Fine-Tuning LLMs for Dutch Language Tasks: Gemma3 4b vs. LLaMA3.2 3b

Adnane Acudad, Akram Chakrouni,

***Abstract*—This project aims to enhance the performance of large language models (LLMs) in a low-resource language setting, specifically for Dutch language tasks such as summarization and comprehension. To achieve this, we fine-tune two pre-trained models, Gemma3 4b and LLaMA3.2 3b, using parameter-efficient techniques (LoRA/QLoRA) with 4-bit quantization on a diverse dataset including conversational data, translations, synthetic instructions, and legal summaries. Additionally, we fine-tune a task-specific model (LLaMA3.2 3b Legal FT) exclusively on Dutch legal summaries. Experimental results indicate that general fine-tuning improves LLaMA3.2 3b over its base version on a mixed evaluation set, while Gemma3 4b shows mixed results compared to its strong baseline. The task-specific legal model demonstrates significantly stronger adaptation to its domain when evaluated on legal data, outperforming other models considerably, although its scores are lower on the mixed evaluation set. These findings underscore the effectiveness of adapting pre-trained models for specific needs in low-resource settings, highlight the trade-offs of task specialization, and stress the importance of domain-aligned evaluation data. Ethical considerations regarding data bias and privacy were maintained throughout the data handling process.**

I. Introduction

The primary objective of this project is to enhance the performance of large language models (LLMs) in a low-resource language setting, specifically for Dutch language tasks such as summarization and comprehension. To achieve this, we fine-tune two pre-trained models; Gemma3 4b [1] and LLaMA3.2 3b [2], using parameter-efficient techniques including LoRA [3] and QLoRA [4], combined with 4-bit quantization. We also explore task-specific adaptation by fine-tuning a LLaMA3.2 3b variant solely on legal documents (‘LLaMA3.2 3b Legal FT‘). This approach aims to improve model fluency and task-specific accuracy without incurring the substantial computational costs associated with training an LLM from scratch.

The project is structured as follows: the **Background and Related Work** section reviews the fundamental concepts and relevant literature on LLMs, fine-tuning, and parameter-efficient adaptation methods. The **Methodology** section details our experimental setup, fine-tuning process, and training parameters for the general and task-specific models. In the **Data Collection and Preprocessing** section, we describe the assembly and preparation of the diverse Dutch instruction dataset used for general fine-tuning and the legal dataset for specialized tuning, including data formatting. The **Experimental Results** section presents a comparative analysis of the models' performance across two distinct evaluation contexts. The **Discussion**, **Ethical Considerations**, and **Conclusion** sections reflect on the outcomes, the evaluation limitations, ethical practices, and potential future directions.

By adapting these models for Dutch, our work demonstrates performance improvements through fine-tuning and explores the effects of task specialization, providing a practical framework for adapting LLMs in low-resource languages.

II. Background and Related Work

The recent rise of large language models (LLMs) can be traced to the seminal work “Attention Is All You Need” by Vaswani et al. [5], which introduced the transformer architecture. Since then, LLMs such as BERT [6], GPT-3 [7], and LLaMA [8] have demonstrated remarkable capabilities in natural language understanding and generation. These models are trained on massive corpora using self-supervised learning objectives and scaled up to billions of parameters. Their ability to generalize across diverse tasks with little to no task-specific data has made them the foundation of modern NLP.

A. Large Language Models

An LLM is a transformer-based model pre-trained on large-scale textual data to predict the next token in a sequence. This autoregressive objective enables the model to learn grammar, semantics, world knowledge, and task-relevant features without explicit supervision. LLMs like GPT-3 have been shown to perform surprisingly well in zero-shot and few-shot settings by conditioning on task instructions and examples [7].

B. Why Fine-Tuning?

Despite their generalization abilities, LLMs often underperform in domain-specific or low-resource languages. Fine-tuning is a method of adapting pre-trained models to a downstream task or domain using a smaller, task-specific dataset. This process allows models to retain general knowledge while specializing in new areas. Fine-tuning can be full (updating all weights) or parameter-efficient (only updating a subset), depending on resource constraints.

Fine-tuning was formalized in various contexts, such as domain-adaptive fine-tuning [9] and instruction tuning [10], where LLMs were taught to follow human-written prompts more effectively. Instruction-tuned models like FLAN-T5 and ChatGPT [11] showed substantial performance gains over base models.

C. Parameter-Efficient Fine-Tuning (PEFT)

Recent advances have made it feasible to adapt LLMs on consumer hardware through Parameter-Efficient Fine-Tuning (PEFT). These methods reduce computational cost by freezing most of the model and only training small, added components. Among the most impactful techniques are:

- **LoRA (Low-Rank Adaptation)**: Introduced by Hu et al. [3], LoRA injects low-rank trainable matrices into transformer layers while keeping the original weights frozen. This drastically reduces trainable parameters and memory consumption.
- **QLoRA (Quantized LoRA)**: Proposed by Dettmers et al. [4], QLoRA applies 4-bit quantization to reduce memory footprint further, enabling training of large models on a single GPU. It combines LoRA with quantized weights and double quantization to retain performance.
- **FlashAttention**: A highly optimized attention kernel that reduces memory and increases throughput by computing attention in blocks. This method was introduced by Dao et al. [12], significantly speeding up transformer training and inference, particularly on long sequences.

