). *Computational Linguistics*, 33(4):493–552.
- Michael Collins. 1997. [Three generative, lexicalised models for statistical parsing](#). In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain. Association for Computational Linguistics.
- Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. [Unsupervised cross-lingual representation learning at scale](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.
- Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2023. Compositional semantic parsing with large language models. In *The Eleventh International Conference on Learning Representations (ICLR)*.
- Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob

- Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, and 48 others. 2026. [Olmo 3](#). *Preprint*, arXiv:2512.13961.
- Ana Ezquerro, Carlos Gómez-Rodríguez, and David Vilares. 2025. [Better benchmarking LLMs for zero-shot dependency parsing](#). In *Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, pages 121–135, Tallinn, Estonia. University of Tartu Library.
- Hayate Funakura and Koji Mineshima. 2023. [Computational semantics and evaluation benchmark for interrogative sentences via Combinatory Categorical Grammar](#). In *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, pages 764–774, Hong Kong, China. Association for Computational Linguistics.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Julia Hockenmaier and Mark Steedman. 2007. [CCG-bank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank](#). *Computational Linguistics*, 33(3):355–396.
- Claudiu Daniel Hromei, Danilo Croce, and Roberto Basili. 2024. U-DepPLLaMA: Universal dependency parsing via auto-regressive large language models. *Italian Journal of Computational Linguistics*, 10(1):21–38.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations (ICLR)*.
- Nikita Kitaev and Dan Klein. 2018. [Constituency parsing with a self-attentive encoder](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.
- Konstantinos Kogkalidis and Michael Moortgat. 2022. [Geometry-aware supertagging with heterogeneous dynamic convolutions](#). *Preprint*, arXiv:2203.12235.
- Arne Köhn. 2018. [Incremental natural language processing: Challenges, strategies, and evaluation](#). In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2990–3003, Santa Fe, New Mexico, USA. Association for Computational Linguistics.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. 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 ’23*, pages 611–626, New York, NY, USA. Association for Computing Machinery.
- Mike Lewis and Mark Steedman. 2014. [A* CCG parsing with a supertag-factored model](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 990–1000, Doha, Qatar. Association for Computational Linguistics.
- 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*.
- Yufang Liu, Tao Ji, Yuanbin Wu, and Man Lan. 2021. [Generating CCG categories](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13443–13451.
- Hiroshi Matsuda, Chunpeng Ma, and Masayuki Asahara. 2025. [Step-by-step instructions and a simple tabular output format improve the dependency parsing accuracy of LLMs](#). In *Proceedings of the 18th International Conference on Parsing Technologies (IWPT, SyntaxFest 2025)*, pages 11–19, Ljubljana, Slovenia. Association for Computational Linguistics.
- Maxwell Nye, Anders Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.
- OpenAI. 2026. GPT-5.4 thinking system card. <https://openai.com/index/gpt-5-4-thinking-system-card/>.
- Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. [Supertagging the long tail with tree-structured decoding of complex categories](#). *Transactions of the Association for Computational Linguistics*, 9:243–260.
- Miloš Stanojević, Jonathan R. Brennan, Donald Dunagan, Mark Steedman, and John T. Hale. 2023. [Modeling structure-building in the brain with CCG parsing and large language models](#). *Cognitive Science*, 47(7):e13312.
- Miloš Stanojević and Mark Steedman. 2020. [Max-margin incremental CCG parsing](#). In *Proceedings*

- of the 58th Annual Meeting of the Association for Computational Linguistics, pages 4111–4122, Online. Association for Computational Linguistics.
- Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA.
- Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. [A minimal span-based neural constituency parser](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.
- Tomoki Sugimoto and Hitomi Yanaka. 2022. [Compositional semantics and inference system for temporal order based on Japanese CCG](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 104–114, Dublin, Ireland. Association for Computational Linguistics.
- Yuanhe Tian, Yan Song, and Fei Xia. 2020. [Supertagging Combinatory Categorical Grammar with attentive graph convolutional networks](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044, Online. Association for Computational Linguistics.
- Yuanhe Tian, Fei Xia, and Yan Song. 2024. [Large language models are no longer shallow parsers](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7131–7142, Bangkok, Thailand. Association for Computational Linguistics.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.
- Ryosuke Yamaki, Tadahiro Taniguchi, and Daichi Mochihashi. 2023. [Holographic CCG parsing](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 262–276, Toronto, Canada. Association for Computational Linguistics.
- 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 41 others. 2025. [Qwen3 technical report](#). Preprint, arXiv:2505.09388.
- Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. 2017. [A* CCG parsing with a supertag and dependency factored model](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 277–287, Vancouver, Canada. Association for Computational Linguistics.
- Yue Zhang and Stephen Clark. 2011. [Shift-reduce CCG parsing](#). In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 683–692, Portland, Oregon, USA. Association for Computational Linguistics.
- Ziyan Zhang, Yang Hou, Chen Gong, and Zhenghua Li. 2025. [Self-correction makes LLMs better parsers](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 6749–6762, Suzhou, China. Association for Computational Linguistics.
- Jinman Zhao and Gerald Penn. 2024. [LLM-supertagger: Categorical grammar supertagging via large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 697–705, Miami, Florida, USA. Association for Computational Linguistics.
- Yida Zhao, Hao Xue, Xiang Hu, and Kewei Tu. 2025. [A systematic study of compositional syntactic transformer language models](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7070–7083, Vienna, Austria. Association for Computational Linguistics.

