Translating Tax Law to Code with LLMs: A Benchmark and Evaluation Framework

Anonymous ACL submission

Abstract

Catala is a domain-specific programming language for tax law, meant to facilitate the translation of legal text into executable computer code, thanks to a syntax close to that of legal language and reasoning. Legal statutes paired with their Catala translation have been published online periodically, but manual translation remains labor-intensive. In this work, we develop a benchmark for the evaluation of Catala code generation from legal text, including a training set to fine-tune Large Language Models. To assess the quality of the generated code, we introduce an evaluation framework extending current metrics for code generation. Our experiments with few-shot learning, as well as fine-tuned models, suggest the feasibility of automating legal code generation, and contrast with prior attempts to translate legal language into a formal representation.[1]

1 Introduction

Since the 1990s, the French tax administration has maintained an expert system to calculate taxes and social benefits. This expert system must be periodically updated to follow the evolution of tax law, a process hampered by the limitations of the current programming paradigm. The Catala programming language (Merigoux et al., 2021) was designed to address these limitations: first, by providing a domain-specific language better aligned with the syntax of legal language and reasoning, and second, by encouraging collaboration between lawyers and computer scientists using pair programming. A considerable amount of Catala code has already been written and published on GitHub (Merigoux, 2023).

How to translate legal language into executable computer code is an open research question (Servantez et al., 2023; Garzo and Palumbo, 2025; Zitouni et al., 2024), which can be traced back to initial efforts at representing parts of legislation with tools from expert systems (McCarty, 1976; Sergot et al., 1986). It is also of practical significance, as many tax agencies across the world face the problem of computing tax amounts, with varying obligations (Lawsky, 2020). A significant challenge is the substantial human effort required for translation: each section of tax law takes hours to convert into code, the volume of existing laws is immense — e.g. the French tax code spans approximately 3,500 pages — and frequent amendments necessitate continuous updates and translations. In addition, the structure of laws is not strictly linear. For instance, some sections modify or override provisions stated in earlier parts. This requires careful management of dependencies between provisions to ensure a consistent and faithful implementation

```
{
  "input": "4 A compter du 1er janvier 2022,
      pour l'application du 5 de l'article D.
      823-17 (...) pas celui des AL.",

  "metadata": "declaration champ d'application
      CalculAidePersonnaliseeLogementLocatif:
      entree loyer_principal contenu argent
      (...)-- Mayotte",

  "output": "champ d'application
      CalculAidePersonnaliseeLogementLocatif
      sous condition date_courante >=
      |2023-01-01| et date_courante <
      |2023-10-01|: exception metropole (...)8
      181 EUR",

  "generated_output": "champ d'application
      CalculAidePersonnaliseeLogementLocatif
      sous condition date_courante >=
      |2023-01-01| et date_courante <=
      |2023-12-31|: exception metropole (...)8
      181 EUR"
}
```

Figure 1: Extracts of one sample from our dataset, with its input, metadata and reference output. We also show an output generated by Qwen2.5-Coder-32B-Instruct.

[1]The dataset is available at `anonymized_url`

of the legal text.

This law-to-code translation task is related to that of semantic parsing of legal language (Pertierra et al., 2017; Morgenstern, 2014; Sinh and Nguyen, 2018). So far, results have been mostly negative, for two main reasons. There is a stark contrast between the language semantic parsers are made for, and legal language. Further, there is no large collection of legal text annotated for semantic parsing. Strictly speaking, Catala code is not a semantic representation of legal language, because it commits to one interpretation. But it trades the ability to represent multiple interpretations for the ability to perform legal reasoning. We report results on par with code generation for other programming languages, making this a positive result in semantic parsing for legal language.

Our main contributions are:

- Starting from the existing Catala code corpus, we created a new dataset suited for the fine-tuning of Large Language Models (LLMs).
- We adapted existing evaluation metrics to assess the accuracy of the outputs produced by our fine-tuned models.
- We benchmark state-of-the-art LLMs, with few-shot learning and fine-tuning.

2 Related work

Meaning representations Semantic parsing aims at faithfully representing the meaning of language and is a long-standing NLP task — see for example (Blackburn and Bos, 2005) for a comprehensive review. First-order logic is sufficient to model legal reasoning, as long as humans provide values for ambiguous or vague predicates, as was done in (Sergot et al., 1986). But formalisms for semantic parsing generally aim for close syntactic alignment between input and output, as can be found in Abstract Meaning Representation (Banarescu et al., 2013) and Universal Decompositional Semantics (White et al., 2020). Semantic parsing of legal language has been shown to be a major challenge (Morgenstern, 2014; Pertierra et al., 2017; Sinh and Nguyen, 2018). In particular, sentence length and logical connectives are a problem (Allen and Engholm, 1977). Alignment between legal language and formal representation is hard to achieve, even if some formalisms achieve moderate correspondence.

Legal expert systems While first-order logic frameworks such as Prolog are sufficient to represent the logic of laws and regulations, legal language has a specific way of expressing logic, for instance through defeasible logic (Nute, 1988). This has prompted the creation of semantic formalisms to represent legal rules. Proleg (Satoh, 2023) is an extension of Prolog designed to represent Japanese law. In particular, it has been augmented with a feature to visualize reasoning traces, to identify bugs in the formalization or issues in a legal text (Fungwacharakorn and Satoh, 2022). There have been attempts to generate Proleg from legal language, with promising results on narrow scopes (Zin et al., 2023, 2024). OpenFisca is a software package aimed at representing financial law. So far, it has been developed and published open-source,[2] and has been used to model specific aspects of law in scientific publications (Pratten and Mathieson, 2024). Logical English (Kowalski and Datoo, 2022) is a simplified version of the English language, which may be easily mapped to first-order logic. In that respect, it is close to a controlled natural language (Kaji, 1999; Fuchs, 2021).

