# Code Generation for Legal Language

Gabriele Lorenzo, Aldo Pietromatera
Supervisor: Nils Holzenberger

**Abstract**

*The translation of legal texts into executable code is a crucial challenge in the automation of legal reasoning, particularly in tax law. Catala, a domain-specific programming language, facilitates this process by enabling structured representations of legislative rules. However, manual translation remains labor-intensive. In this work, we develop a dataset tailored for fine-tuning large language models (LLMs) to automate Catala code generation from legal texts. To assess the quality of generated code, we introduce an evaluation framework that extends existing machine translation metrics with structural and syntactic analyses, including tree edit distance (TED) and a modified version of CodeBLEU. Our experiments demonstrate that fine-tuned models can produce syntactically valid code with minimal human intervention, highlighting the feasibility of automating legal code generation.*

## 1 Introduction

The automation of legal reasoning and the application of laws through their translation into computer code has had a significant impact, particularly in tax law. Since the 1990s, the French tax administration has developed an expert system for calculating taxes and social benefits. While this system has proven to be highly successful, several critical issues have emerged over time. To address these challenges, **Catala** (Merigoux et al., 2021), a programming language specifically designed to meet the unique requirements of tax law, was developed in 2020.

Catala is a programming language designed to enable collaboration between lawyers and computer scientists using pair programming, ensuring an accurate translation of legislation into code. A considerable portion of Catala code has already been written, showcasing its potential as a reliable expert system for tax law.

However, a significant challenge of this method is the substantial human effort required for translation. Each section of tax law takes hours to convert into code, and given that the French tax code spans approximately 3,500 pages, the workload is immense. Additionally, frequent amendments to the law necessitate continuous updates and translations.

In addition, the structure of laws is not strictly linear, as they often contain sections that modify or override provisions stated in earlier parts. This adds significant complexity to their interpretation, as a concept introduced in one section may be redefined, limited, or expanded later. This characteristic makes translating laws into code particularly challenging, requiring careful management of dependencies between provisions to ensure a consistent and faithful implementation of the legal text.

These considerations motivated our work. The main contributions of our research are the following:

- Starting from the existing Catala code corpus we created a new dataset suited for the fine-tuning of LLMs.
- We modified existing machine translation evaluation metrics to assess the accuracy of the outputs produced by our fine-tuned models.
- Given our limited computational resources, we strategically selected models that were both close to the state of the art for this task and optimized for efficiency, balancing performance with constraints on time and hardware.

## 2 Methodology

### 2.1 Dataset

To create a dataset suitable for our project, we leveraged the publicly available Catala code repository on GitHub[1]. This repository contains examples of legal texts implemented in Catala. From this source, we systematically extracted and structured the data into JSON format.
Each sample in our dataset is structured as follows:

- **Input (Text Law)**: The original legal provision in French that serves as the input to our model. This text describes rules, conditions, and regulations that need to be translated into executable logic.
- **Metadata**: Structural information about the legal concepts and data types involved in the implementation. This includes definitions of enumerations, structures, and dependencies, which provide context to guide the

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

model in learning structured representations.

- **Output (Catala Code)**: The corresponding implementation of the legal text in Catala.

**Dataset Description** The dataset was randomly split into 85% training and 15% test, ensuring that no explicit sampling bias was introduced. Since samples come from diverse legal contexts and are shuffled before splitting, the training and test sets share similar statistical properties. However, this does not fully guarantee broad generalization, as certain legal contexts may still be underrepresented, potentially affecting the model's ability to adapt to unseen cases.

As shown in Table 1, the dataset has 502 training 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 around 85% of the samples.

**Enhancing Dataset Utilization** 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:

- **Automate Metadata Generation**: Instead of relying on predefined metadata, we plan to train the model to generate both output code and metadata directly from legal text.
- **Generate Complete Legal Documents**: By learning from legal documents across different domains, the model should be able to generalize better and generate new legal documents for any domain.

This dataset provides valuable context for training models to translate legal text into code. However, it's just a starting point. To achieve better performance, we need to refine it to include more diverse and heterogeneous samples.