A Treebank Statistics

Tables 6 and 7 show the number of sentences in each split of CCGbank and PMB 4.0.

Split	# Sentences
Train (Sections 02-21)	39604
Dev (Section 00)	1913
Test (Section 23)	2407

Table 6: Number of sentences in each split of CCGbank.

Split	EN		DE	IT	NL
	Before	After			
Train	8586	6724	1741	685	539
Dev	1174		559	540	438
Test	1053		547	461	491

Table 7: Number of sentences in each split of PMB 4.0 (gold-annotated portion). For the English training set, **Before** is the count prior to removing documents that appear in the development or test sets of other languages, and **After** is the count used in our experiments. Cleaning is applied only to the English training set.

Path	Llama-3.1-8B	Olmo-3-1025-7B	Qwen3-8B-Base
Direct	288.5	288.5	289.4
Two-step	470.7	470.7	472.4
LexTD	2668.9	2668.9	2681.8
LexBU	3933.2	3933.3	3946.2
SymTD	912.4	912.4	995.7
SymBU	912.4	912.4	995.7

Table 8: Mean number of output tokens per sentence on the CCGbank training set, under each base model’s tokenizer.

B Training Hyperparameters

LoRA is applied with rank 16, alpha 32, and dropout 0.05, targeting q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, and down_proj modules. We train for 10 epochs with a batch size of 8 and a learning rate of 3e-5, using the same configuration across all three models and applying early stopping on development-set loss.

C Computational Resources

All experiments were run on a mixture of NVIDIA A100 and H100 GPUs (single GPU per run). The full experimental setup, including LLM fine-tuning and inference, the Hol-CCG baseline, and additional diagnostic experiments, required approximately 3500 GPU-hours.

D Prompts

This section provides the full task description used for each variant. We use a different task description for each variant, since each one expects a different output format. The task description is followed by the input sentence in the form:

Parse the following sentence: '{sentence}'

D.1 Direct (no reasoning)

As an expert linguist, you are going to perform CCG (Combinatory Categorical Grammar) parsing. Given a sentence, output its CCG parse tree as a well-formed bracketed string on a single line, → using curly brackets as delimiters.
Do NOT explain your reasoning.
Output ONLY the final CCG parse, enclosed within <FINAL> and </FINAL> tags.
Example output schematic, where c1, c2, c3, etc., represent CCG categories, and w1, w2, w3, etc., → represent tokens:
<FINAL>
{c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 {c10 w5}}}
</FINAL>

D.2 Two-step (supertagging then parsing)

As an expert linguist, you are going to perform CCG (Combinatory Categorical Grammar) parsing. Given a sentence, first output its lexical supertags, then output the final CCG parse tree. Both must be well-formed bracketed strings on single lines, using curly brackets as delimiters.
Do NOT add any commentary or explanation.
Output the supertagging sequence enclosed within <SUPERTAGGING> and </SUPERTAGGING> tags, → immediately followed by the final CCG parse enclosed within <FINAL> and </FINAL> tags.
Example output schematic, where c1, c2, c3, etc., represent CCG categories, and w1, w2, w3, etc., → represent tokens:
<SUPERTAGGING>
{c4 w1} {c5 w2} {c8 w3} {c9 w4} {c7 w5}
</SUPERTAGGING>
<FINAL>
{c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 w5}}
</FINAL>

D.3 LexTD (lexically grounded top-down)

As an expert linguist, you are going to perform CCG (Combinatory Categorical Grammar) parsing. Given a sentence, construct its CCG parse tree step-by-step, in a top-down manner.
Output the intermediate states of the tree, showing partial structures as they are formed. Enclose → these within <STEPS> and </STEPS> tags.
Start with a single root category encompassing the entire sequence of tokens.
In each subsequent expansion step, apply all possible expansions to the current sequence of → non-terminal constituents, following CCG combinatory rules.
Do NOT proceed to deeper levels of expansion until all possible expansions at the current level are → completed.
Each expansion step must appear on a single line within its own <STEP> and </STEP> tags.
Conclude with the complete, fully expanded CCG parse tree, enclosed within <FINAL> and </FINAL> → tags.
Use curly brackets as delimiters.
Do NOT add any commentary or explanation.
Ensure the intermediate steps and the final parse tree are well-formed.
Example output schematic, where c1, c2, c3, etc., represent CCG categories, and w1, w2, w3, etc., → represent tokens:
<STEPS>
<STEP>{c1 w1 w2 w3 w4 w5}</STEP>
<STEP>{c1 {c2 w1 w2} {c3 w3 w4 w5}}</STEP>
<STEP>{c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 w3 w4} {c7 w5}}</STEP>
<STEP>{c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 {c10 w5}}}</STEP>
</STEPS>
<FINAL>
{c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 {c10 w5}}}
</FINAL>

D.4 LexBU (lexically grounded bottom-up)

As an expert linguist, you are going to perform CCG (Combinatory Categorical Grammar) parsing. Given a sentence, construct its CCG parse tree step-by-step, in a bottom-up manner.
Output the intermediate states of the tree, showing partial structures as they are formed. Enclose → these within <STEPS> and </STEPS> tags.
Start with the assignment of lexical categories to individual tokens.
In each subsequent derivation step, apply all possible CCG combinatory rules to the current sequence → of constituents.

