POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Privacy-preserving transformer encoder for Automatic Speech Recognition

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Mark Federico Zampedroni

Advisor: Prof. Pier Luca Lanzi

Co-advisor: Dot. Ing. Andrea Cola

Academic year: 2022-2023

1. Introduction

This study is aimed to the development of a privacy-preserving transformer-based encoder for Automatic Speech Recognition (ASR), to employ for the extraction of an acoustic context from spoken audio recordings. The model ensures the user's anonymity by concealing the input values and results through the use of homomorphic encryption (HE) techniques. Subsequently, the extracted acoustic context can be given to a decoder to complete the ASR process and derive the audio transcription.
The work unfolds into two parallel challenges; on one hand, due to the limitations of HE, it is necessary to first develop an encoder that operates solely with additions and products. On the other hand, such encoder is integrated with a HE scheme, for which a new packing method is proposed, specifically designed to be compatible with transformers; aiming to reduce memory requirements and significantly increase the speed of the inference process. In addition to the final solution, a key result of this work is the successful application of the proposed packing method, which enables the possibility to work with matrices of dimensions that are generally considered prohibitive when using HE.

2. Baseline architecture

A transformer-based ASR encoder has to be operated within a larger end-to-end architecture, that complements it with an embeddings extractor and a decoder. As a baseline architecture to train and test the privacy-preserving encoder, we chose the current state-of-the-art for multitask and multilanguage ASR, *Whisper* [1]; which original encoder is modified to reach a solution compatible with HE. Before the integration with HE the complete end-to-end ASR processing is conducted as follows: the raw audio data is given to the embeddings extractor and splitted into 30 seconds chunks. An 80-channel log-mel spectrogram representation is computed on 25ms windows with a stride of 10ms. These features are then normalized and processed by two convolution layers, and the result is added to sinusoidal positional embeddings. The acoustic embeddings found are then processed by a number of encoder blocks to find an acoustic context, a rich representation of the acoustic content of the original input speech data. Such context is then employed by a sequence of decoder blocks that autoregressively reconstruct the speech transcription.

3. Speech datasets

The selection of an appropriate dataset for training, validating and testing is a pivotal decision in the development process. Since this work focus is on ASR, the dataset has to contain segments of speech audio data in the form of audio samples, which are individual amplitude values representing the sound wave at specific points in time, and a transcription of the audio contents; used as reference to compute the loss function and to assess the final results. Moreover, several factors warrant consideration when making the choice of which dataset to adopt: its samples have to be in english, and have good data quality, high speaker variability, variable background noise and transcripts provided by humans. Since it is difficult to find a verified dataset with all these characteristics, in this work two different candidates were selected: the *Common Voice* dataset, comprised of more than 1500 hours of audio, and the *GigaSpeech* dataset, containing 2500 hours. These two datasets complement each other, ensuring at least some degree of variety: *Common Voice* reports demographic information about the speakers (i.e. gender, age and accent), while *GigaSpeech* is focused on providing transcriptions from different domains, and recordings in various environments: audiobooks, podcasts and YouTube videos segments. In fact, in this instance, the acoustic condition of the sample is related to the audio source. Audiobooks do not contain much noise, podcasts are usually indoors, more spontaneous and sometimes with background music. Audio extracted from YouTube may have been recorded outdoors, or in a noisy environment. After these considerations, both datasets were preprocessed and merged: the recordings, resampled to 16kHz, were stored as their log-mel spectrograms, and the transcriptions were normalized to a shared textual format. Overall, the inhomogeneity of the resulting dataset samples helps the model to generalize during training, learning from vocal features of both genders, all ages and in many types of environments.

4. Approximated encoder

To reach a privacy-preserving solution through HE, due to its limitations, it is first necessary to build an encoder that operates by using only additions and products. Starting from the original *Whisper* [1] encoder, the SoftMax, Layer Normalization and GELU activation functions are replaced with polynomial approximations or removed. Furthermore, due to the limited number of products computable on the HE ciphertexts, the encoder block has to be reduced in depth, while maintaining as much accuracy as possible.

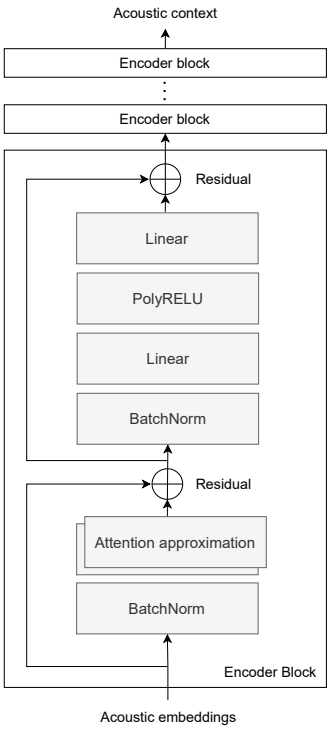

Figure 1: Final solution of the fully polynomial encoder. It approximates the *Whisper* encoder.