## 2.2 Metrics

Evaluating the quality of automatically generated code is important for understanding how well a model performs. Code is not just text — it follows strict rules and has a logical structure. This means that traditional text-based evaluation methods, such as BLEU (Papineni et al., 2002), do not fully capture the accuracy of generated code, as shown in (Evtikhiev et al., 2023). To improve the evaluation, we use a mix of different metrics that analyze the code from various perspectives. Our approach considers lexical similarity, syntactic correctness, and structural integrity. The evaluation framework includes:

- ChrF (Character-based similarity)
- BERTScore (Semantic similarity using deep learning models)
- TED - Tree Edit Distance (Structural similarity using syntax trees)
- Is Valid Syntax Tree (Checks if the generated code is syntactically correct)

| Dataset | Samples | Mean Input | Mean Metadata | Mean Output | Max Input | Max Metadata | Max Output |
|---|---|---|---|---|---|---|---|
| Train | 502 | 1264.9 | 2509.9 | 700.9 | 44211 | 11081 | 37583 |
| Test | 89 | 1658.3 | 2847.7 | 487.0 | 26267 | 9662 | 2626 |

Table 1: Summary of dataset statistics, length measured in number of characters

- CodeBLEU-lite (A modified version of CodeBLEU)

These metrics together provide a more complete understanding of how accurate and useful the generated code is. Next, we provide a more detailed explanation of each metric.

### 2.2.1 ChrF

ChrF (Character n-gram F-score) (Popović, 2015) measures how similar two pieces of code are by analyzing sequences of characters. This metric 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*[2] library by Hugging Face to compute this score.

According to (Evtikhiev et al., 2023), ChrF aligns best with human assessment. However, the authors note that it is not a 'perfect' metric for code generation due to the structural differences between code and natural language.

### 2.2.2 BERTScore

BERTScore (Zhang et al., 2020) uses a transformer encoder-only model to compare the meaning of two pieces of code by computing the similarity between their embeddings. Unlike token-based methods, it evaluates similarity based on context and relationships between words. This is useful because different pieces of code can have different syntax but still perform the same task. We use the BERTScore implementation from the Python *evaluate*[3] library to assess semantic similarity. Again, BERTScore - toghether with ChrF - is the closest to human assessment (Evtikhiev et al., 2023).

### 2.2.3 Tree Edit Distance (TED)

To evaluate the structural similarity between the generated and reference code, we employ Tree Edit Distance (TED). 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 include node insertion, deletion, and modification, each assigned a uniform cost of **1**. This metric is particularly well-suited for programming languages, as it considers the syntactic structure rather than just token-based similarity. For us, it is important to understand how close the generated tree is with respect to the reference tree.

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

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

**How to compute TED** To compute the TED, we first generate the Abstract Syntax Tree for both the generated and reference code using the *tree-sitter*[4] parser generator tool. In order to do this, we exploit the *Catala grammar for tree-sitter*[5] provided by the contributors to the Catala project.

Once the ASTs are obtained, we convert them into a format compatible with the **zss** library[6] 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:

$$TED_n = \frac{TED_{zss}}{\max(n_r, n_p) - \text{ex. nodes}} \qquad (1)$$

where $TED_{zss}$ is the computed edit distance, $n_r$ and $n_p$ refer to the number of nodes in the actual and generated ASTs, respectively and *ex. nodes* is the number of excluded common nodes (4 in our case).

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

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

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

A lower TED value indicates a high structural similarity between the generated and reference code, meaning fewer transformations are needed to make them identical. Conversely, a higher TED value suggests significant structural differences, highlighting substantial deviations in the generated code.

For example, in the case illustrated in Figure 1, 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 0.455. This result quantifies the degree of structural similarity between the two trees, helping us assess how much modification is required for the generated code to match the reference.

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

Listing 1: A very simple example of Catala code.

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

Listing 2: Another basic example of Catala code.

### 2.2.4 Is Valid Syntax Tree

A fundamental aspect of evaluating generated code is verifying its syntactic correctness. Even if a generated code snippet appears similar to a reference implementation, it may still contain syntax