Do NOT proceed to higher levels of derivation until all possible combinations at the current level
 → are completed.
 Each derivation step must appear on a single line within its own <STEP> and </STEP> tags.
 Conclude with the complete, fully derived CCG parse tree, enclosed within <FINAL> and </FINAL> tags.
 Use curly brackets as delimiters.
 Do NOT add any commentary or explanation.
 Ensure the intermediate steps and the final parse tree are well-formed.
 Example output schematic, where c1, c2, c3, etc., represent CCG categories, and w1, w2, w3, etc.,
 → represent tokens:
 <STEPS>
 <STEP>{c4 w1} {c5 w2} {c8 w3} {c9 w4} {c7 w5}</STEP>
 <STEP>{c2 {c4 w1} {c5 w2}} {c6 {c8 w3} {c9 w4}} {c7 w5}</STEP>
 <STEP>{c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 w5}}</STEP>
 <STEP>{c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 w5}}}</STEP>
 </STEPS>
 <FINAL>
 {c1 {c2 {c4 w1} {c5 w2}} {c3 {c6 {c8 w3} {c9 w4}} {c7 w5}}}
 </FINAL>

D.5 SymTD (symbolic top-down)

As an expert linguist, you are going to perform CCG (Combinatory Categorical Grammar) parsing.
 Given a sentence, construct its CCG parse tree step-by-step, in a top-down manner.
 Output the sequence of expansion steps used to build the tree. Enclose these within <STEPS> and
 → </STEPS> tags.
 Start with the root constituent represented by an abstract sequential variable (e.g., C1).
 In each expansion step, decompose the constituents at the current depth level into smaller
 → constituents following CCG combinatory rules.
 Assign a new, sequential variable (C2, C3, C4, etc.) to each newly created constituent as you expand
 → top-down, level-by-level, from left-to-right.
 Do NOT proceed to deeper levels of expansion until all possible expansions at the current level are
 → completed.
 Each expansion step must appear on a single line.
 Conclude with the complete, fully expanded CCG parse tree, enclosed within <FINAL> and </FINAL>
 → tags.
 Use curly brackets as delimiters for the final tree.
 Do NOT add any commentary or explanation.
 Ensure the expansion steps logically map to the final parse tree and are syntactically well-formed.
 Example output schematic, where c1, c2, c3, etc., represent CCG categories, w1, w2, w3, etc.,
 → represent tokens, and C1, C2, C3, etc., represent abstract constituent variables:
 <STEPS>
 C1 := combine(C2, C3) : c6
 C2 := combine(C4, C5) : c4
 C3 := unary(C6) : c5
 C4 := w1 : c1
 C5 := w2 : c2
 C6 := w3 : c3
 </STEPS>
 <FINAL>
 {c6 {c4 {c1 w1} {c2 w2}} {c5 {c3 w3}}}
 </FINAL>

D.6 SymBU (symbolic bottom-up)

As an expert linguist, you are going to perform CCG (Combinatory Categorical Grammar) parsing.
 Given a sentence, construct its CCG parse tree step-by-step, in a bottom-up manner.
 Output the sequence of derivation steps used to build the tree. Enclose these within <STEPS> and
 → </STEPS> tags.
 Start by assigning lexical categories to the individual tokens, representing each constituent with
 → an abstract sequential variable (e.g., C1, C2).
 In each derivation step, apply CCG rules to combine adjacent constituents into larger constituents,
 → assigning a new sequential variable to each result.
 Do NOT proceed to higher levels of derivation until all possible combinations at the current level
 → are completed.
 Each derivation step must appear on a single line.
 Conclude with the complete, fully derived CCG parse tree, enclosed within <FINAL> and </FINAL> tags.
 Use curly brackets as delimiters for the final tree.
 Do NOT add any commentary or explanation.
 Ensure the derivation steps logically map to the final parse tree and are syntactically well-formed.

Example output schematic, where c_1, c_2, c_3 , etc., represent CCG categories, w_1, w_2, w_3 , etc.,
 $\Rightarrow$ represent tokens, and C_1, C_2, C_3 , etc., represent abstract constituent variables:

```
<STEPS>
C1 := w1 : c1
C2 := w2 : c2
C3 := w3 : c3
C4 := combine(C1, C2) : c4
C5 := unary(C3) : c5
C6 := combine(C4, C5) : c6
</STEPS>
<FINAL>
{c6 {c4 {c1 w1} {c2 w2}} {c5 {c3 w3}}}
```