D. Relevance to Dutch Language Adaptation

Although general-purpose LLMs exhibit limited performance in non-English contexts, their architecture allows for efficient adaptation to low-resource languages through fine-tuning. By applying PEFT techniques, we can specialize models like LLaMA3.2 3b and Gemma3 4b in Dutch tasks such as summarization and understanding without retraining from scratch. This approach enables democratized access to high-quality language tools for underrepresented linguistic communities.

III. Methodology

This section details the experimental setup, fine-tuning process, and training parameters employed to adapt the Gemma3 4b and LLaMA3.2 3b models for enhanced performance in Dutch language tasks, including a task-specific legal model.

A. Experimental Setup

Five models were compared:

- **Gemma3 4b Base**: The original pre-trained Gemma3 4b model.
- **LLaMA3.2 3b Base**: The original pre-trained LLaMA3.2 3b model.
- **Gemma3 4b Fine-tuned (‘aacudad/gemma-3-DUTCH‘)**: Gemma3 4b base fine-tuned on the general Dutch dataset.
- **LLaMA3.2 3b Fine-tuned (‘aacudad/llama-3b-DUTCH‘)**: LLaMA3.2 3b base fine-tuned on the general Dutch dataset.
- **LLaMA3.2 3b Legal Fine-tuned (‘aacudad/dutchlegal-32k‘)**: LLaMA3.2 3b base fine-tuned specifically on a dataset of 100,000 Dutch legal summaries.

Baseline performance for the base models was established prior to adaptation.

B. Fine-Tuning Process

The fine-tuning process is executed using the Unsloth [13] framework, facilitating efficient adaptation via parameter-efficient techniques. Key aspects include:

1) **Model Loading and Quantization**: All models are loaded with 4-bit quantization to reduce memory footprint.
2) **Adapter Training with LoRA/QLoRA**: Only low-rank adapter modules are fine-tuned using LoRA [3] / QLoRA [4], allowing adaptation to Dutch while minimizing computational demands.
3) **Inference Setup**: Adapted models are configured for inference tasks. Pipelines are optimized for low-latency responses.

C. Training Parameters

Fine-tuning was performed on an NVIDIA RTX A6000 (48 GB VRAM). Key hyperparameters:

- **Training Duration**:
  - **Gemma3 Fine-tuned**: Approx. 9 hours (15,000 steps on the general dataset).
  - **LLaMA3.2 Fine-tuned**: Approx. 46 hours (one epoch over the general dataset).
  - **LLaMA3.2 Legal Fine-tuned**: Approx. 3.5 days (one epoch over 100,000 legal summaries).
- **Batch Size**: 16 to 32 samples per iteration.
- **Learning Rate**: Between 1×10^{-4} and 5×10^{-4}.
- **Context Window**: Gemma3 4b models used a 4096-token window. LLaMA3.2 3b models used a 16384-token window (or 32k for the legal model, matching its name).
- **Quantization Strategy**: All models utilized 4-bit quantization for memory efficiency.

IV. Data Collection and Preprocessing

To ensure robust training, distinct datasets were used for general and task-specific fine-tuning. This involved assembling data from various sources and formatting it appropriately for instruction tuning.

A. General Fine-tuning Dataset Sources

A comprehensive dataset (292,500 examples) was assembled by merging several sources for the general fine-tuning of Gemma3 4b and LLaMA3.2 3b:

- **ultrachat_200k_dutch** [14]: 200,000 Dutch dialogues (instruction-response).
- **Translated Nemotron Data** [15]: 7,500 English rows translated to Dutch (Gemini API).
- **Synthetic Data** [16]: 80,000 Dutch examples generated (GPT-4o-mini, Gemini APIs) across diverse topics.
- **Legal Case Summaries** [17]: 5,000 Dutch legal cases extracted from *rechtspraak.nl* [18] and summarized (GPT-4o-mini). This subset was included for diversity in the general training mix.

Table I provides an overview of the sources used for the general dataset.

Source	Count	Purpose for General Fine-tuning
ultrachat_200k_dutch [14]	200,000	Instruction tuning baseline (conversational)
Translated Nemotron [15]	7,500	Instruction diversity via translation
Synthetic (GPT-4o-mini + Gemini APIs) [16]	80,000	General-domain instruction generation
Summarized Cases (rechtspraak.nl) [17]	5,000	Long-form legal summarization diversity
Total (General Dataset)	**292,500**	**Merged and shuffled before fine-tuning Gemma3/LLaMA3.2**

TABLE I
OVERVIEW OF DATA SOURCES FOR GENERAL FINE-TUNING

B. Task-Specific Legal Dataset

For the ‘aacudad/dutchlegal-32k‘ model, a separate, larger dataset was curated:

- **Legal Summaries**: A collection of 100,000 Dutch legal case summaries, specifically prepared for training the specialized legal model. This dataset focuses solely on the legal domain, derived from public records on *rechtspraak.nl* [18].