4.1. Training and validation

The training and validation process follows a structured approach: the architecture is initialized using the *Whisper* (the *tiny* version with 4 encoder and 4 decoder blocks) checkpoint available on the *HuggingFace* API, which encoder is replaced with the approximated version being tested. To preserve some of the knowledge inherited from the original model, a fine tuning strategy is implemented. This involves freezing the embeddings extractor and decoder during the training process, allowing adjustments solely to the new encoder. When the validation loss has plateaued, or no longer improves significantly, the fine tuning is halted, and the previously frozen modules are unfrozen. After which, further refinement is achieved by continuing the training with a lower learning rate.
The tested configurations are all built and

trained using the PyTorch framework; the optimizer is the Adaptive Moment Estimation with Weight Decay (adamW) algorithm and the loss function used is the categorical cross-entropy, seeing the Automatic Speech Recognition (ASR) task as a classification of each inferred token over all the possible tokens. The training parameters are $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$, with a weight decay of 0.1. The learning rate is scaled dynamically, starting from 10^{-5} and 10^{-4}, respectively for the fine tuning and model refinement, with a warm up period of 500 training steps. The batches are of 16 samples, and no gradient accumulation is used. During the experiments it is noticeable the tendency of the model to overfit on the training data when fine tuning, thus the process is constantly monitored throughout the training using an early stopping callback, and a dropout with 5% probability is added within the attention module. Furthermore, to select the best layers approximations, the Word Error Rate (WER) is computed on the detokenized results during the validation.

4.2. Encoder block

Within each original *Whisper* encoder block the layers listed in section 4 have to be replaced with polynomial applications in order to enable the adoption of HE. After the consideration and testing of many alternatives, the final solution blocks in figure 1 replaced the GELU activation function with the RELU polynomial approximation $0.563059 + 0.5x + 0.078047x^2$, as suggested in [4]. The layer normalizations were changed to batch normalizations, which at inference time use constant values for the mean and variance, later allowing the privacy-preserving encoder to pre-compute and encode its coefficients in a way that only additions and products are employed. While these two changes attained a transcription accuracy similar to the original model, finding an alternative to the softmax of the attention mechanism was challenging; the best WER of 36.91 was reached by using a RELU polynomial approximation followed by a batch normalization layer, which, in this instance, partially reproduces the behavior of the softmax. However, this solution adds a depth of 4 multiplicative levels to the model, which when later integrating with HE becomes excessive. For this reason, only the batch normalization was adopted, adding 1 multiplicative level and attaining a final WER of 40.70. Moreover, in this final solution, the query Q and key K projections used in the self-attention computation were constrained to be equal, which does not incur in a loss of accuracy and saves one product. The total required multiplicative depth, before the folding in the next section, is 11 per encoder block, plus 1 for the last batch normalization; a total of 45.

4.3. Folding

Some layers of the fully polynomial encoder obtained (ref. figure 1) can be merged (i.e. folded) and reformulated into an equivalent form to reduce the total multiplicative depth, and later boosting the model performance when working using HE operations. In fact, a fully connected (FC) layer with a weights matrix W and bias b following a polynomial activation of the form $i + jx + kx^2$ can be rewritten as $Activ(FC(x)) = i + \frac{j}{\sqrt{k}}(x\sqrt{k}W + \sqrt{k}b) + (x\sqrt{k}W + \sqrt{k}b)^2$. The new coefficients can be pre-computed and stored, reducing their combined number of sequential products at inference time from 3 to 2. Similarly, a FC layer can be folded when preceded by a Batch Normalization $BN(x) = \gamma\frac{x-\mu_B}{\sqrt{\sigma_B^2+\epsilon}} + \beta$ where γ, μ_B, σ_B, ϵ and β are all constants during inference: $FC(BN(x)) = x(\frac{\gamma W}{\sqrt{\sigma_B^2+\epsilon}}) + (b + \beta W - \frac{\gamma\mu_B W}{\sqrt{\sigma_B^2+\epsilon}})$; which also reduces the multiplicative depth by 1. Applying this concept wherever possible allows to save 3 multiplicative levels per encoder block, lowering the overall depth from 45 to 33.

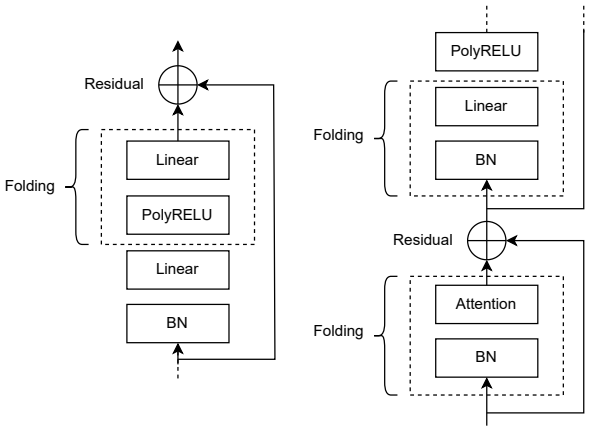

Figure 2: Folding of the layers in figure 1, on the left $Activ(FC(x))$, on the right $FC(BN(x))$.