Code generation Existing models can generate code in a variety of programming languages, and at varying levels of granularity (Chen et al., 2021). In particular, GitHub repositories are a source of data to train LLMs on code. Codex (Chen et al., 2021) is a GPT-3 model fine-tuned on code from GitHub. Similarly, Deepseek-Coder-V2 was fine-tuned from Deepseek-V2 (DeepSeek-AI et al., 2024), and CodeLlama from Llama 2 (Rozière et al., 2023). In contrast, StarCoder models were trained on code only (Lozhkov et al., 2024). LLMs trained on code are generally proficient on widely-used languages such as Python. But Catala is a very-low-ressource language. To the best of our knowledge, the only existing ressource is the GitHub repository we used in this paper. Querying the tool "Am I in the Stack?"[3] for "CatalaLang" showed that Stack v2.0.1 and v1.2 (Lozhkov et al., 2024) contain the repositories `CatalaLang/catala` and `CatalaLang/catala-website`. The former holds the compiler for Catala, in OCaml. The latter is the source code for `http://catala-lang.paris.inria.fr/`. This means StarCoder models have seen a trace amount of Catala code, in the form

[2] https://openfisca.org/

[3] https://huggingface.co/spaces/bigcode/in-the-stack

of snippets written on the Catala website. Code generation with LLMs may leverage controlled languages and constrained decoding (Shin et al., 2021). Given the amount of data available, we turn instead to efficient methods for fine-tuning LLMs: low-rank parameter adaptation (Hu et al., 2022) and its quantized versions (Dettmers et al., 2023).

Evaluation metrics Benchmarks for code generation generally pair natural-language instructions with reference, expected code output. This makes it possible to evaluate code generation as a machine-translation task. Borrowing from the BLEU score (Papineni et al., 2002), (Ren et al., 2020) introduce CodeBLEU, a combination of 4 metrics meant to measure different aspects of the generated code. How to appropriately assess the quality of code is an active field of research (Paul et al., 2024; Evtikhiev et al., 2023), and we use all relevant metrics to measure model performance. Some benchmarks additionally have unit tests for the generated code, allowing to measure metrics based on functional correctness, such as Pass@k (Chen et al., 2021). While we do have access to some unit tests for Catala code, they are scarce and operate at the level of an entire Catala program, so that we leave to future research how to best leverage them for code evaluation.

3 Dataset

The publicly available Catala code repository on GitHub[4] contains examples of legal texts translated into Catala by computer scientists and lawyers. Topics include housing aid *(aides logement)*, family allowances *(allocations familiales)*, the monthly basis for family benefits *(base mensuelle allocations familiales)*, inheritance law *(droit successions)*, and income tax (*impôt sur le revenu)*. We extracted and structured the data into JSON format. Each sample in our dataset corresponds to a single provision in a legal statute, structured as follows (see Figure 1):

- **Input**: The text of the original legal provision in French. This text describes rules, conditions, and regulations that need to be translated into Catala code.
- **Metadata**: Catala code describing legal concepts and data types involved in the implementation. This includes definitions of enumerations, structures, and dependencies, used directly in the Catala translation of the input.
- **Output**: The translation of the Input in Catala.

The dataset was randomly split into 70% training, 15% validation and 15% test. Since samples come from diverse legal contexts and are shuffled before splitting, the training, validation and test sets share similar statistical properties. As shown in Table 1, the dataset has 416 training, 86 validation and 89 test samples, with varying input and metadata lengths. This can be challenging, as our 4096-token context window may not capture all information. However, we estimate it fully covers 85% of the samples. The size of the resulting dataset is comparable to other specialized code generation datasets (Ling et al., 2016; Yin et al., 2018).

4 Metrics

We use multiple metrics, each analyzing the code from a different perspective. Our approach considers lexical similarity, syntactic correctness, and structural validity. The evaluation framework includes 5 metrics: (1) ChrF, character-based similarity between reference and generated code, (2) BERTScore: semantic similarity using text embedding models, (3) Tree Edit Distance (TED): structural similarity of syntax trees, (4) Valid Syntax (VS): checks if the generated code is syntactically correct, and (5) CodeBLEU (Ren et al., 2020).

4.1 ChrF

Character n-gram F-score (ChrF) (Popović, 2015) is often used in translation tasks because it captures small differences that word-based metrics might miss. In our evaluation, we use the python *evaluate*[5] library by Hugging Face to compute this score. According to (Evtikhiev et al., 2023), ChrF aligns best with human assessment among other code generation metrics.

4.2 BERTScore

BERTScore (Zhang et al., 2020) uses an encoder-only transformer model to compare the meaning of two pieces of text by computing the similarity between their embeddings. Unlike token-based methods, it evaluates similarity based on context and text embeddings. This is useful because different pieces of code can have different syntax but still perform

[4] https://github.com/CatalaLang/catala-examples

[5] https://huggingface.co/spaces/evaluate-metric/chrf

Split	Number of samples	Mean length			Max length		
		Input	Metadata	Output	Input	Metadata	Output
Train	416	1293.1	2491.4	716.3	44211	10136	37583
Validation	86	1128.5	2599.6	626.5	10214	11081	7574
Test	89	1658.3	2847.7	487.0	26267	9662	2626

Table 1: Dataset statistics. Length measured in number of characters.

the same task. We use the BERTScore implementation from the *evaluate*[6] library. BERTScore — together with ChrF — is the closest metric to human assessment (Evtikhiev et al., 2023).

4.3 Tree Edit Distance

TED quantifies the differences between two Abstract Syntax Trees (ASTs) by computing the minimum number of operations required to transform one tree into another. The allowed operations are node insertion, deletion, and modification, each assigned a cost of 1. This metric considers the global syntactic structure of the code.

To compute the TED, we first generate the Abstract Syntax Tree for both the generated and reference code using the *tree-sitter*[7] parser generator tool. In order to do this, we exploit the *Catala grammar for tree-sitter*[8]. Once the ASTs are obtained, we convert them into a format compatible with the *zss* library[9] for tree edit distance computation. Specifically, we traverse the tree-sitter AST and transform it into a zss tree. After constructing the zss tree representations, we compute the zss distance using the tree edit distance algorithm as described by (Zhang and Shasha, 1989).