C. Data Formatting and Preprocessing

Regardless of the source, all data was standardized into a conversational instruction-following format suitable for fine-tuning. Each data point consists of a user instruction and the expected assistant response. The general dataset, particularly the ultrachat and synthetic components, heavily utilizes this multi-turn conversational structure. An example entry format is shown below:

```
[
  {
    "content": "Hoe begin ik met bullet
    ↪  journaling?",
    "hash": "4f97b0ad1cd8fb8...",
    "conversations": [
      {
        "role": "user",
        "content": "Hoe begin ik met bullet
        ↪  journaling?"
      },
      {
        "role": "assistant",
        "content": "Bullet journaling is een
        ↪  flexibel systeem om je leven te
        ↪  organiseren..."
      }
    ]
  }
  // ... more entries ...
]
```

Key preprocessing steps included:

1) **Generation and Translation**: Synthetic data was generated using Python scripts leveraging Gemini and GPT-4o-mini APIs across a wide range of topics to ensure diversity. English datasets like Nemotron were translated via API calls. (See Appendix A for generation script snippets).
2) **Cleaning**: Standard text cleaning involved removing inconsistencies, normalizing whitespace, and ensuring UTF-8 encoding.
3) **Structuring**: Data from various sources (dialogues, summaries, translations) was converted into the ‘conversations‘ list format with ‘user‘ and ‘assistant‘ roles.
4) **Hashing**: A SHA-256 hash of the initial user instruction (‘content‘ field) was included to help identify potential duplicates during merging.
5) **Merging and Shuffling**: The components for the general dataset were merged using custom scripts (Appendix A) and thoroughly shuffled before training. The 100k legal dataset was kept separate for specialized tuning.
6) **Domain-Specific Processing**: Raw legal documents from *rechtspraak.nl* were processed using custom scripts and summarized via GPT-4o-mini to create the legal summary datasets.

This strategy provided a diverse, consistently formatted general dataset and a focused legal dataset for their respective fine-tuning processes. The conversational structure aimed to improve the models’ interactive capabilities.

V. EXPERIMENTAL RESULTS

This section presents the performance of the base and fine-tuned models on Dutch language tasks, primarily summarization and instruction following. Performance is evaluated using ROUGE and BLEU metrics. Two distinct evaluation contexts are presented: one representing general-purpose content, and another focusing specifically on legal domain content. Evaluation was automated using a Python script leveraging the Ollama [19] interface and standard metric libraries (see Appendix A for snippets).

A. Performance Metrics

- **ROUGE-1 / ROUGE-2**: Unigram/bigram overlap with reference summaries/responses.
- **ROUGE-L**: Longest common subsequence overlap.
- **BLEU**: N-gram precision relative to reference summaries/responses, measuring fluency and similarity.

Scores are presented as percentages (original scores multiplied by 100).

B. Overall Performance on Mixed Training Data

The first evaluation was conducted on a sample derived from the general-purpose content dataset, comprising 125 examples from each of the four main sources (ultrachat, translated, synthetic, legal summaries) for a total of 500 samples. Table II summarizes the performance, and Figure 1 visualizes these metrics.

Analysis based on mixed training set evaluation:

- **Base Models:** Gemma3 4b Base generally outperforms LLaMA3.2 3b Base across metrics on this mixed dataset sample.

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
Gemma3 Base	36.4	11.1	17.9	8.0
Gemma3 Fine-tuned	34.2	9.5	18.0	5.8
LLaMA3.2 Fine-tuned	32.7	7.2	15.9	4.8
LLaMA3.2 Base	28.2	6.5	14.8	3.1
LLaMA3.2 Legal FT	24.4	6.9	13.4	3.6

TABLE II
OVERALL BENCHMARK SCORES ON MIXED DUTCH TRAINING DATA SAMPLES (N=500)

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
Gemma3 Base	36.5	13.1	19.6	6.2
Gemma3 Fine-tuned	34.9	12.0	19.5	6.3
LLaMA3.2 Base	27.8	7.9	15.7	3.6
LLaMA3.2 Fine-tuned	34.4	10.3	18.1	5.2
LLaMA3.2 Legal FT	**42.5**	**21.1**	**24.4**	**12.2**

TABLE III
BENCHMARK SCORES ON DUTCH LEGAL DATASET SAMPLES (N=125, EVALUATED ON TRAINING DATA)

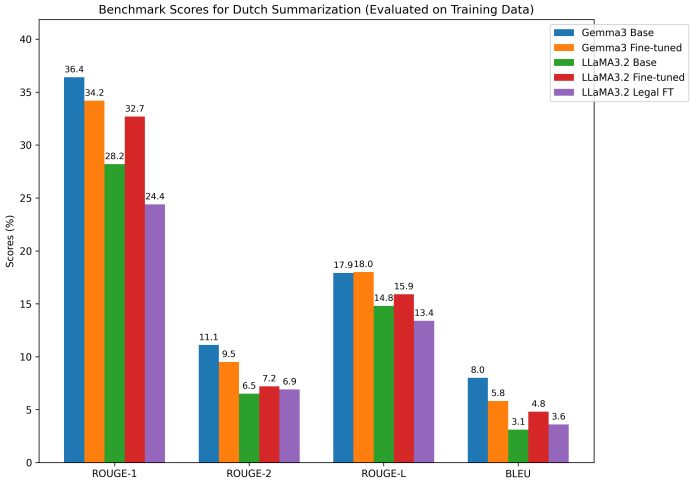

Fig. 1. Overall Benchmark Scores on Mixed Dutch Training Data Samples (N=500).