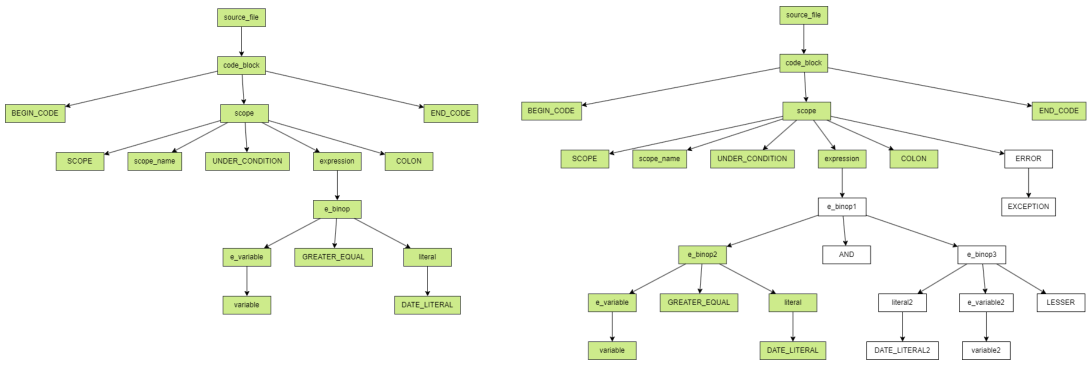

Figure 1: ASTs (left - Listing 1) vs (right - Listing 2)

errors that prevent it from compiling. To efficiently assess this, we analyze the AST of the generated code.

Our approach leverages the parsing capabilities of Tree-Sitter (as described in subsection subsubsection 2.2.3). Given the predefined grammar, the parser constructs an AST representation of the generated code. If the parser encounters syntactic anomalies, it introduces specific error-labeled nodes within the tree structure. Our validation process consists of detecting the presence of these error nodes (you can see an *ERROR* node in right tree in Figure 1). If such nodes exist, the generated code is marked as syntactically invalid (hence, the code in Listing 2 is not valid). This method offers a practical way to assess whether a model's output produces functional code.

### 2.2.5 CodeBLEU-lite

CodeBLEU-lite is a modified version of the original CodeBLEU metric (Ren et al., 2020), designed to evaluate the similarity between generated and reference code while considering syntactic structure and keyword importance. Unlike the full version of CodeBLEU, CodeBLEU-lite does not include the data-flow similarity component, as we did not implement it. However, it retains key components that make it a more effective metric than traditional BLEU for code evaluation. Still, it is important to note that the data-flow similarity component would have been very useful, as it helps understand how variables are connected and how the code actually works, making sure that two pieces of code are not just similar in structure but also in meaning. The evaluation consists of three main components:

- BLEU Score (Lexical Similarity)
- Weighted N-gram Match (Keyword Importance)
- Syntax Tree Match (Structural Similarity)

Each of these components contributes to the final score through a weighted sum, as described later in this section.

**BLEU Score - Lexical Similarity** The first component of CodeBLEU-lite is the *standard* BLEU score. For computing BLEU, we use the default space-based tokenizer. This approach allows us to measure the n-gram overlap between the generated and reference code, providing a rough indication of how similar they are in terms of token sequences.

**Weighted N-gram Match - Keyword Importance** Not all tokens in a programming language have the same importance. Certain keywords, 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-lite 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 (Appendix A.1). 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 - Structural Similarity** Programming languages follow strict syntactic rules, and even small deviations in structure can cause a program to become invalid. To incorporate syntax awareness, CodeBLEU-lite includes a syntax tree match component, which evaluates the similarity between the ASTs of the generated and reference code.

It is important to notice that here we compare the trees by counting the number of matching subtrees (different from what we do for *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 over-generated (very 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:

$$S(A_1, A_2) = \frac{|T(A_1) \cap T(A_2)|}{\max(\text{len}(T(A_1)), \text{len}(T(A_2)))} \quad (2)$$

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.

**CodeBLEU-lite Final Score Computation** The final score is obtained by computing a weighted sum of its three main components:

$$\begin{aligned} \text{CodeBLEU-lite} = \\ 0.33 \times \text{BLEU} \\ +0.33 \times \text{Weighted N-gram Match} \\ +0.33 \times \text{Syntax Tree Match} \\ +\mathbf{0} \times \text{Dataflow Match} \end{aligned} \tag{3}$$

Since the data-flow component was not implemented, its weight is set to 0, making CodeBLEU-lite a reduced but still effective version of the full CodeBLEU metric.

We adapted the implementation of the CodeBLEU Python library[7] to suit our specific use case.

## 2.3 Models

Code generation can be approached as either an autoregressive task or a translation task, with large language models (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.

For our research, we focused on decoder-only models, as they are by far the most common architecture used when working with text-to-code generation. However, we plan to explore encoder-decoder models in future work to evaluate their potential benefits.

Given our hardware constraints, we only selected and tested the smaller variations of three families of models:

- Qwen 2.5 - base and coder version 7B-14B-32B (Hui et al., 2024)(Qwen et al., 2025)
- Llama 3 - 3.1-8B, 3.2-3B, 3.3-70B (Grattafiori et al., 2024)
- Phi 4 (Abdin 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.