One important aspect of using TED for evaluation is normalization. Since AST sizes can vary significantly, raw TED values alone are not always informative. To ensure a fair comparison, we normalize TED by dividing it by the number of nodes in the larger tree, excluding certain common nodes that do not add meaningful differences. The normalized TED is given by:

$$TED_n = \frac{TED_{zss}}{\max(n_r, n_p) - \text{ex. nodes}}$$

where TED_{zss} is the computed edit distance, n_r and n_p are the number of nodes in the reference and generated ASTs respectively, and *ex. nodes* is the number of excluded common nodes — 4 in our case.[10]

A lower TED value means fewer transformations are needed to make the syntax trees identical, indicating a high structural similarity between the generated and reference code. Conversely, a higher TED value suggests significant structural differences. For example, in the case illustrated in Figure 2, the two ASTs contain 16 and 26 nodes. The raw TED value is equal to 10 (the number of white nodes in the Figure), and after normalization, the final TED_n score is 45.5%.

4.4 Valid Syntax

Even if a generated code snippet appears similar to a reference implementation, it may still contain syntax errors that prevent it from compiling. We measure whether a snippet of generated code compiles using its AST. While generating the AST, the Tree-Sitter parser introduces specific error-labeled nodes when encountering syntactic anomalies in the input code. We check for the presence of these error nodes (see for instance the *ERROR* node in the right tree in Figure 2). If such nodes exist, the generated code is marked as syntactically invalid. This metrics effectively assesses how often model produces functional code.

4.5 CodeBLEU

The CodeBLEU metric (Ren et al., 2020) is designed to evaluate the similarity between generated and reference code while taking into consideration syntactic structure and semantics. The evaluation consists of four components: (1) BLEU Score, (2) Weighted N-gram Match, (3) Syntax Tree Match, and (4) Semantic Data Flow Match. Each of these components contributes to the final score through a weighted sum, as described later in this section.

[6] https://huggingface.co/spaces/evaluate-metric/bertscore

[7] https://tree-sitter.github.io/tree-sitter/

[8] https://github.com/CatalaLang/tree-sitter-catala

[9] https://pythonhosted.org/zss

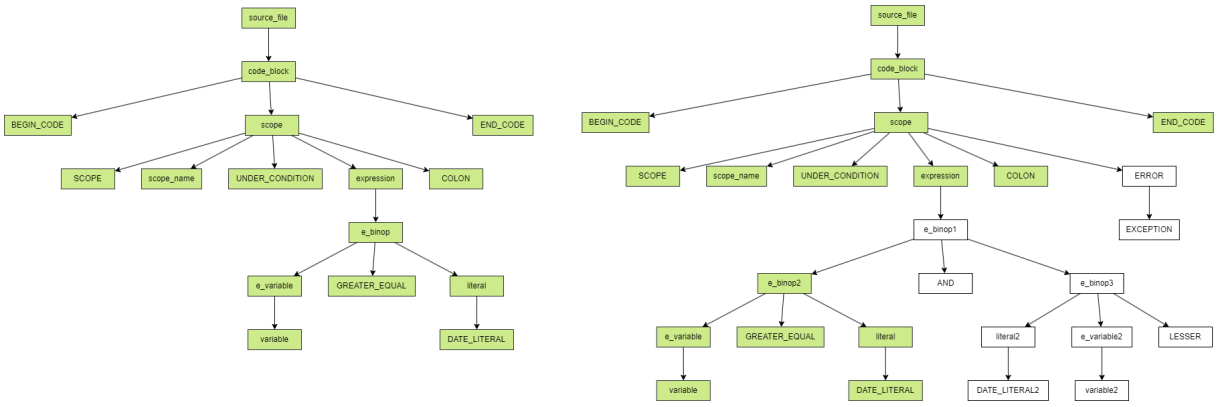

Figure 2: Comparison of ASTs from Figure 3 (left) and Figure 4 (right). Green nodes are shared by both ASTs, while white nodes appear only in the right-hand tree. The labels of the nodes correspond to the elements defined in the grammar, such as keywords and symbols.

```
champ d'application
CalculAidePersonnalisee
sous condition date_courante
>= |2023-01-01|:
```

Figure 3: Example of generated Catala code.

```
champ d'application
CalculAidePersonnalisee
sous condition date_courante
>= |2023-01-01| et
date_courante < |2023-10-01|:
exception metropole
```

Figure 4: Example of reference Catala code.

BLEU Score The first component of CodeBLEU is the standard BLEU score, measuring n-gram overlap between the generated and reference code. We use the default space-based tokenizer.

Weighted N-gram Match Keywords in the programming language play a crucial role in defining the logic and structure of a program, while variable names and literals can often be modified without affecting the overall functionality. To address this, CodeBLEU incorporates a weighted n-gram match component, where keywords are given higher importance compared to variable names. We achieve this by using a specialized tokenizer that splits the code based on a predefined list of Catala-specific keywords (see Appendix A). Each token is then assigned a weight (1 for the keywords and 0.2 for the others), ensuring that incorrect predictions of keywords impact the final score more than incorrect predictions of variable names.

Syntax Tree Match To incorporate syntax awareness, CodeBLEU includes a syntax tree match component, which evaluates the similarity between the ASTs of the generated and reference code. Here, we compare both trees by counting the number of matching subtrees, making this a different metric from TED. The more subtrees that match between the generated and reference ASTs, the higher the score. To measure similarity, we compute the number of common subtrees and normalize it using the longest subtree list. This helps reduce the impact of overly long ASTs. We extract all subtrees from both ASTs while preserving duplicates. The intersection gives the count of common subtrees, and normalization is based on the length of the longest subtree list rather than set cardinality. The similarity score is defined as

$$S(A_1, A_2) = \frac{|T(A_1) \cap T(A_2)|}{\max(\text{len}(T(A_1)), \text{len}(T(A_2)))}$$

where $T(A_1)$ and $T(A_2)$ are the lists of subtrees for ASTs A_1 and A_2, respectively. $|T(A_1) \cap T(A_2)|$ represents the number of common subtrees. The denominator ensures that if an AST prediction contains excessive erroneous substructures, the similarity score is penalized.