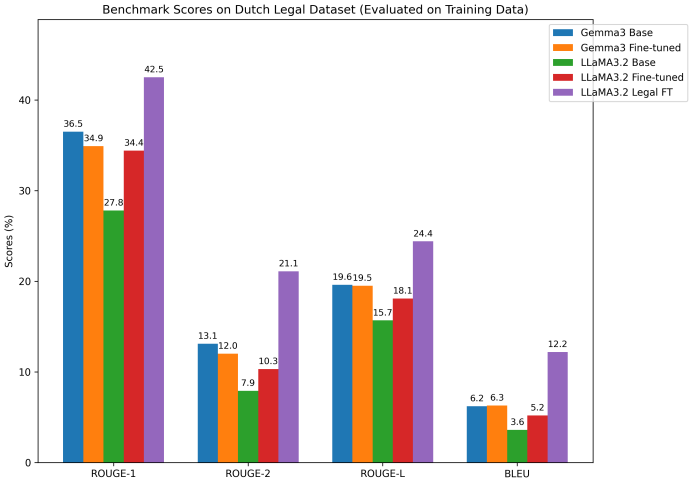

Fig. 2. Benchmark Scores on Dutch Legal Dataset Samples (N=125, Evaluated on Training Data).

- **General Fine-tuning:** LLaMA3.2 3b Fine-tuned shows clear improvements over its base version across all metrics (e.g., ROUGE-1: 28.2 to 32.7, BLEU: 3.1 to 4.8) when evaluated on data similar to its training distribution. Gemma3 4b Fine-tuned shows slightly lower scores compared to its strong base model on most metrics in this mixed evaluation, except for a marginal increase in ROUGE-L. This suggests the base Gemma3 4b was already quite capable on this data mix, and fine-tuning might require further optimization (perhaps 1 full epoch instead of 15,000 steps, or adjustments to learning rate/LoRA rank).
- **Task-Specific Model ('LLaMA3.2 3b Legal FT'):** When evaluated on this mixed dataset, the legal model shows the lowest overall scores. This is expected, as it was trained exclusively on legal text and is being tested here on a diverse set including conversational and general data, diluting its specialized strength.

C. Performance on Dutch Legal Training Data

The second evaluation focused specifically on the legal domain, using 125 samples drawn from the Dutch legal summaries dataset component. Table III and Figure 2 show the results.

Analysis based on legal training set evaluation:

- **Task-Specific Dominance ('LLaMA3.2 3b Legal FT'):** The model fine-tuned exclusively on legal data significantly outperforms all other models, including the generally fine-tuned ones and the base models, across all metrics. The gains are substantial (e.g., ROUGE-1 of 42.5 vs. 34-36 for others; BLEU of 12.2 vs. 5-6), demonstrating successful adaptation to its training domain.
- **Potential of Specialization:** This clearly demonstrates the effectiveness of task-specific fine-tuning for adapting a model to a narrow domain. The 'LLaMA3.2 3b Legal FT' model shows strong performance within the domain it was trained on. Initial qualitative tests have even suggested promising performance on unseen legal data, however, formal evaluation on a held-out legal test set was not possible due to time constraints.
- **General Models on Legal Data:** The generally fine-tuned LLaMA3.2 3b model also shows improvement over its base on legal data samples, indicating some benefit from the legal subset included in its general training mix. Gemma3 Fine-tuned remains close to its base performance on this subset, consistent with the mixed evaluation results.

D. Insights and Observations

Based on the evaluations on training data samples:

1) **Fine-tuning Efficacy:** General fine-tuning clearly improved LLaMA3.2 3b's performance on the mixed training distribution compared to its base. For Gemma3, the strong baseline performance made improvements less evident in the general mix, suggesting potential for further tuning or that the base model already generalized well to parts of our dataset.

2) **Task Specialization Value:** The legal model demonstrates the significant benefits and trade-offs of specialization. It excels dramatically when evaluated on data from its target domain (legal) but performs less well on a general, mixed dataset, highlighting the importance of training data alignment with the target application.
3) **Critical Need for Test Set Evaluation:** The lack of evaluation on a held-out test set remains a significant limitation. The current scores measure how well models have learned the training data but not how they generalize to new, unseen examples. Future work must prioritize creating and utilizing rigorous test sets (both mixed and domain-specific) to assess true generalization capabilities.

These findings illustrate the potential of PEFT for Dutch language adaptation but strongly emphasize the critical need for appropriate, domain-aligned evaluation protocols using unseen data.

VI. Discussion

Our work initially focused on building an LLM from scratch (Appendix A), providing valuable insights but proving resource-intensive for a low-resource language like Dutch. We pivoted to fine-tuning pre-trained models (Gemma3 4b, LLaMA3.2 3b) using PEFT (LoRA/QLoRA, 4-bit quantization), which proved more practical.