**Fine-tuning strategy** 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., 2021), which enables efficient fine-tuning with reduced memory usage.
The fine-tuning was conducted using Unsloth[8], an optimized framework designed for efficient training of large language models on resource-limited hardware. Given our computational constraints, all models were fine-tuned in their quantized 4-bit versions.

To assess the impact of quantization on model performance, we compared the results of the fine-tuned quantized models

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

[8] https://unsloth.ai/

with their full-precision counterparts. Our evaluation, as can be seen in Figure 2, revealed that the quantized models did not exhibit significantly lower performance, indicating that QLoRA-based 4-bit training is a viable approach for our task without major trade-offs in quality.

The detailed list of hyperparameters used during training can be found in Appendix A.2.

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 into Catala code.
- A **user query** containing the legal paragraph and metadata.
- An **assistant response** with the expected Catala code output.

**Evaluation Strategy** To evaluate the performance of our fine-tuned models, we conducted a systematic evaluation using the metrics defined in subsection 2.2.

Our evaluation process consisted in generating outputs, given a natural language law input and the corresponding metadata, and comparing them against the expected Catala code.

To ensure reproducibility, we set a fixed random seed. Moreover, all model outputs were generated using greedy decoding without sampling, ensuring deterministic results.

## 3 Experimental Results

Our primary goal in this experimental evaluation is to assess the effectiveness of different large language models (LLMs) in translating legal text into Catala code.

### 3.1 Results and Analysis

Figure 2 illustrates the impact of different quantization levels on model performance. 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 (QAT) methods were not used. However, we found that models quantized during both finetuning and inference perform similarly to their fully non-quantized counterparts. Based on these results, we chose 4-bit quantized models for our full evaluation.

Table 2 presents a comprehensive comparison of model performance across all evaluation metrics. We categorize models into different families and highlight the best-performing models both within each family (underlined) and overall (bolded).

The Llama-3-3.70B-Instruct model achieves the highest CodeBLEU (0.485), BertScore (0.811), ChrF (0.738), TED (0.423), and is.valid (0.875) across all evaluated models.

Among smaller models, phi-4 performs well in comparison to Qwen2.5, but the larger Qwen2.5-32B-Instruct model shows competitive results.
While Qwen2.5-Coder models show improved is_valid scores, their CodeBLEU remains slightly below Llama-3-3.70B-Instruct. This suggests that although these models generate syntactically valid code more consistently, they may not always produce the most precise translations.

### 3.1.1 Sample Analysis and Observations

**Sample a - Appendix A.3.1** The generated output is mostly correct in structure but contains two issues. First, the date condition uses `date_courante <= |2023-04-30|` instead of `date_courante < |2023-05-01|`. Although logically equivalent, maintaining consistency with the reference format would have prevented potential syntax errors and parsing ambiguities. Second, an extra closing parenthesis before the `consequence` statement makes the generated output invalid.
The syntax error results in an invalid output (`Validity = False`), while the `TED Score = 0.0216` and `Syntax Match Score = 0.8784` indicate minor structural discrepancies. Despite these errors, `BERTScore = 0.9888` and `CHRF = 0.97001` confirm high token-level similarity.

**Sample b - Appendix A.3.2** This sample demonstrates a well-structured output with high alignment to the reference, achieving strong scores across all evaluation metrics. However, some values (the amount of euros at the end) are incorrect because they are established in a previous legal article and do not explicitly appear in the context of this specific sample. This demonstrates that the context is highly valuable when generating code.

**Sample c - Appendix A.3.3** The generated output closely follows the structure of the training samples, capturing the logical flow and formatting accurately. Notably, the model correctly inferred and assigned numerical values for the rent ceilings based on the provided input context. This highlights the model's ability to leverage contextual information effectively when generating structured outputs.

**Sample d - Appendix A.3.4** The generated output is very different from the reference and does not follow the correct structure or numerical logic. Instead of listing the predefined monetary values for different household types, the model wrongly uses a formula that adds a fixed `500€` per dependent instead of applying the correct step-by-step conditions.
The cause of this failure is unclear, as the model has encountered similar structures during training and the correct numerical values are explicitly present in the input. This suggests potential issues in reasoning over structured numerical data or misinterpretation of conditional logic.

## 3.2 Limitations

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.