Semantic Data Flow Match The meaning and functionality of code depends on how variables are related. To capture this, CodeBLEU includes a semantic matching method based on data-flow. A data-flow graph (Guo et al., 2021) represents how values move between variables in a program. Even if two code snippets have similar syntax or structure, their behavior can be different. For example, two functions might be identical, up to the final return statement, one returning the variable `x` and the

other the variable y. Other metrics may still assign a high score, but the semantics of both functions are quite different.

To measure the semantic similarity using data-flow, we follow three steps, following (Guo et al., 2021): (1) Construct data-flow graphs for both candidate and reference code. These graphs are built based on the AST and show how values are passed between variables. (2) Normalize the data-flows. We ignore the original variable names and rename them as `var_0`, `var_1`, etc., based on their order of appearance. (3) Compute the semantic data-flow match score as:

$$\text{Match}_{df} = \frac{\text{Count}_{clip}(DF_{cand})}{\text{Count}(DF_{ref})}$$

Here, $\text{Count}(DF_{ref})$ is the total number of data-flows in the reference, and $\text{Count}_{clip}(DF_{cand})$ is the number of data-flows in the candidate that match the reference.

In this work, we focused on the most fundamental and commonly used operators in Catala: assignments and if-then-else constructs. Specifically, for if-then-else statements, the DFG is computed separately for the condition, then-branch, and else-branch. Variable states from all branches are then unified, while variables that appear only in the condition are discarded, as they do not contribute to the semantic data dependencies.

CodeBLEU Final Score Computation The final CodeBLEU score is a weighted sum of the 4 metrics described above. By default, all weights are equal to $\frac{1}{4}$. If no data-flows are extracted from the reference code ($\text{Count}(DF_{ref}) == 0$), the data-flow match score is set to 0. In this case, we ignore the data-flow component and adjust the weights used in the final CodeBLEU score. The new weights become $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0$ for the n-gram match, weighted syntax match, AST match, and data-flow match respectively. We adapted the implementation of the CodeBLEU Python library[11] to suit our specific use case.

5 Experiments

Our primary goal in this experimental evaluation is to assess the effectiveness of different LLMs in translating legal text into Catala code. Code generation can be approached as either an autoregressive task or a translation task, with LLMs representing the current frontier in this domain. These two interpretations correspond to different model architectures: decoder-only models, which generate code token-by-token in an autoregressive manner, and encoder-decoder models, which process input and output as a sequence-to-sequence task. We focus on decoder-only models, as they are the most common architecture used when working with text-to-code generation.

[11] https://pypi.org/project/codebleu/

n	CodeBLEU	BERTScore	ChrF	TED	VS
0	2.3	59.3	36.6	98.8	2.2
1	39.7	74.9	64.5	61.3	46.1
2	48.7	76.5	67.7	49.5	62.9
4	50.4	77.5	69.3	46.7	69.7
8	51.5	76.8	69.4	45.8	83.1
16	**52.2**	**78.6**	**70.3**	**43.2**	**88.8**

Table 2: Performance (in %) of GPT-4.1 with varying number of few-shot examples (n). Best value for each metric is in **bold**.

5.1 Few-shot prompting with retrieval

As a starting point, we evaluate OpenAI's GPT-4.1 model (gpt-4.1-2025-04-14) using few-shot prompting, without any fine-tuning. We set the temperature to 0, for reproducibility. To retrieve the most relevant few-shot examples for each test input, we use BM25, a ranking algorithm commonly used in information retrieval (Trotman et al., 2014). We use it to retrieve samples from the training set whose input is most similar to the input of the current test sample. For each input, we create a structured prompt that includes the legal text, a set of few-shot examples in JSON format, and optional metadata. The model then responds with the generated Catala code.

We evaluate performance using the metrics defined in Section 4. Table 2 reports our results. We experimented with varying number of few-shot examples, finding that performance consistently and markedly improves with more samples. This is expected, as GPT-4.1 likely hasn't seen any Catala during its training. We note that even with 1 or 2 examples, results are on par with those typically obtained on other benchmarks (Yang et al., 2025).

5.2 Fine-tuning with QLORA

Since Catala is an uncommon programming language, we can reasonably expect to reach higher performance by fine-tuning smaller models on our training set. We selected and tested the smaller variants of four families of models:

- Qwen 2.5 - base and coder version 7B-14B-32B (Hui et al., 2024; Yang et al., 2024)
- Llama 3 - 3.1-8B, 3.2-3B, 3.3-70B (Grattafiori et al., 2024)
- Phi 4 (Abdin et al., 2024)
- DeepSeek-Coder-V2-Lite-Instruct (DeepSeek-AI et al., 2024)

All of these models were previously fine-tuned by their creators to produce the "Instruct" variants. We opted for this version instead of the base one, as the conversational style aligns better with typical user interactions.

Each training sample was formatted using a structured chat template to align with the conversational style of instruction-tuned models. The template includes:

- A **system message** providing high-level instructions on translating legal text to Catala code.
- A **user query** containing the legal paragraph and metadata.
- An **assistant response** for the Catala code output.

5.2.1 Quantization

To adapt the selected models to our task, we fine-tuned them using QLoRA (Dettmers et al., 2023), a variation of LoRA (Low-Rank Adaptation) (Hu et al., 2022), which enables efficient fine-tuning with reduced memory usage. The fine-tuning was conducted using the Unsloth library. (Daniel Han and team, 2023)

First, to assess the impact of 4-bit quantization on model performance, we compared the results of the fine-tuned quantized models with their full-precision counterparts. Fine-tuning was done for 3 epochs, with a maximum sequence length of 4096 tokens and a learning rate of 3×10^{-4}.

Our evaluation, reported in Table 3, illustrates the impact of different quantization levels on model performance, comparing no quantization (*none*), quantization at test time only (*eval*) and quantization at both train and test time (*both*). While quantization enables efficiency in deployment, it often comes at the cost of reduced precision in code generation. Our experiments confirm this trade-off, showing that models quantized only during inference suffer from performance degradation — an expected outcome since Quantization-Aware Training methods were not used. However, we found that models quantized during both finetuning and inference perform similarly to their non-quantized counterparts. Based on these results, we chose 4-bit quantized models for the remainder of our evaluation.