We explored both general fine-tuning on a diverse Dutch dataset, formatted primarily as conversational instruction-response pairs, and task-specific fine-tuning for legal summarization (‘aacudad/dutchlegal-32k‘). The results presented, however, are based on evaluating the models *on samples drawn from the data they were trained on*. This was a necessary compromise due to project time constraints which prevented the creation and validation of dedicated, held-out test datasets representative of all task types and domains.

Consequently, the reported ROUGE and BLEU scores (Tables II and III, Figures 1 and 2) reflect how well the models learned their respective training distributions but cannot reliably predict performance on new, unseen Dutch text. The observed improvements, especially for the fine-tuned LLaMA3.2 models over their base (general FT on mixed data, legal FT on legal data), suggest successful adaptation to the training data’s style and content, but generalization remains unverified. The task-specific legal model’s strong performance on its own training data subset (Table III) highlights its successful specialization, further emphasizing the need for domain-aligned evaluation, ideally on unseen data.

This limitation underscores a critical aspect of model development: robust evaluation is paramount. While fine-tuning shows promise for Dutch, future iterations must prioritize creating and utilizing proper test sets that mirror real-world usage scenarios.

For details on our initial LLM-from-scratch work, see Appendix A. For code snippets related to data generation and evaluation, see Appendix A.

VII. Ethical Considerations

Developing and deploying language models carries ethical responsibilities. Throughout this project, we aimed to adhere to best practices regarding data handling and potential model biases.

Data Bias Mitigation: We consciously assembled the general fine-tuning dataset from diverse sources (conversational, translated, synthetic across various topics, legal) to reduce the risk of skewing the model towards a narrow domain or style. The synthetic data generation process used a broad list of topics (see Appendix A) to encourage coverage of different areas. However, we acknowledge that biases present in the source datasets (like ultrachat or the models used for synthesis/translation like Gemini/GPT-4o) could still be inherited. Completely eliminating bias is a significant challenge, and ongoing vigilance and testing would be required before any real-world deployment.

Data Privacy and Usage: The legal data was sourced from *rechtspraak.nl* [18], which contains publicly available court rulings. While public, we aimed to use this data responsibly, primarily for summarization tasks. Custom scripts were used for extraction and processing, and care was taken not to store or process any information that could be considered sensitive personal data beyond what is already public in the rulings. No other private or personal data was intentionally collected or used. Our data handling practices aim to align with the principles of data minimization and purpose limitation, respecting the spirit of regulations like the GDPR and the developing EU AI Act regarding responsible data use and transparency. We did not scrape websites in a way that would violate terms of service or compromise private information.

Exclusion and Discrimination: We strived to create datasets and fine-tune models that are broadly useful for Dutch speakers. The diversity in the general dataset was an attempt to avoid overly niche models (unless specifically intended, like the legal model). However, the limitations of available Dutch data and potential inherent biases mean that the models might still perform unevenly across different demographic groups or dialects. Further testing and potentially targeted data augmentation would be needed to address specific concerns about fairness and inclusivity.

We believe these efforts represent a responsible approach within the scope of this research project, but acknowledge that continuous ethical assessment is crucial for LLM development.

VIII. Conclusion

This work explored adapting pre-trained LLMs (Gemma3 4b, LLaMA3.2 3b) for Dutch tasks using parameter-efficient fine-tuning (PEFT). We investigated both general adaptation on a diverse, conversationally formatted dataset and task-specific fine-tuning for legal summarization. Our methodology leveraged LoRA/QLoRA and 4-bit quantization for resource efficiency.

Experimental results, **evaluated solely on samples from the training data due to time limitations**, suggest that fine-tuning can adapt models to the target language and tasks present in the training distribution. LLaMA3.2 3b showed clear

improvement over its base on the mixed training set after general fine-tuning. The task-specific legal model demonstrated strong specialization, significantly outperforming other models on samples from its legal training data, highlighting the potential for domain-specific adaptation but also the performance trade-off on out-of-domain data.

While the initial plan to build an LLM from scratch was insightful, fine-tuning proved more feasible. However, the lack of evaluation on a held-out test set is a major caveat; the reported scores indicate learning capacity on the training distribution but not generalization to unseen data. Ethical considerations were addressed through diverse data sourcing and careful handling of public legal data, aiming to minimize bias and respect privacy.

In summary, this project highlights the potential of PEFT for low-resource languages like Dutch and demonstrates the effectiveness of task-specific tuning using appropriately formatted data. It also strongly emphasizes the necessity of rigorous evaluation on unseen data in future work. Further exploration should involve creating appropriate test sets, refining fine-tuning strategies (e.g., hyperparameter optimization, exploring different PEFT ranks), and potentially incorporating human feedback to validate and improve real-world performance for both general and specialized Dutch LLMs.

APPENDIX

LLM FROM SCRATCH: INITIAL EXPERIMENTS

In the early stages of this project, our approach focused on constructing a large language model (LLM) from the ground up. This appendix provides a technical overview of the methods and design decisions implemented during these initial experiments.