</FINAL>

E Full Results on CCGbank and PMB 4.0

Model	Variant	SV	Cat	Dep LF
Yoshikawa et al. (2017)			–	88.80
Stanojević and Steedman (2020)			–	90.60
Tian et al. (2020)			96.25	90.58
Bhargava and Penn (2020)			96.00	90.90
Liu et al. (2021)			96.05	90.87
Prange et al. (2021)			96.22	90.91
Kogkalidis and Moortgat (2022)			96.29	–
Clark (2021)			–	92.90
Yamaki et al. (2023)			96.60	92.12
Zhao and Penn (2024)			96.64	92.05
Llama-3.1-8B	Direct	96.08 ± 0.69	90.02 ± 0.85	85.66 ± 0.18
	Two-step	96.20 ± 2.06	90.00 ± 2.48	84.65 ± 1.54
	LexTD	99.71 ± 0.08	95.48 ± 0.22	89.48 ± 0.56
	LexBU	97.87 ± 1.16	92.60 ± 1.50	85.31 ± 1.44
	SymTD	98.24 ± 0.79	89.61 ± 2.19	84.15 ± 1.43
	SymBU	98.95 ± 0.35	93.73 ± 0.48	86.48 ± 0.80
Olmo-3-7B	Direct	82.33 ± 6.24	73.48 ± 6.42	75.15 ± 3.98
	Two-step	91.44 ± 1.30	83.36 ± 1.88	80.40 ± 1.00
	LexTD	94.97 ± 2.43	88.82 ± 1.87	85.44 ± 0.79
	LexBU	92.80 ± 1.94	85.12 ± 2.29	79.77 ± 0.80
	SymTD	74.02 ± 13.43	65.92 ± 8.41	67.67 ± 5.90
	SymBU	92.11 ± 1.44	83.16 ± 1.98	78.55 ± 1.09
Qwen3-8B	Direct	96.48 ± 0.83	89.75 ± 1.03	84.54 ± 0.29
	Two-step	96.12 ± 0.74	90.28 ± 1.12	84.84 ± 1.15
	LexTD	99.56 ± 0.11	94.90 ± 0.36	88.43 ± 0.68
	LexBU	98.73 ± 0.35	93.50 ± 0.67	85.61 ± 0.64
	SymTD	98.49 ± 0.85	89.35 ± 1.94	83.68 ± 1.62
	SymBU	98.83 ± 0.49	93.36 ± 1.03	85.63 ± 0.80
Llama-3.1-8B (Direct, 5-shot)		3.61 ± 0.05	0.77 ± 0.01	0.03 ± 0.01
Olmo-3-7B (Direct, 5-shot)		4.74 ± 0.13	0.82 ± 0.02	0.02 ± 0.01
Qwen3-8B (Direct, 5-shot)		8.10 ± 0.04	1.52 ± 0.02	0.06 ± 0.01
GPT-5.4 (none, 5-shot)		11.59	4.60	1.31
GPT-5.4 (high, 5-shot)		71.04	51.98	39.81

Table 9: Full results on CCGbank Section 23 with standard deviations across three random seeds. Same data as Table 2 in the main paper, with standard deviations included for each LLM-based cell.

Model	Variant	Labeled Bracketed F1				Supertagging accuracy			
		EN	DE	IT	NL	EN	DE	IT	NL
Llama-3.1-8B	Direct	89.26 ± 0.09	64.74 ± 1.50	68.70 ± 1.24	66.82 ± 1.80	90.87 ± 0.49	69.99 ± 3.52	75.00 ± 1.09	72.62 ± 3.70
	Two-step	89.02 ± 0.44	67.75 ± 0.61	68.36 ± 1.24	68.96 ± 0.23	91.96 ± 0.30	77.50 ± 1.02	75.74 ± 0.66	78.73 ± 1.54
	LexTD	90.36 ± 0.16	69.50 ± 0.64	72.86 ± 0.10	71.92 ± 0.76	93.48 ± 0.56	78.55 ± 0.72	83.94 ± 0.65	82.16 ± 0.64
	LexBU	88.47 ± 0.78	68.18 ± 0.68	67.97 ± 0.28	68.91 ± 0.79	92.16 ± 1.25	78.33 ± 1.07	78.22 ± 0.22	79.87 ± 1.31
	SymTD	88.47 ± 0.55	62.28 ± 2.39	66.18 ± 2.79	62.23 ± 2.74	90.10 ± 1.16	65.03 ± 4.32	71.59 ± 5.18	63.85 ± 4.25
	SymBU	89.46 ± 0.77	68.88 ± 0.91	70.10 ± 0.67	70.16 ± 0.46	92.61 ± 0.91	81.91 ± 1.78	85.11 ± 0.65	83.65 ± 0.13
Olmo-3-7B	Direct	83.61 ± 2.99	51.50 ± 1.35	54.47 ± 2.51	53.28 ± 2.46	82.64 ± 4.73	46.15 ± 2.67	50.27 ± 4.71	48.10 ± 4.84
	Two-step	83.32 ± 0.94	58.81 ± 1.22	61.54 ± 1.38	57.82 ± 0.81	83.47 ± 1.80	60.93 ± 3.18	64.83 ± 0.98	57.08 ± 2.95
	LexTD	88.23 ± 0.26	62.46 ± 1.01	67.00 ± 0.19	63.91 ± 0.56	89.75 ± 0.82	65.38 ± 1.76	71.54 ± 0.22	65.87 ± 1.16
	LexBU	87.28 ± 0.50	61.31 ± 1.82	64.42 ± 1.17	60.11 ± 0.44	90.33 ± 0.55	67.12 ± 2.55	68.37 ± 1.47	63.21 ± 1.48
	SymTD	77.95 ± 2.39	51.70 ± 1.76	56.88 ± 1.09	51.47 ± 0.26	74.18 ± 3.56	49.44 ± 2.77	54.98 ± 1.40	48.05 ± 1.38
	SymBU	78.82 ± 3.35	59.48 ± 3.88	65.15 ± 2.89	57.78 ± 4.10	76.20 ± 5.59	64.90 ± 7.15	71.17 ± 4.59	59.89 ± 7.29
Qwen3-8B	Direct	87.28 ± 1.85	65.34 ± 1.60	66.10 ± 1.15	65.50 ± 1.18	88.17 ± 3.03	71.04 ± 2.32	69.29 ± 1.87	72.51 ± 1.13
	Two-step	89.61 ± 0.49	68.15 ± 0.31	69.71 ± 1.25	66.82 ± 1.24	91.98 ± 0.81	77.84 ± 0.90	78.17 ± 1.81	75.18 ± 2.38
	LexTD	90.10 ± 0.68	71.35 ± 0.31	73.93 ± 0.12	72.79 ± 0.46	92.80 ± 1.51	80.69 ± 1.11	85.69 ± 0.10	83.32 ± 1.37
	LexBU	88.22 ± 0.54	65.36 ± 0.53	69.05 ± 1.13	66.01 ± 0.58	92.08 ± 0.38	75.74 ± 0.59	77.18 ± 3.11	75.12 ± 0.26
	SymTD	88.24 ± 0.12	66.23 ± 1.98	68.83 ± 2.61	66.22 ± 1.31	89.88 ± 0.15	71.20 ± 2.62	73.15 ± 2.75	70.60 ± 2.49
	SymBU	89.40 ± 0.22	69.43 ± 1.29	71.59 ± 0.52	71.78 ± 0.38	92.99 ± 0.99	82.88 ± 1.16	85.74 ± 1.00	83.92 ± 0.12
GPT-5.4 (none, 5-shot)		65.74	61.56	62.74	59.01	61.51	59.20	63.05	55.71
GPT-5.4 (high, 5-shot)		82.36	72.76	74.76	73.35	79.65	79.73	86.10	80.26
Hol-CCG (XLM-R-large)		90.81 ± 0.54	67.36 ± 1.14	69.51 ± 0.88	71.14 ± 0.53	95.10 ± 0.24	74.50 ± 2.07	79.12 ± 1.73	78.94 ± 0.81