Additionally, our models were trained and evaluated with a maximum context window of 4096 tokens. This limitation may have impacted performance, penalizing samples with longer dependencies.

Beyond model size, dataset quality also plays a crucial role in performance. A larger dataset, with more diverse and well-structured examples, could enhance generalization and robustness. Improvements in dataset curation—such as reducing noise, ensuring consistent formatting, and including a broader range of examples—could lead to significant gains in model effectiveness.

## 4 Conclusion

Our study confirms the feasibility of automating legal code translation with fine-tuned LLMs, reducing the need for extensive manual intervention.

By leveraging a structured evaluation framework, we assessed both syntactic correctness and logical consistency, highlighting the strengths and limitations of current models. While larger models achieve better performance, computational constraints and dataset limitations remain challenges.

Future work should focus on improving dataset quality, optimizing models for efficiency, and integrating more advanced techniques such as reinforcement learning and constraint decoding to enhance accuracy and adaptability in legal code automation.

| **Model** | **CodeBleu** | **Bertscore (F1)** | **ChrF** | **TED** | **is_valid** |
|---|---|---|---|---|---|
| | | **Llama-3** | | | |
| Llama-3.1-8B-Instruct | 0.360 | 0.764 | 0.656 | 0.552 | 0.629 |
| Llama-3.2-3B-Instruct | 0.203 | 0.670 | 0.497 | 0.742 | 0.292 |
| Llama-3.3-70B-Instruct | <u>**0.485**</u> | <u>**0.811**</u> | <u>**0.738**</u> | <u>**0.423**</u> | <u>**0.875**</u> |
| | | **phi-4** | | | |
| phi-4 | <u>0.445</u> | <u>0.802</u> | <u>0.702</u> | <u>0.451</u> | <u>0.798</u> |
| | | **Qwen2.5** | | | |
| Qwen2.5-7B-Instruct | 0.264 | 0.731 | 0.610 | 0.690 | 0.213 |
| Qwen2.5-14B-Instruct | 0.429 | 0.787 | 0.705 | 0.468 | <u>0.854</u> |
| Qwen2.5-32B-Instruct | <u>0.471</u> | <u>0.809</u> | <u>0.720</u> | <u>0.426</u> | 0.831 |
| | | **Qwen2.5-Coder** | | | |
| Qwen2.5-Coder-7B-Instruct | 0.368 | 0.764 | 0.650 | 0.529 | 0.809 |
| Qwen2.5-Coder-14B-Instruct | 0.390 | 0.779 | 0.671 | 0.507 | 0.831 |
| Qwen2.5-Coder-32B-Instruct | <u>0.474</u> | <u>0.806</u> | <u>0.729</u> | <u>0.426</u> | <u>0.865</u> |

Table 2: Performance metrics of different models. The best value within each model family is underlined, while the overall best values are bolded.

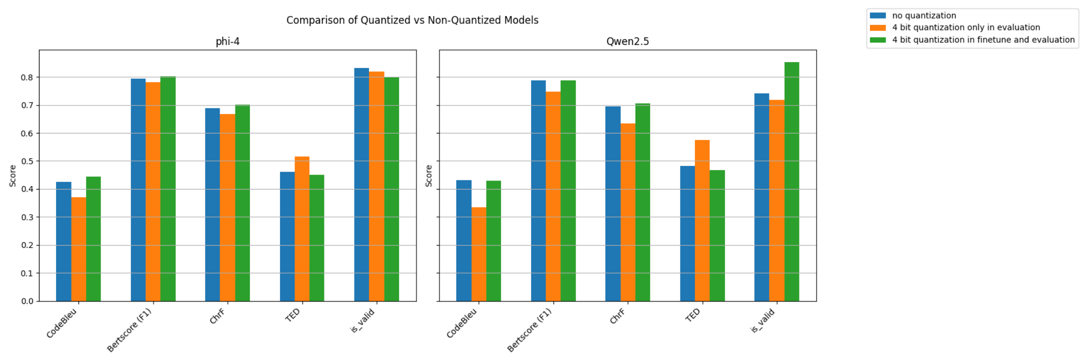

Figure 2: Comparison between various degrees of quantization

# A Appendix

## A.1 Catala Keywords for CodeBLEU-lite

The following is a non-exhaustive list of Catala-specific keywords used in our tokenizer. While we have relied on these keywords for tokenization, we cannot guarantee that this list includes all the keywords in the language.