Our methodology began with tokenization, where raw text data was converted into a sequence of tokens using a specialized encoding scheme. We used the tiktoken library [20] to perform tokenization, which allowed for efficient and consistent mapping of text to token IDs. This step was essential for ensuring compatibility with model input requirements and maintaining alignment with the vocabulary size dictated by the selected base model. Token counting and encoding were implemented following best practices outlined in the OpenAI Cookbook [20], enabling precise control over context window limits and dataset sizing.
The core of the model architecture was inspired by the seminal work "Attention Is All You Need." [5] We implemented a decoder-only transformer architecture, which consisted of the following key components:

- **Token and Positional Embeddings**: These layers transform token indices into dense vector representations while integrating positional information. This combination is crucial for capturing the order of tokens in a sequence.
- **Transformer Encoder Layers as Decoders**: Although originally designed for encoding, transformer encoder layers were repurposed to function as decoders. Each layer comprised multi-head self-attention with causal masking, ensuring that each token could only attend to preceding tokens, and a subsequent point-wise feedforward network to refine the representations.
- **Final Normalization and Projection**: A concluding layer normalization followed by a linear projection transformed the hidden states into logits over the vocabulary, facilitating next-token prediction.

The process of data formatting involved preparing an expanded text dataset. This was achieved by:

- Processing raw text inputs and applying tokenization.
- Truncating or padding sequences to a defined maximum length to ensure uniformity across batches.
- Creating training samples where each input sequence is paired with a target sequence, shifted by one token, to enable next-token prediction.

The training loop employed standard optimization techniques, utilizing a cross-entropy loss function to measure prediction error and an Adam optimizer for parameter updates. Training was conducted over multiple epochs, with the model iteratively learning to predict subsequent tokens in the sequence.

Finally, the model was evaluated using an autoregressive generation process, where a prompt is provided and the model iteratively predicts additional tokens.

In summary, our initial experiments in building an LLM from scratch provided critical insights into the challenges of training such models, particularly in terms of resource allocation and model design. Although this approach laid a strong foundation, it ultimately highlighted the efficiency gains achievable through fine-tuning pre-trained models—a strategy that forms the basis of our subsequent work.

APPENDIX

CODE AVAILABILITY

All code used in this project, including scripts for data generation, preprocessing, model fine-tuning (using Unsloth/PEFT), and evaluation, is openly available in the following GitHub repository:

https://github.com/aacudad/DutchGPT [21]

This codebase supports reproducibility of the experiments described in this report. Key scripts referenced in Appendix A can be found within this repository.

APPENDIX

PROJECT REFLECTION

We began this project aiming to build an LLM from scratch, diving deep into the *Attention Is All You Need* paper [5]. This foundational work was crucial but challenging to implement practically, especially given resource limitations and the lack of extensive Dutch training data. Our initial prototype confirmed our understanding but highlighted the infeasibility of scaling this approach within the project scope.

Transitioning to fine-tuning pre-trained Gemma3 and LLaMA3.2 3B models using PEFT (Unsloth, LoRA, QLoRA) proved far more effective. This required learning new techniques like quantization and adapter tuning but significantly accelerated progress. Data preparation remained a hurdle; assembling, cleaning, formatting into a conversational structure, and merging 290k general instructions and a separate 100k legal summary dataset involved considerable effort with API usage, scripting, and balancing. We also created a task-specific model (‘aacudad/dutchlegal-32k‘) to explore domain adaptation.

A key challenge emerged late: time constraints prevented us from creating a proper held-out test set. Consequently, all reported evaluations were performed on samples drawn from the training data. While this showed the models learned the training distribution (especially the fine-tuned LLaMA variants compared to base, and the legal model on its domain-specific data), it doesn't guarantee generalization. This limitation was a significant practical constraint we had to acknowledge clearly.

Collaboration was vital, requiring clear roles and good code management practices using GitHub. Despite the evaluation caveat, the project provided extensive hands-on experience with the LLM lifecycle, from theory to data preparation, fine-tuning, domain specialization, and recognizing the critical importance of validation data distinct from training data.

Key Takeaways

- Foundational theory is essential, but practical implementation reveals true challenges.
- Fine-tuning with PEFT is a viable strategy for low-resource languages under constraints.
- Task-specific fine-tuning can yield significant performance gains within a domain, but requires domain-aligned data.
- Data quality, diversity, formatting (e.g., conversational), and quantity are critical, but balanced datasets are hard to curate.
- Efficient tooling (Unsloth, LoRA, Hugging Face libraries) is invaluable for resource-limited development.
- **Rigorous evaluation on unseen test data is non-negotiable for assessing true model performance; evaluating on training data shows learning capacity on that data but not generalization.**
- Ethical considerations regarding data sourcing, bias, and privacy are integral to responsible LLM development.

Overall, this project provided a deep dive into LLM development, enhancing our ability to train, fine-tune, and critically assess models, including awareness of their limitations and the importance of evaluation methodology, especially regarding domain specificity and the training/test data split. The groundwork supports future efforts towards more robust Dutch LLMs, contingent on establishing proper testing procedures and continued ethical reflection.