Setting	C.BLEU	BERTS.	ChrF	TED	VS
Phi-4:					
none	42.6	79.4	68.8	46.0	**83.1**
eval	37.0	78.1	66.7	51.5	82.0
both	**44.5**	**80.2**	**70.2**	**45.1**	79.8
Qwen2.5-14B-Instruct:					
none	**43.2**	**78.7**	69.5	48.2	74.2
eval	33.5	74.7	63.3	57.5	71.9
both	42.9	**78.7**	**70.5**	**46.8**	**85.4**

Table 3: Comparison between various settings of quantization. Best for each quantization configuration is **bolded**. Metrics in %.

5.2.2 Hyperparameter search

We performed a grid search over LoRA-specific hyperparameters to identify the combination yielding the best results under our hardware constraints. We decided to optimize *rank* (8, 16, 32, 64)[12] and *dropout* (0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6), as preliminary experiments showed they had the most significant impact on downstream performance, while other LoRA parameters (such as *alpha*) and the learning rate contributed minimal improvements. The list of best hyperparameters used during training can be found in Appendix B.

6 Discussion

Table 4 presents a comprehensive comparison of fine-tuned model performance across our evaluation metrics. We note that the smallest model with fine-tuning achieves performance comparable to that of few-shot GPT-4.1. Other models further improve on few-shot GPT-4.1, and reach performance beyond that achieved on other code benchmarks (Yang et al., 2025). As expected, larger models tend to perform better.

Our results break away from previous findings on semantic parsing of legal language, and represent a qualitative jump. Based on the metrics we report, LLMs frequently produce valid Catala code, that could be used in production with moderate edits. Some of that qualitative jump likely

[12]For Llama-70B, we did not try values of Rank beyond 8.

Model	CodeBLEU	BERTScore	ChrF	TED	VS
LLaMA-3.1-8B-Instruct	46.6	76.1	62.9	49.2	74.1
LLaMA-3.2-3B-Instruct	44.9	75.0	61.5	52.6	71.9
LLaMA-3.3-70B-Instruct	48.5	81.1	73.8	42.3	87.5
Phi-4	56.4	81.4	71.8	39.8	92.1
Qwen2.5-7B-Instruct	46.6	76.3	65.1	52.4	61.8
Qwen2.5-14B-Instruct	60.3	82.5	76.3	46.8	**93.2**
Qwen2.5-32B-Instruct	59.0	81.9	76.7	40.6	86.5
Qwen2.5-Coder-7B-Instruct	47.3	77.2	64.2	50.0	71.9
Qwen2.5-Coder-14B-Instruct	58.1	82.0	75.0	41.6	88.8
Qwen2.5-Coder-32B-Instruct	**61.2**	**82.9**	**77.3**	**39.7**	**93.2**
DeepSeek-Coder-V2-Lite-Instruct	25.1	57.5	73.0	80.9	25.8

Table 4: Performance (in %) of instruction-tuned models across evaluation metrics. Best within each family is underlined, overall best is **bolded and underlined**.

stems from design choices in the Catala language, whose syntax is meant to align with that of legal language. Our findings partially confirm that this design choice was implemented successfully. Indeed, as compared to other code benchmarks (Ling et al., 2016; Yin et al., 2018), the translation of legal language to Catala code seems to have a high sample efficiency, both for few-shot learning and fine-tuning. While the quality of the generated code may be far from the quality required of an expert system computing taxes at the scale of an entire country, it may be good enough to help during the pair-programming process intended in Catala translation (Huttner and Merigoux, 2022).

We complete our quantitative assessment with a qualitative analysis of model outputs.

Sample A — Appendix C.1 The generated output is correct in structure. Interestingly, the model generates `date_courante <= |2023-04-30|` instead of the reference `date_courante < |2023-05-01|`. Although logically equivalent, this lowers scores based on exact matches. The TED Score of 7.3% and Syntax Match Score of 89.0% indicate minor structural discrepancies. Despite this, the BERTScore (99.2%) and ChrF score (97.4%) confirm high token-level similarity.

Sample B — Appendix C.2 This example shows that the model can correctly extract the amount of euros (`8,70`) from the input. However, the dates are incorrect due to their absence from the input.

Sample C — Appendix C.3 The generated output closely matches the reference and follows the correct structure and logic. It correctly interprets the input, especially the linear relationship at the end of the input (*323 par personne a charge supplementaire*). The start date (`2022-07-01`) is correct while the end date, which is not present in the input text, is invented by the model.

Sample D — Appendix C.4 This example reveals some limitations and illustrates common errors. First, the code is invalid and does not conform to the Catala grammar. Second, the meaning is only partially captured. The input introduces an exception rule with *"sauf s'il s'agit..."*, which is entirely missing in the generated output. Instead, it attempts — unsuccessfully — to express all logic in a single condition. Additionally, it introduces a date check *date_courante* >= |*2023-04-05*|, which is not present in the input text.

7 Conclusion

In this paper, we have introduced a benchmark and metrics for translating legal text to computer-executable code, starting from open-source Catala code. We further experiment with LLMs in few-shot learning and fine-tuning settings. The performance we report is comparable to other low-resource programming languages. Our results contrast with prior attempts at semantic parsing of legal language, as we reach non-trivial performance.

At present, the model takes as input the legal text and its associated metadata, guiding the generation of the corresponding Catala code. In future iterations, we aim to (1) train and evaluate the model on generating both output code and metadata directly from legal text, (2) translate entire documents at once and (3) include unit tests in the evaluation.

Limitations

We experimented with a specific subset of legal language, French tax law, and with a specific target language, Catala. While we report reasonably good performance, this is not directly comparable to prior work on semantic parsing of legal language, due to a mismatch in evaluation data, input language and domain, and target semantic representation.