`alors, argent, assertion, avec, booléen, champ d'application, combinaison de, condition, conséquence non rempli, conséquence rempli, contenu, contexte, contient, dans, date, date arrondi croissant, date arrondi décroissant, de, donnée, durée, décimal, déclaration, dépend de, entier, entrée, est maximum, est minimum, et, exception, existe, faux, initialement, interne, liste, liste de, mais en remplaçant, maximum, minimum, n'importe quel, nombre, non, on a, ou, ou bien, parmi, pour, pour tout, résultat, selon, si, sinon, soit, somme, sous condition, sous forme, structure, tel que, vrai, égal à, énumération, état, étiquette`

## A.2 Hyperparameters

| **Hyperparameter** | **Value** |
|---|---|
| **Model Configuration** | |
| Max Sequence Length | 4096 |
| **LoRA Parameters** | |
| LoRA Rank (r) | 8 |
| LoRA Alpha | 16 |
| LoRA Dropout | 0 |
| **Training Parameters** | |
| Optimizer | `adamw_8bit` |
| Learning Rate | $3 \times 10^{-4}$ |
| Weight Decay | 0.01 |
| Warmup Steps | 30 |
| Number of Epochs | 3 |
| LR Scheduler Type | linear |

Table 3: Hyperparameters used for fine-tuning

| **Hyperparameter** | **Value** |
|---|---|
| Maximum New Tokens | 1024 |
| Do Sample | False |
| Cache Implementation | offloaded |

Table 4: Hyperparameters used for evaluation

## A.3 Samples

### A.3.1 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 3: 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 4: Generated Output for sample a

| **Metric** | **Score** |
|---|---|
| CodeBLEU-lite | 0.8985 |
| N-gram Match Score | 0.9003 |
| Weighted N-gram Match Score | 0.9168 |
| Syntax Match Score | 0.8784 |
| BERTScore F1 | 0.9888 |
| CHRF Score | 0.97001 |
| TED Score | 0.0216 |
| Validity | False |

Table 5: Evaluation scores for Sample a

### A.3.2 Sample b

```
champ d'application CalculAidePersonnaliseeLogementLocatif
sous condition date_courante >= |2020-10-01| et date_courante < |2021-10-01|: exception
    base
definition multiplicateur_majoration_loyer_reference sous condition
(selon residence sous forme
-- Guadeloupe: vrai
-- Martinique: vrai
-- LaReunion: vrai
-- Mayotte: vrai
-- SaintBarthelemy: vrai
-- SaintMartin: vrai
-- n'importe quel: faux) et
nombre_personnes_a_charge >= 6
consequence egal a 5.0
```

Listing 5: Reference Output for Sample b

```
champ d'application CalculAidePersonnaliseeLogementLocatif
sous condition date_courante >= |2020-10-01| et date_courante < |2021-10-01|: exception
    base
definition multiplicateur_majoration_loyer_reference sous condition
(selon residence sous forme
-- Guadeloupe: vrai
-- Martinique: vrai
-- LaReunion: vrai
-- Mayotte: vrai
-- SaintBarthelemy: vrai
-- SaintMartin: vrai
-- n'importe quel: faux) et
nombre_personnes_a_charge >= 6
consequence egal a 6.0
```

Listing 6: Generated Output for Sample b

| **Metric** | **Score** |
|---|---|
| CodeBLEU | 0.9872 |
| N-gram Match Score | 0.9802 |
| Weighted N-gram Match Score | 0.9814 |
| Syntax Match Score | 1.0000 |
| BERTScore F1 | 0.9781 |
| CHRF Score | 0.9937 |
| TED Score | 0.0000 |
| Validity | True |

Table 6: Evaluation scores for Sample b

### A.3.3 Sample c

```
Input: ### Article 8 | LEGIARTI000042378446 [archive]

Dans le cas ou le logement occupe est une chambre, les plafonds
de loyers vises au 2eme de l'article D. 823-16 du meme code sont
fixes comme suit, quelle que soit la taille de la famille (en euros) :

- 90 % des loyers plafonds de location pour une personne isolee ;
- 75 % des loyers plafonds de location pour une personne isolee,
dans le cas des personnes agees ou handicapees adultes hebergees
a titre onereux chez des particuliers.

On obtient les loyers plafonds suivants (en euros) :

MONTANTS DES LOYERS PLAFONDS CHAMBRE EN APL ET EN AL
(arrondis au centime d'euros le plus proche)

-----------------------------------------------------------
Beneficiaires                     Zones  Montants
--------------------------------- ------ --------
Cas general                         I      267,14

II       232,82

III    218,21


Cas des personnes agees         I      222,62
ou handicapees adultes
hebergees a titre onereux
chez des particuliers

II     194,02

III    181,85
-----------------------------------------------------------
```