APPENDIX

DATA GENERATION AND EVALUATION SCRIPT SNIPPETS

This appendix provides illustrative snippets from the Python scripts used for synthetic data generation and model evaluation, as referenced in the main text. The full scripts are available in the project's GitHub repository [21].

Synthetic Data Generation (using Gemini API)

The script utilized the Gemini API to generate diverse Dutch instruction-response pairs based on a predefined list of topics. It operated in batches and formatted the output into the JSON structure shown in the Data Collection section.

```python
# Snippet from synthetic data generation script

import json
import random
import hashlib
from google import genai # Assuming google.genai client is initialized

# Example subset of topics used
TOPICS = [
    "online winkelen", "klantenservice", "duurzaam winkelen",
    "technische ondersteuning", "cybersecurity", "hobbyprojecten",
    "fotografie tips", "tuinieren", "huisrenovatie", "koken en recepten",
    "gezondheid en welzijn", "mentale gezondheid", "reisadviezen",
    "financieel advies", "budgetteren", "opvoedingstips", "carrièreplanning",
    "huisdierenverzorging", "milieuproblematiek", "sportactiviteiten",
    # ... many more topics ...
    "bullet journaling", "AI-tools gebruiken", "timemanagement",
    "band vervangen", "rugpijn voorkomen", "duurzaam poetsen",
    "studeren met flashcards", "roadtrip plannen", "burgerparticipatie",
    "kruiden kweken", "zelf bier brouwen"
]

# Core function (simplified) to generate pairs for a batch of topics
def generate_ir_pairs_batch(topics: list[str], model_name: str) -> list[dict]:
    topics_str = ", ".join(topics)
    # System prompt instructs the model on generation format and style
    system_prompt = '''Je bent een assistent die helpt bij het genereren van Nederlandstalige
    ↪  instructiedata...
    Genereer een natuurlijke instructie... Varieer actief in toon...
    [... Full system prompt guiding generation ...]'''

    contents = f"{system_prompt}\n\nGenereer instructie-respons paren voor: {topics_str}."

    # Using Gemini API with JSON mode expected
    try:
        response = client.models.generate_content( # client assumed initialized
            model=model_name,
            contents=contents,
            config={
                'response_mime_type': 'application/json',
                # 'response_schema': Defined Pydantic model for structure
            },
        )
        # Process parsed JSON response (error handling omitted for brevity)
        parsed_response = response.parsed # Assuming direct parsing works
        # Convert to list of dicts if needed
        return [pair.dict() for pair in parsed_response.pairs]
    except Exception as e:
        print(f"Error during generation: {e}")
        return []

# Main loop (simplified structure)
# for batch_num in range(total_batches):
#     batch_topics = random.sample(TOPICS, batch_size)
#     ir_pairs = generate_ir_pairs_batch(batch_topics, "gemini-1.5-flash-latest")
#
#     # Format each pair into the target JSON structure
#     for pair_data in ir_pairs:
#         instruction = pair_data["instruction"]
#         response = pair_data["response"]
#         # Create the final entry structure with hash and conversations list
#         entry = { ... }
```

```
#         # Append entry to file (using append_to_jsonl function)
```

Data Merging Script

A simple script was used to combine multiple ‘.jsonl‘ files (potentially generated by different processes or APIs) into a single Hugging Face compatible ‘.json‘ file, with an option to remove duplicates based on the pre-calculated hash.

```
# Snippet from data merging script

import json
import os
from tqdm import tqdm

def combine_jsonl_files_to_huggingface(input_files: list[str], output_file: str, remove_duplicates: bool =
↪  True):
    all_entries = []
    unique_hashes = set()
    duplicates_found = 0

    for input_file in input_files:
        print(f"Processing: {input_file}")
        try:
            with open(input_file, 'r', encoding='utf-8') as f:
                for line in tqdm(f, desc=f"Reading {os.path.basename(input_file)}"):
                    if line.strip():
                        entry = json.loads(line)
                        if remove_duplicates:
                            entry_hash = entry.get("hash")
                            if entry_hash and entry_hash in unique_hashes:
                                duplicates_found += 1
                                continue
                            if entry_hash:
                                unique_hashes.add(entry_hash)
                        all_entries.append(entry)
        except Exception as e:
            print(f"Error processing {input_file}: {e}")

    # Format for Hugging Face (often the input format is already compatible)
    formatted_data = all_entries # Assuming format matches target

    # Write to output JSON file
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(formatted_data, f, ensure_ascii=False, indent=2)

    print(f"Combined {len(input_files)} files. Total entries: {len(formatted_data)}. Duplicates removed:
    ↪  {duplicates_found if remove_duplicates else 'N/A'}.")

# Example usage:
# input_files = ["gen_data_1.jsonl", "legal_data_sample.jsonl", ...]
# combine_jsonl_files_to_huggingface(input_files, "combined_dataset.json")
```

Model Evaluation Script (using Ollama and Metrics Libraries)