The metrics we report have been generally found to correlate with human assessments of the quality of the code. However, Catala code quality is held to a very high standard, given the implications of faulty code in an expert system deployed at a large scale. We do not claim that code generated by LLMs can be used as-is. In addition, we did not include metadata generation, which would be desirable for a practical application.

Finally, our experiments indicate a clear trend: larger models consistently achieve better performance across all evaluation metrics. This suggests that even larger-scale models could yield further improvements. However, due to hardware constraints, we were unable to test models beyond a certain size, limiting our exploration of this scaling effect.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, and 8 others. 2024. Phi-4 technical report. *Preprint*, arXiv:2412.08905.

Layman E Allen and C Rudy Engholm. 1977. Normalized legal drafting and the query method. *J. Legal Educ.*, 29:380.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, LAW-ID@ACL 2013, August 8-9, 2013, Sofia, Bulgaria*, pages 178–186. The Association for Computer Linguistics.

Patrick Blackburn and Johan Bos. 2005. *Representation and Inference for Natural Language - a First Course in Computational Semantics*. CSLI Studies in Computational Linguistics. CSLI Publications.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Michael Han Daniel Han and Unsloth team. 2023. Unsloth.

DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y. Wu, Yukun Li, Huazuo Gao, Shirong Ma, Wangding Zeng, Xiao Bi, Zihui Gu, Hanwei Xu, Damai Dai, Kai Dong, Liyue Zhang, Yishi Piao, and 21 others. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *Preprint*, arXiv:2406.11931.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. 2023. Out of the bleu: How should we assess quality of the code generation models? *Journal of Systems and Software*, 203:111741.

Norbert E. Fuchs. 2021. The law of inertia and the frame problem in attempto controlled English. In *Proceedings of the Seventh International Workshop on Controlled Natural Language (CNL 2020/21)*, Amsterdam, Netherlands. Special Interest Group on Controlled Natural Language.

Wachara Fungwacharakorn and Ken Satoh. 2022. Toward a practical legal rule revision in legal debugging. *Comput. Law Secur. Rev.*, 46:105696.

Grazia Garzo and Alessandro Palumbo. 2025. Human-in-the-Loop: Legal Knowledge Formalization in Attempto Controlled English. In *ISDFS - 13th International Symposium on Digital Forensics and Security*, Boston, United States.

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.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcodebert: Pre-training code representations with data flow. *Preprint*, arXiv:2009.08366.

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 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Liane Huttner and Denis Merigoux. 2022. Catala: moving towards the future of legal expert systems. *Artificial intelligence and law*, pages 1–24.

Hiroyuki Kaji. 1999. Controlled languages for machine translation: state of the art. In *Proceedings of Machine Translation Summit VII, MTSummit 1999, Singapore, September 13-17, 1999*, pages 37–39.

Robert A. Kowalski and Akber Datoo. 2022. Logical english meets legal english for swaps and derivatives. *Artif. Intell. Law*, 30(2):163–197.

Sarah Lawsky. 2020. Form as formalization. *Ohio St. Tech. LJ*, 16:114.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew W. Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, and 38 others. 2024. Starcoder 2 and the stack v2: The next generation. *CoRR*, abs/2402.19173.

L Thorne McCarty. 1976. Reflections on taxman: An experiment in artificial intelligence and legal reasoning. *Harv. L. Rev.*, 90:837.

Denis Merigoux. 2023. Experience report: implementing a real-world, medium-sized program derived from a legislative specification. In *Programming Languages and the Law 2023 (affiliated with POPL)*.

Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. 2021. Catala: a programming language for the law. *Proceedings of the ACM on Programming Languages*, 5(ICFP):1–29.

Leora Morgenstern. 2014. Toward automated international law compliance monitoring (tailcm). Technical report, LEIDOS HOLDINGS INC RESTON VA.

Donald Nute. 1988. Defeasible reasoning and decision support systems. *Decis. Support Syst.*, 4(1):97–110.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Debalina Ghosh Paul, Hong Zhu, and Ian Bayley. 2024. Benchmarks and metrics for evaluations of code generation: A critical review. In *IEEE International Conference on Artificial Intelligence Testing, AITest 2024, Shanghai, China, July 15-18, 2024*, pages 87–94. IEEE.

Marcos A. Pertierra, Sarah Lawsky, Erik Hemberg, and Una-May O'Reilly. 2017. Towards formalizing statute law as default logic through automatic semantic parsing. In *Proceedings of the Second Workshop on Automated Semantic Analysis of Information in Legal Texts co-located with the 16th International Conference on Artificial Intelligence and Law (ICAIL 2017), London, UK, June 16, 2017*, volume 2143 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

David Robert Pratten and Luke Mathieson. 2024. Relational expressions for data transformation and computation. In *Databases Theory and Applications*, pages 241–255, Cham. Springer Nature Switzerland.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *Preprint*, arXiv:2009.10297.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Ken Satoh. 2023. PROLEG: practical legal reasoning system. In David Scott Warren, Verónica Dahl, Thomas Eiter, Manuel V. Hermenegildo, Robert A. Kowalski, and Francesca Rossi, editors, *Prolog: The Next 50 Years*, volume 13900 of *Lecture Notes in Computer Science*, pages 277–283. Springer.

Marek J. Sergot, Fariba Sadri, Robert A. Kowalski, Frank Kriwaczek, Peter Hammond, and H Terese Cory. 1986. The british nationality act as a logic program. *Communications of the ACM*, 29(5):370–386.

Sergio Servantez, Nedim Lipka, Alexa Siu, Milan Aggarwal, Balaji Krishnamurthy, Aparna Garimella, Kristian Hammond, and Rajiv Jain. 2023. Computable contracts by extracting obligation logic graphs. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law*, ICAIL '23, page 267–276, New York, NY, USA. Association for Computing Machinery.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7699–7715. Association for Computational Linguistics.

Vu Trong Sinh and Le Minh Nguyen. 2018. An empirical evaluation of AMR parsing for legal documents. In *New Frontiers in Artificial Intelligence - JSAI-isAI 2018 Workshops, JURISIN, AI-Biz, SKL, LENLS, IDAA, Yokohama, Japan, November 12-14, 2018, Revised Selected Papers*, volume 11717 of *Lecture Notes in Computer Science*, pages 131–145. Springer.

Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. Improvements to BM25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium, ADCS 2014, Melbourne, VIC, Australia, November 27-28, 2014*, page 58. ACM.

Aaron Steven White, Elias Stengel-Eskin, Siddharth Vashishtha, Venkata Subrahmanyan Govindarajan, Dee Ann Reisinger, Tim Vieira, Keisuke Sakaguchi, Sheng Zhang, Francis Ferraro, Rachel Rudinger, Kyle Rawlins, and Benjamin Van Durme. 2020. The universal decompositional semantics dataset and decomp toolkit. In *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*, pages 5698–5707. European Language Resources Association.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Zezhou Yang, Sirong Chen, Cuiyun Gao, Zhenhao Li, Xing Hu, Kui Liu, and Xin Xia. 2025. An empirical study of retrieval-augmented code generation: Challenges and opportunities. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pages 476–486. ACM.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. *Preprint*, arXiv:1904.09675.

May Myo Zin, Ha-Thanh Nguyen, Ken Satoh, Saku Sugawara, and Fumihito Nishino. 2023. Improving translation of case descriptions into logical fact formulas using legalcasener. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law, ICAIL 2023, Braga, Portugal, June 19-23, 2023*, pages 462–466. ACM.

May Myo Zin, Ken Satoh, and Georg Borges. 2024. Leveraging LLM for identification and extraction of normative statements. In *Legal Knowledge and Information Systems - JURIX 2024: The Thirty-seventh Annual Conference, Brno, Czech Republic, 11-13 December 2024*, volume 395 of *Frontiers in Artificial Intelligence and Applications*, pages 215–225. IOS Press.

Mounira Nihad Zitouni, Amal Ahmed Anda, Sahil Rajpal, Daniel Amyot, and John Mylopoulos. 2024. Towards the llm-based generation of formal specifications from natural-language contracts: Early experiments with symboleo. *CoRR*, abs/2411.15898.

A Catala Keywords for CodeBLEU

The following is the list of Catala-specific French keywords used in our tokenizer. We used keywords from the Catala tree-sitter grammar. `champ d'application, conséquence, donnée, dépend de, déclaration, contexte, décroissant, croissant, de, liste, contient, énumération, entier, argent, texte, décimal, date, durée, booléen, somme, rempli, définition, état, étiquette, exception, égal à, selon, n'importe quel, sous forme, sous condition, si, alors, sinon, condition, contenu, structure, assertion, varie, avec, pour, tout, on a, fixé, par, règle, soit, existe, dans, parmi, tel, que, et, ou, ou bien, non, maximum, minimum, combinaison de, initialement, est, vide, mais en remplaçant, nombre, an, mois, jour, vrai, faux, entrée, résultat, interne, arrondi, accès_jour, accès_mois, accès_année, premier_jour_du_mois, dernier_jour_du_mois, Inclusion, Module, Usage de, en tant que, externe`

B Hyperparameters

LoRA hyperparameters used for fine-tuning the models.

Model	Rank	Alpha	Dropout
Llama-3.1-8B	64	64	0.2
Llama-3.2-3B	128	128	0.2
Llama-3.3-70B	8	16	0.0
phi-4	64	64	0.4
Qwen2.5-7B	64	64	0.6
Qwen2.5-14B	64	64	0.3
Qwen2.5-32B	64	64	0.0
Qwen2.5-Coder-7B	64	64	0.4
Qwen2.5-Coder-14B	64	64	0.1
Qwen2.5-Coder-32B	128	128	0.2
DeepSeek-Coder-V2	64	64	0.0

Table 5: LoRA hyperparameters used during fine-tuning for each model

C Samples

All the generated outputs below are generate by the best model according to our metrics: Qwen2.5-Coder-32B-Instruct.

C.1 Sample A

```
champ d'application Smic :
definition brut_horaire sous condition
date_courante >= |2023-01-01| et
date_courante < |2023-05-01| et (
  (residence = Metropole) ou
  (residence = Guadeloupe) ou
  (residence = Guyane) ou
  (residence = Martinique) ou
  (residence = LaReunion) ou
  (residence = SaintBarthelemy) ou
  (residence = SaintMartin) ou
  (residence = SaintPierreEtMiquelon)
)
consequence egal a 11.27 EUR
```

Listing 1: Reference Output for sample A

```
champ d'application Smic :
definition brut_horaire sous condition
date_courante >= |2023-01-01| et
date_courante <= |2023-04-30| et (
  (residence = Metropole) ou
  (residence = Guadeloupe) ou
  (residence = Guyane) ou
  (residence = Martinique) ou
  (residence = LaReunion) ou
  (residence = SaintBarthelemy) ou
  (residence = SaintMartin) ou
  (residence = SaintPierreEtMiquelon)
)
consequence egal a 11.27 EUR
```

Listing 2: Generated Output for sample A

Metric	Score
CodeBLEU	91.9
N-gram Match Score	93.3
Weighted N-gram Match Score	93.3
Syntax Match Score	89.0
BERTScore F1	99.2
CHRF Score	97.4
TED Score	0.7
Validity	True

Table 6: Evaluation scores (in %) for Sample A

C.2 Sample B

```
A Mayotte, son montant est fixe a
8,70 euros l'heure.
```

Listing 3: Input for Sample B

```
champ d'application Smic :
definition brut_horaire sous condition
date_courante >= |2023-05-01| et
date_courante < |2024-01-01| et (
(residence = Mayotte)
)
consequence egal a 8,70 EUR
```

Listing 4: Reference Output for Sample B

```
champ d'application Smic :
definition brut_horaire sous condition
date_courante >= |2023-01-01| et
date_courante <= |2023-04-30| et (
(residence = Mayotte)
)
consequence egal a 8,70 EUR
```

Listing 5: Generated Output for Sample B

Metric	Score
CodeBLEU	71.9
N-gram Match Score	75.6
Weighted N-gram Match Score	73.4
Syntax Match Score	66.7
BERTScore F1	95.1
CHRF Score	92.3
TED Score	2.2
Validity	True