Listing 7: Input for Sample c

```
champ d'application CalculAidePersonnaliseeLogementLocatif
sous condition
date_courante < |2021-10-01| et date_courante >= |2020-10-01| et
logement_est_chambre:

etiquette chambre exception base
definition plafond_loyer_d823_16_2 egal a
selon zone sous forme
-- Zone1: 267,14 EUR
-- Zone2: 232,82 EUR
-- Zone3: 218,21 EUR

exception chambre definition plafond_loyer_d823_16_2 sous condition
agees_ou_handicap_adultes_hebergees_onereux_particuliers
consequence egal a
selon zone sous forme
-- Zone1: 222,62 EUR
-- Zone2: 194,02 EUR
-- Zone3: 181,85 EUR
```

Listing 8: Reference Output for Sample c

```
champ d'application CalculAidePersonnaliseeLogementLocatif
sous condition
date_courante >= |2020-10-01| et date_courante < |2021-10-01| et
logement_est_chambre:

etiquette chambre exception base
definition plafond_loyer_d823_16_2 egal a
selon zone sous forme
-- Zone1: 267,14 EUR
-- Zone2: 232,82 EUR
-- Zone3: 218,21 EUR

exception chambre definition plafond_loyer_d823_16_2 sous condition
agees_ou_handicap_adultes_hebergees_onereux_particuliers
consequence egal a
selon zone sous forme
-- Zone1: 222,62 EUR
-- Zone2: 194,02 EUR
-- Zone3: 181,85 EUR
```

Listing 9: Generated Output for Sample c

| **Metric** | **Score** |
|---|---|
| CodeBLEU | 0.9168 |
| N-gram Match Score | 0.9487 |
| Weighted N-gram Match Score | 0.9411 |
| Syntax Match Score | 0.8604 |
| BERTScore F1 | 0.9888 |
| CHRF Score | 1.0 |
| TED Score | 0.0202 |
| Validity | True |

Table 7: Evaluation scores for Sample c

### A.3.4 Sample d

```
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 10: Input for Sample d

```
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 11: Reference Output for Sample d

```
champ d'application CalculAidePersonnaliseeLogementLocatif
sous condition date_courante >= |2022-07-01| et date_courante < |2023-10-01|:
etiquette base definition abattement_forfaitaire_d823_17 egal a
(selon situation_familiale_calcul_apl sous forme
-- PersonneSeule: 4 870 EUR
-- Couple: 6 977 EUR) +
500 EUR * (decimal de nombre_personnes_a_charge)
```

Listing 12: Generated Output for Sample d

| **Metric** | **Score** |
|---|---|
| CodeBLEU | 0.1588 |
| N-gram Match Score | 0.0822 |
| Weighted N-gram Match Score | 0.1612 |
| Syntax Match Score | 0.2329 |
| BERTScore F1 | 0.6979 |
| CHRF Score | 0.4364 |
| TED Score | 0.7117 |
| Validity | True |

Table 8: Evaluation scores for Sample d

## 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, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report, 2024. URL https://arxiv.org/abs/2412.08905.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023. URL https://arxiv.org/abs/2305.14314.

Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. Out of the bleu: How should we assess quality of the code generation models? *Journal of Systems and Software*, 203: 111741, September 2023. ISSN 0164-1212. doi: 10.1016/j.jss.2023.111741. URL http://dx.doi.org/10.1016/j.jss.2023.111741.

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, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan

Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

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, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. Catala: a programming language for the law. *Proceedings of the ACM on Programming Languages*, 5 (ICFP):1–29, August 2021. ISSN 2475-1421. doi: 10.1145/3473582. URL http://dx.doi.org/10.1145/3473582.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 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, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://doi.org/10.3115/1073083.1073135.

Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. URL https://aclanthology.org/W15-3049/.

Qwen, :, 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, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong

Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis, 2020. URL https://arxiv.org/abs/2009.10297.

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18 (6):1245–1262, 1989. doi: 10.1137/0218082. URL https://doi.org/10.1137/0218082.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020. URL https://arxiv.org/abs/1904.09675.