The evaluation script queried models hosted via an Ollama server, compared their responses against ground truth data loaded from JSON files, and calculated ROUGE and BLEU scores. It supported parallel evaluation across multiple models.

```
# Snippet from model evaluation script

import json
import numpy as np
from openai import OpenAI # Used to interface with Ollama's OpenAI-compatible API
from rouge_score import rouge_scorer
import nltk
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from tqdm import tqdm
import pandas as pd
import concurrent.futures

# Initialize Ollama client
client = OpenAI(
    base_url='http://localhost:11434/v1', # Default Ollama URL, adjust if needed
```

```
    api_key='ollama', # Required placeholder
)

# Function to get response from a model via Ollama
def get_model_response(model_name, query):
    try:
        response = client.chat.completions.create(
            model=model_name,
            messages=[{"role": "user", "content": query}],
            max_tokens=1024 # Limit response length
        )
        return response.choices[0].message.content
    except Exception as e:
        print(f"Error querying {model_name}: {e}")
        return "Error: Could not get response"

# Function to calculate ROUGE scores
def calculate_rouge(prediction, reference):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    scores = scorer.score(reference, prediction)
    return {
        'rouge1': scores['rouge1'].fmeasure,
        'rouge2': scores['rouge2'].fmeasure,
        'rougeL': scores['rougeL'].fmeasure
    }

# Function to calculate BLEU score
def calculate_bleu(prediction, reference):
    reference_tokens = [nltk.word_tokenize(reference.lower())] # List of reference token lists
    prediction_tokens = nltk.word_tokenize(prediction.lower())
    smoothie = SmoothingFunction().method1
    return sentence_bleu(reference_tokens, prediction_tokens, smoothing_function=smoothie)

# Function executed in parallel for each model-sample pair
def process_model_response(model, query, ground_truth, dataset_source):
    prediction = get_model_response(model, query)
    rouge_scores = calculate_rouge(prediction, ground_truth)
    bleu_score = calculate_bleu(prediction, ground_truth)
    return {
        'model': model, 'prediction': prediction, 'dataset': dataset_source,
        **rouge_scores, 'bleu': bleu_score # Combine dicts
    }

# Main evaluation loop structure (simplified for parallel execution)
# test_data = load_test_data(...) # Load list of {'query': ..., 'ground_truth': ...}
# results = {model: {'rouge1': [], ...} for model in models}
# all_responses_details = []

# with concurrent.futures.ThreadPoolExecutor(max_workers=...) as executor:
#     futures = []
#     for sample in test_data:
#         for model in models_to_test:
#             futures.append(executor.submit(process_model_response,
#                                            model, sample['query'],
#                                            sample['ground_truth'],
#                                            sample['dataset']))
#
#     for future in tqdm(concurrent.futures.as_completed(futures), total=len(futures)):
#         result = future.result()
#         # Append result metrics to the correct model's list in 'results' dict
#         # Store detailed response in 'all_responses_details'

# After loop: Calculate average scores from 'results' and create summary DataFrame
```

REFERENCES

[1] Gemma Team and Google DeepMind, "Gemma 3 Technical Report," Tech. Rep., 2025. [Online]. Available: https://storage.googleapis.com/deepmind-media/gemma/Gemma3Report.pdf.

[2] Meta Platforms, Inc., *Get started with llama*. [Online]. Available: https://ai.meta.com/llama/get-started/.

[3] E. Hu *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[4] T. Dettmers *et al.*, "Qlora: Efficient fine-tuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.

[5] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Adv. neural information processing systems*, vol. 30, 2017.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] T. B. Brown *et al.*, "Language models are few-shot learners," *Adv. neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[8] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[9] S. Gururangan *et al.*, "Don't stop pretraining: Adapt language models to domains and tasks," *ACL*, 2020.

[10] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[11] S. Longpre *et al.*, "The flan collection: Designing data and methods for effective instruction tuning," *arXiv preprint arXiv:2301.13688*, 2023.

[12] T. Dao *et al.*, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *arXiv preprint arXiv:2205.14135*, 2022.

[13] Unsloth AI, *Unsloth documentation*. [Online]. Available: https://docs.unsloth.ai.

[14] B. Vanroy, *Geitje 7b ultra: A conversational model for dutch*, 2024. arXiv: 2412.04092 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2412.04092.

[15] A. Acudad, *8k_dutch_nemotron_translation*. [Online]. Available: https://huggingface.co/datasets/aacudad/8K_DUTCH_NEMOTRON_TRANSLATION.

[16] A. Acudad, *86k_dutch_conversational*. [Online]. Available: https://huggingface.co/datasets/aacudad/86k_DUTCH_conversational.

[17] A. Acudad, *5k_dutch_legal_summary*. [Online]. Available: https://huggingface.co/datasets/aacudad/5K_DUTCH_LEGAL_SUMMARY.

[18] De Rechtspraak, *Rechtspraak.nl: De officiële website van de nederlandse rechtspraak*. [Online]. Available: https://www.rechtspraak.nl.

[19] Ollama Team, *Ollama: Run llama 2, code llama, and other models locally*. [Online]. Available: https://ollama.com/.

[20] OpenAI, *How to count tokens with tiktoken*, 2023. [Online]. Available: https://cookbook.openai.com/examples/how_to_count_tokens_with_tiktoken.

[21] A. Acudad, *Dutchgpt: Fine-tuning llms for dutch*. [Online]. Available: https://github.com/aacudad/DutchGPT.