Table 10: Full results on the PMB test set with standard deviations across three random seeds for fine-tuned LLMs and ten random seeds for Hol-CCG. Same data as Table 3 in the main paper, with standard deviations included for each LLM-based cell.

Model	Variant	Structural Validity (SV)			
		EN	DE	IT	NL
Llama-3.1-8B	Direct	99.78 $\pm$ 0.15	97.80 $\pm$ 0.48	98.12 $\pm$ 0.70	97.90 $\pm$ 0.65
	Two-step	99.18 $\pm$ 0.24	95.37 $\pm$ 2.56	98.26 $\pm$ 0.44	97.22 $\pm$ 0.62
	LexTD	99.75 $\pm$ 0.15	96.89 $\pm$ 0.63	99.20 $\pm$ 0.13	97.83 $\pm$ 0.31
	LexBU	99.21 $\pm$ 0.11	97.20 $\pm$ 0.58	99.13 $\pm$ 0.38	98.37 $\pm$ 0.35
	SymTD	99.47 $\pm$ 0.48	98.41 $\pm$ 0.94	96.89 $\pm$ 1.19	97.01 $\pm$ 1.89
	SymBU	99.68 $\pm$ 0.22	99.51 $\pm$ 0.56	99.93 $\pm$ 0.13	99.39 $\pm$ 0.00
Olmo-3-7B	Direct	95.09 $\pm$ 4.16	90.80 $\pm$ 3.97	91.90 $\pm$ 4.81	91.86 $\pm$ 5.70
	Two-step	97.22 $\pm$ 0.52	92.50 $\pm$ 0.80	95.30 $\pm$ 0.98	93.49 $\pm$ 1.67
	LexTD	98.07 $\pm$ 1.13	94.15 $\pm$ 2.19	97.61 $\pm$ 0.44	94.84 $\pm$ 0.47
	LexBU	98.96 $\pm$ 0.19	97.44 $\pm$ 1.14	97.11 $\pm$ 1.07	97.49 $\pm$ 0.31
	SymTD	92.21 $\pm$ 2.00	90.61 $\pm$ 1.85	92.91 $\pm$ 3.08	90.22 $\pm$ 0.54
	SymBU	92.08 $\pm$ 4.90	89.76 $\pm$ 6.85	94.00 $\pm$ 4.23	87.64 $\pm$ 7.66
Qwen3-8B	Direct	98.89 $\pm$ 0.70	96.83 $\pm$ 0.87	96.17 $\pm$ 1.96	97.76 $\pm$ 1.08
	Two-step	99.46 $\pm$ 0.33	98.54 $\pm$ 0.49	98.63 $\pm$ 0.13	98.10 $\pm$ 1.39
	LexTD	99.68 $\pm$ 0.11	99.27 $\pm$ 0.66	99.50 $\pm$ 0.13	98.64 $\pm$ 0.63
	LexBU	99.59 $\pm$ 0.15	98.30 $\pm$ 1.53	98.84 $\pm$ 0.67	98.23 $\pm$ 0.96
	SymTD	99.56 $\pm$ 0.22	98.96 $\pm$ 0.74	98.92 $\pm$ 0.66	98.92 $\pm$ 0.65
	SymBU	99.31 $\pm$ 0.63	98.84 $\pm$ 0.76	99.42 $\pm$ 0.45	99.32 $\pm$ 0.66

Table 11: Structural validity rate on the PMB test set, evaluated zero-shot on German (DE), Italian (IT), and Dutch (NL) after training on English (EN) only. Mean $\pm$ standard deviation across three random seeds.