Table 7: Evaluation scores (in %) for Sample B

C.3 Sample C

```
NOTA :

Conformement a l'article 3 de l'arrete du
16 aout 2022 (TREL2220744A), ces
dispositions sont applicables pour les
    prestations
dues a compter du 1er
juillet 2022.

### Article 15 | LEGIARTI000046126962 [archive]

Pour l'application du 5 degre de l'article D.
    823-17
du meme code, le forfait " R0 "
est fixe selon
le tableau suivant (en euros) :

Composition du foyer MONTANT (en euros)
----------------------------------------
    ------------------
Personne seule sans personne a charge 4 870
Couple sans personne a charge 6 977
Personne seule ou couple ayant :
-une personne a charge 8 322
-deux personnes a charge 8 509
-trois personnes a charge 8 834
-quatre personnes a charge 9 163
-cinq personnes a charge 9 488
-six personnes a charge 9 816
-par personne a charge supplementaire 323
```

Listing 6: Input for Sample C

```
champ d'application
CalculAidePersonnaliseeLogementLocatif
sous condition date_courante >= |2022-07-01| et
date_courante <= |2022-12-31|:

etiquette metropole
definition multiplicateur_majoration_r0 egal a
decimal de (nombre_personnes_a_charge - 6)

etiquette metropole
definition abattement_forfaitaire_d823_17 egal a
si nombre_personnes_a_charge = 0 alors
selon situation_familiale_calcul_apl sous forme
-- PersonneSeule: 4 870 EUR
-- Couple: 6 977 EUR
sinon si nombre_personnes_a_charge = 1 alors
8 322 EUR
sinon si nombre_personnes_a_charge = 2 alors
8 509 EUR
sinon si nombre_personnes_a_charge = 3 alors
8 834 EUR
sinon si nombre_personnes_a_charge = 4 alors
9 163 EUR
sinon si nombre_personnes_a_charge = 5 alors
9 488 EUR
sinon si nombre_personnes_a_charge = 6 alors
9 816 EUR
sinon
9 816 EUR + 323 EUR *
    multiplicateur_majoration_r0
```

Listing 7: Reference Output for Sample C

```
champ d'application
CalculAidePersonnaliseeLogementLocatif
sous condition date_courante >= |2022-07-01| et
date_courante < |2023-10-01|:
etiquette metropole definition
multiplicateur_majoration_r0 egal a
decimal de (nombre_personnes_a_charge - 6)

etiquette metropole definition
abattement_forfaitaire_d823_17 egal a
si nombre_personnes_a_charge = 0 alors
(selon situation_familiale_calcul_apl sous forme
-- PersonneSeule: 4 870 EUR
-- Couple: 6 977 EUR)
sinon si nombre_personnes_a_charge = 1 alors
8 322 EUR
sinon si nombre_personnes_a_charge = 2 alors
8 509 EUR
sinon si nombre_personnes_a_charge = 3 alors
8 834 EUR
sinon si nombre_personnes_a_charge = 4 alors
9 163 EUR
sinon si nombre_personnes_a_charge = 5 alors
9 488 EUR
sinon si nombre_personnes_a_charge = 6 alors
9 816 EUR
sinon
9 816 EUR + 323 EUR *
    multiplicateur_majoration_r0
```

Listing 8: Generated Output for Sample C

Metric	Score
CodeBLEU	95.0
N-gram Match Score	93.6
Weighted N-gram Match Score	94.5
Syntax Match Score	91.8
BERTScore F1	98.0
CHRF Score	98.5
TED Score	0.6
Validity	True

Table 8: Evaluation scores (in %) for Sample C

C.4 Sample D

```
###### Article L822-4 | LEGIARTI000049392199

Les aides personnelles au logement ne sont pas
    dues si le local est loue ou
sous-loue en partie a des tiers, sauf s'il s'
    agit d'une personne agee ou
handicapee adulte qui a passe un contrat
    conforme aux dispositions de
l' article L. 442-1 du code de l'action sociale
    et des familles ou d'une
personne de moins de trente ans.
```

Listing 9: Input for Sample D

```
champ d'application
    EligibiliteAidesPersonnelleLogement:
etiquette cas_base_l822_4 regle
    condition_logement_location_tiers
sous condition
menage.logement.loue_ou_sous_loue_a_des_tiers
    sous forme
LoueOuSousLoueADesTiers.Non
consequence non rempli

exception cas_base_l822_4 regle
    condition_logement_location_tiers
sous condition
selon menage.logement.
    loue_ou_sous_loue_a_des_tiers sous forme
-- LoueOuSousLoueADesTiers.Non: vrai
-- LoueOuSousLoueADesTiers.Oui de personne:
(resultat de France.VerificationAgeSuperieurA
    avec
{ -- date_naissance: personne.
    date_naissance_personne_sous_location
-- date_courante: date_courante
-- annees: 30 an }).est_superieur
ou
personne.conforme_article_l442_1
consequence rempli
```

Listing 10: Reference Output for Sample D

```
champ d'application
    EligibiliteAidesPersonnelleLogement:
regle condition_non_ouverture_l822_4
sous condition
selon menage.logement.
    loue_ou_sous_loue_a_des_tiers sous forme
-- Oui de personne_sous_location:
date_courante >= |2023-04-05| et
(
(personne_sous_location.
    date_naissance_personne_sous_location +
personne_sous_location.
    conformement_article_l442_1 * 1 an <=
date_courante) ou
(personne_sous_location.
    date_naissance_personne_sous_location +
personne_sous_location.
    conformement_article_l442_1 * 30 an <=
date_courante)
-- Non: faux
consequence rempli
```

Listing 11: Generated Output for Sample D

Metric	Score
CodeBLEU	28.6
N-gram Match Score	15.6
Weighted N-gram Match Score	11.6
Syntax Match Score	37.0
BERTScore F1	66.8
CHRF Score	50.8
TED Score	80.8
Validity	False

Table 9: Evaluation scores (in %) for Sample D