Model	Variant	Application Validity (AppV)			
		EN	DE	IT	NL
Llama-3.1-8B	Direct	99.49 $\pm$ 0.08	97.74 $\pm$ 0.16	98.90 $\pm$ 0.16	97.69 $\pm$ 0.15
	Two-step	98.35 $\pm$ 0.30	93.45 $\pm$ 0.67	94.98 $\pm$ 0.55	92.81 $\pm$ 0.90
	LexTD	99.70 $\pm$ 0.15	98.56 $\pm$ 0.64	98.75 $\pm$ 0.33	98.67 $\pm$ 0.51
	LexBU	98.30 $\pm$ 0.40	92.38 $\pm$ 0.47	93.88 $\pm$ 0.34	92.13 $\pm$ 0.27
	SymTD	99.19 $\pm$ 0.12	98.01 $\pm$ 0.24	98.16 $\pm$ 0.24	97.95 $\pm$ 0.40
	SymBU	98.53 $\pm$ 0.21	93.42 $\pm$ 0.26	96.05 $\pm$ 0.88	93.03 $\pm$ 0.99
Olmo-3-7B	Direct	98.98 $\pm$ 0.28	95.96 $\pm$ 0.91	97.26 $\pm$ 0.36	96.62 $\pm$ 0.51
	Two-step	97.49 $\pm$ 0.34	89.80 $\pm$ 1.74	90.08 $\pm$ 0.61	89.86 $\pm$ 1.15
	LexTD	99.41 $\pm$ 0.15	96.62 $\pm$ 0.43	97.47 $\pm$ 0.70	97.01 $\pm$ 0.47
	LexBU	97.30 $\pm$ 0.10	88.16 $\pm$ 0.18	87.93 $\pm$ 0.51	86.76 $\pm$ 0.92
	SymTD	98.05 $\pm$ 0.08	94.73 $\pm$ 0.87	95.14 $\pm$ 1.21	94.64 $\pm$ 0.54
	SymBU	96.81 $\pm$ 0.35	92.14 $\pm$ 0.19	94.37 $\pm$ 0.88	90.83 $\pm$ 0.47
Qwen3-8B	Direct	99.16 $\pm$ 0.26	96.21 $\pm$ 0.97	98.13 $\pm$ 0.42	97.51 $\pm$ 0.69
	Two-step	98.52 $\pm$ 0.09	92.51 $\pm$ 0.27	94.48 $\pm$ 0.68	92.52 $\pm$ 1.05
	LexTD	99.55 $\pm$ 0.16	98.22 $\pm$ 0.35	98.94 $\pm$ 0.55	98.62 $\pm$ 0.26
	LexBU	97.84 $\pm$ 0.26	90.44 $\pm$ 0.49	91.82 $\pm$ 1.91	91.54 $\pm$ 0.19
	SymTD	99.46 $\pm$ 0.13	98.69 $\pm$ 0.19	98.46 $\pm$ 0.46	98.09 $\pm$ 0.45
	SymBU	98.46 $\pm$ 0.36	94.73 $\pm$ 0.49	96.82 $\pm$ 0.32	94.15 $\pm$ 0.44

Table 12: Application validity (AppV) on the PMB test set, evaluated zero-shot on German (DE), Italian (IT), and Dutch (NL) after training on English (EN) only. Mean $\pm$ standard deviation across three random seeds.

Model	Variant	Derivation Validity (DerivV)			
		EN	DE	IT	NL
Llama-3.1-8B	Direct	97.37 $\pm$ 0.34	88.54 $\pm$ 1.12	94.91 $\pm$ 0.61	89.59 $\pm$ 1.21
	Two-step	93.33 $\pm$ 0.95	72.22 $\pm$ 2.77	79.18 $\pm$ 1.17	71.50 $\pm$ 1.76
	LexTD	98.51 $\pm$ 0.92	93.01 $\pm$ 3.15	94.97 $\pm$ 1.33	93.75 $\pm$ 3.11
	LexBU	92.75 $\pm$ 1.96	69.72 $\pm$ 1.96	75.13 $\pm$ 0.57	68.95 $\pm$ 0.87
	SymTD	96.60 $\pm$ 0.31	90.52 $\pm$ 1.85	91.79 $\pm$ 0.49	89.89 $\pm$ 2.77
	SymBU	94.44 $\pm$ 0.63	77.04 $\pm$ 0.87	83.79 $\pm$ 2.16	77.26 $\pm$ 2.52
Olmo-3-7B	Direct	95.59 $\pm$ 0.64	81.74 $\pm$ 2.54	88.61 $\pm$ 0.90	84.93 $\pm$ 1.67
	Two-step	89.74 $\pm$ 1.08	60.29 $\pm$ 5.16	59.33 $\pm$ 2.48	57.97 $\pm$ 4.96
	LexTD	97.46 $\pm$ 0.43	85.29 $\pm$ 3.07	90.16 $\pm$ 2.46	87.41 $\pm$ 1.75
	LexBU	89.44 $\pm$ 0.12	55.72 $\pm$ 0.53	53.38 $\pm$ 1.93	50.98 $\pm$ 0.90
	SymTD	91.85 $\pm$ 0.91	76.70 $\pm$ 5.58	79.92 $\pm$ 4.10	77.65 $\pm$ 0.71
	SymBU	90.00 $\pm$ 1.11	73.27 $\pm$ 1.33	78.42 $\pm$ 1.82	70.75 $\pm$ 2.07
Qwen3-8B	Direct	95.93 $\pm$ 1.50	83.93 $\pm$ 3.57	91.68 $\pm$ 1.76	87.46 $\pm$ 3.32
	Two-step	93.67 $\pm$ 0.20	69.76 $\pm$ 1.47	76.32 $\pm$ 2.19	68.56 $\pm$ 3.14
	LexTD	97.65 $\pm$ 0.81	91.66 $\pm$ 1.98	94.98 $\pm$ 2.46	94.02 $\pm$ 1.23
	LexBU	91.58 $\pm$ 1.22	62.29 $\pm$ 4.53	69.78 $\pm$ 5.55	64.63 $\pm$ 2.14
	SymTD	97.65 $\pm$ 0.57	93.66 $\pm$ 0.77	92.77 $\pm$ 2.21	91.36 $\pm$ 1.98
	SymBU	94.04 $\pm$ 0.87	81.08 $\pm$ 1.90	88.94 $\pm$ 1.08	77.93 $\pm$ 0.88

Table 13: Derivation validity (DerivV) on the PMB test set, evaluated zero-shot on German (DE), Italian (IT), and Dutch (NL) after training on English (EN) only. Mean $\pm$ standard deviation across three random seeds.

F Constrained Decoding

This section reports a preliminary experiment with constrained decoding (CD). The purpose is to investigate whether the differences in parsing performance between reasoning paths depend on the models' ability to produce structurally valid output. Under CD, a formal grammar restricts which tokens the model may generate at each step, so any completed output belongs to the language the grammar defines. We design one grammar per output format, written in GBNF, a Backus-Naur-form grammar format,⁶ and apply it during generation through vLLM's structured-output support (Kwon et al., 2023). The results below therefore show what this particular set of grammars achieves; a more systematic study is left to future work. All other experimental settings follow Section 5.4. We evaluate on the CCGbank development set.

F.1 Grammars

All six grammars share the following rules, which define CCG categories and tokens:

```
cat      ::= func | atom | special
func     ::= arg slash arg
arg      ::= atom | "(" func ")"
atom     ::= ("NP" | "N" | "PP" | "PR" | "S" | "conj") feat?
feat     ::= "[" [a-z]+ "]"
slash    ::= "/" | "\"
special  ::= "CONJ" | "LRB" | "RRB" | "LQU" | "RQU" | "." | "," | ";" | ":" | "?" | "!"
WORD     ::= [^{} \t\n\r]+
```

The remaining rules differ by output format.

Direct (no reasoning)

```
root     ::= "<FINAL>\n" tree "\n</FINAL>"
tree     ::= leaf | tunary | tbinary
leaf     ::= "{" cat " " WORD "}"
tunary   ::= "{" cat " " tree "}"
tbinary  ::= "{" cat " " tree " " tree "}"
```

Two-step (supertagging then parsing)

```
root     ::= "<SUPERTAGGING>\n" supertags "\n</SUPERTAGGING>\n<FINAL>\n" tree "\n</FINAL>"
supertags ::= leaf (" " leaf)*
tree     ::= leaf | tunary | tbinary
leaf     ::= "{" cat " " WORD "}"
tunary   ::= "{" cat " " tree "}"
tbinary  ::= "{" cat " " tree " " tree "}"
```

LexTD (lexically grounded top-down)

```
root     ::= "<STEPS>\n" step+ "</STEPS>\n<FINAL>\n" ftree "\n</FINAL>"
step     ::= "<STEP>" ptree "</STEP>\n"
ptree    ::= "{" cat " " pbody "}"
pbody    ::= span | pkids
pkids    ::= ptree (" " ptree)?
span     ::= WORD (" " WORD)*
ftree    ::= fleaf | funary | fbinary
fleaf    ::= "{" cat " " WORD "}"
funary   ::= "{" cat " " ftree "}"
fbinary  ::= "{" cat " " ftree " " ftree "}"
```

⁶<https://github.com/ggml-org/llama.cpp/blob/master/grammars/README.md>

LexBU (lexically grounded bottom-up)

```
root      ::= "<STEPS>\n" step+ "</STEPS>\n<FINAL>\n" tree "\n</FINAL>"
step      ::= "<STEP>" forest "</STEP>\n"
forest    ::= tree (" " tree)*
tree      ::= leaf | unary | binary
leaf      ::= "{" cat " " WORD "}"
unary     ::= "{" cat " " tree "}"
binary    ::= "{" cat " " tree " " tree "}"
```

SymTD (symbolic top-down)

```
root      ::= "<STEPS>\n" action+ "</STEPS>\n<FINAL>\n" tree "\n</FINAL>"
action    ::= CELL " := " operation " : " cat "\n"
operation ::= combine | unary | LEXWORD
combine   ::= "combine(" CELL " , " CELL ")"
unary     ::= "unary(" CELL ")"
tree      ::= leaf | tunary | tbinary
leaf      ::= "{" cat " " WORD "}"
tunary    ::= "{" cat " " tree "}"
tbinary   ::= "{" cat " " tree " " tree "}"
CELL      ::= "C" [1-9] [0-9]*
LEXWORD   ::= [^{}]() \t\n\r|+
```

SymBU (symbolic bottom-up)

```
root      ::= "<STEPS>\n" lex_action+ op_action* "</STEPS>\n<FINAL>\n" tree "\n</FINAL>"
lex_action ::= CELL " := " LEXWORD " : " cat "\n"
op_action ::= CELL " := " op " : " cat "\n"
op         ::= combine | unary
combine    ::= "combine(" CELL " , " CELL ")"
unary      ::= "unary(" CELL ")"
tree       ::= leaf | tunary | tbinary
leaf       ::= "{" cat " " WORD "}"
tunary     ::= "{" cat " " tree "}"
tbinary    ::= "{" cat " " tree " " tree "}"
CELL       ::= "C" [1-9] [0-9]*
LEXWORD    ::= [^{}]() \t\n\r|+
```

F.2 Results

Table 14 shows structural validity and labeled dependency F1 on the CCGbank development set, with and without constrained decoding. We make two observations.

- The relative performance of the reasoning paths is largely preserved under CD: LexTD remains ahead of the others on all three models. This holds even on Olmo-3-1025-7B, where structural validity without CD is lowest and rises to near 100% for most paths. This suggests that the differences between reasoning paths are not explained by differences in structural validity.
- The gains from CD are inversely related to how often a path already produced structurally valid output. LexTD, whose structural validity is above 99% on Llama-3.1-8B and Qwen3-8B-Base without CD, gains 0.03 and 0.00 F1, while Direct and SymTD on Olmo-3-1025-7B, with the lowest validity rates, gain 8.60 and 12.40. CD also reduces seed variance where validity was unstable: the standard deviation of SymTD’s F1 on Olmo-3-1025-7B falls from 6.28 to 0.76.

Note that it is possible for structural validity to not reach 100% under CD, since generation can be truncated at the token limit before the derivation is complete; this affects LexBU most, the format with the longest outputs (Table 8).

Model	Variant	SV		Dep LF		Δ
		w/o CD	w/ CD	w/o CD	w/ CD	
Llama-3.1-8B	Direct	96.03 $\pm$ 1.28	99.60 $\pm$ 0.22	85.49 $\pm$ 0.86	86.83 $\pm$ 0.32	+1.34
	Two-step	96.52 $\pm$ 1.44	99.55 $\pm$ 0.14	84.70 $\pm$ 0.89	85.66 $\pm$ 0.25	+0.96
	LexTD	99.55 $\pm$ 0.16	99.76 $\pm$ 0.03	89.44 $\pm$ 0.63	89.47 $\pm$ 0.55	+0.03
	LexBU	97.46 $\pm$ 1.47	97.82 $\pm$ 1.45	84.83 $\pm$ 1.77	84.89 $\pm$ 1.75	+0.06
	SymTD	97.98 $\pm$ 0.72	99.62 $\pm$ 0.24	83.54 $\pm$ 1.81	84.37 $\pm$ 1.67	+0.83
	SymBU	98.59 $\pm$ 0.19	99.79 $\pm$ 0.09	85.32 $\pm$ 0.36	85.54 $\pm$ 0.53	+0.22
Olmo-3-7B	Direct	83.01 $\pm$ 4.67	99.62 $\pm$ 0.11	75.91 $\pm$ 3.00	84.51 $\pm$ 0.28	+8.60
	Two-step	92.11 $\pm$ 1.22	99.79 $\pm$ 0.00	80.34 $\pm$ 1.15	82.17 $\pm$ 0.60	+1.83
	LexTD	92.84 $\pm$ 4.54	99.55 $\pm$ 0.29	84.17 $\pm$ 2.08	88.14 $\pm$ 0.48	+3.97
	LexBU	92.48 $\pm$ 0.93	96.24 $\pm$ 0.57	79.24 $\pm$ 0.89	81.80 $\pm$ 0.94	+2.56
	SymTD	73.64 $\pm$ 13.90	99.51 $\pm$ 0.16	66.68 $\pm$ 6.28	79.08 $\pm$ 0.76	+12.40
	SymBU	92.14 $\pm$ 1.02	99.77 $\pm$ 0.06	78.68 $\pm$ 1.32	82.43 $\pm$ 1.04	+3.75
Qwen3-8B	Direct	96.58 $\pm$ 0.97	99.25 $\pm$ 0.47	84.47 $\pm$ 0.58	85.20 $\pm$ 0.38	+0.73
	Two-step	96.22 $\pm$ 0.96	99.50 $\pm$ 0.26	84.46 $\pm$ 0.69	85.26 $\pm$ 0.75	+0.80
	LexTD	99.27 $\pm$ 0.19	99.58 $\pm$ 0.05	88.22 $\pm$ 0.82	88.22 $\pm$ 0.72	+0.00
	LexBU	98.76 $\pm$ 0.11	98.81 $\pm$ 0.16	85.02 $\pm$ 0.45	85.09 $\pm$ 0.41	+0.07
	SymTD	98.22 $\pm$ 0.97	99.65 $\pm$ 0.13	82.75 $\pm$ 2.06	83.13 $\pm$ 1.51	+0.38
	SymBU	98.38 $\pm$ 0.53	99.83 $\pm$ 0.11	85.39 $\pm$ 0.98	85.67 $\pm$ 0.95	+0.28

Table 14: Structural validity (SV) and labeled dependency F1 on the CCGbank development set, with and without constrained decoding (CD), across three random seeds. Δ is the change in Dep LF from CD.