5. HE integration

The approximated encoder constructed in section 4 operates on plain data, but since it employs only additions and multiplications, it can be integrated with Homomorphic Encryption (HE) to obtain a privacy-preserving solution. This integration does not require any further training; the model layers remain unchanged, and it suffices to adopt an HE scheme for the computations: the scheme we chose for this work is the Cheon-Kim-Kim-Song (CKKS), a Leveled Homomorphic Encryption (LHE) scheme [3]. CKKS can encode an entire vector of complex or real numbers into a single plaintext, where each element is stored into a slot, and the operations between CKKS plaintexts and ciphertexts are done at a slot-wise level, enabling the possibility to do HE computations in a SIMD fashion. This is referred to as packing, and vastly reduces the online computational time required for the inferences.

5.1. Packing method

An input matrix is usually packed by rows or columns, and then encrypted; depending on this choice it changes the way the matrix multiplication can be computed by leveraging the slot-wise parallelism. Currently, the best state-of-the-art matrix multiplication algorithms [2] implemented for HE are the algorithms proposed by Halevi and Shoup and Jiang et al. Unfortunately, they are not as efficient when working with transformer-based models, since their packing cannot be easily employed for the multihead attention computation. For this reason, this work proposes a column-wise diagonal packing method and the procedures to compute matrix multiplications; to overcome the multihead computations problem two algorithms are defined, one splits an input matrix into submatrices, the other concatenates multiple matrices to obtain a single matrix.

Algorithm	Operations	Levels
CipherPlainMatMul	m^2	1
CipherCipherMatMul	m^2	1
SplitHeads	$m \cdot h$	1
JoinHeads	$m \cdot h$	1

Table 1: Defined algorithms, m is the dimension of the input, and h is the number of heads.

Each of these operations requires one multiplicative level, adding two levels to the multiplicative depth of each encoder block, due to the multihead matrices management, which in the plain version was not needed, rising the total multiplicative depth from 33 to 41. The performance tests are conducted using the scheme parameters later defined, and run on a local machine with a single AMD Ryzen 5 PRO 5650U - 6 CPU cores and 16 GB of RAM. They compare the proposed packing against the commonly used approach when working with transformers and HE: a naive encoding where each ciphertext contains only one element instead of a packed vector.

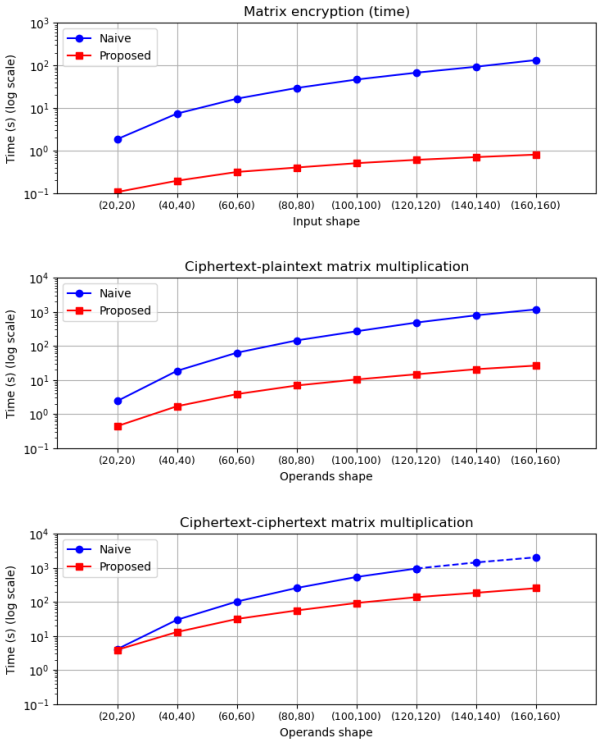

Figure 3: Computational times comparison.

Moreover, encoding a matrix of shape (n, m) using the naive method requires nm ciphertexts, while the proposed packing only m. The most commonly matrix shape used in the approximated encoder is $(1500, 384)$: if naively encrypted it would need 576.000 ciphertexts and 301.95GB of memory, if packed by diagonals it is reduced to 384 ciphertexts and 0.2GB.

5.2. Multiprocessing

In order to further optimize the computational times of the privacy-preserving encoder, the algorithms proposed in table 1 can be executed

using multiple processes. Their operands are accessed in a read-only manner, and each process can compute a portion of the output ciphertexts, containing the diagonals of the corresponding matrix. Following the defined approach for each procedure, they are tested to find the best trade-off between the number of processes and the overhead, using the same parameters and machine as the tests in section 5.1. SplitHeads and JoinHeads do not particularly benefit from an increased number of processes, but the matrix multiplications times are reduced to a third:

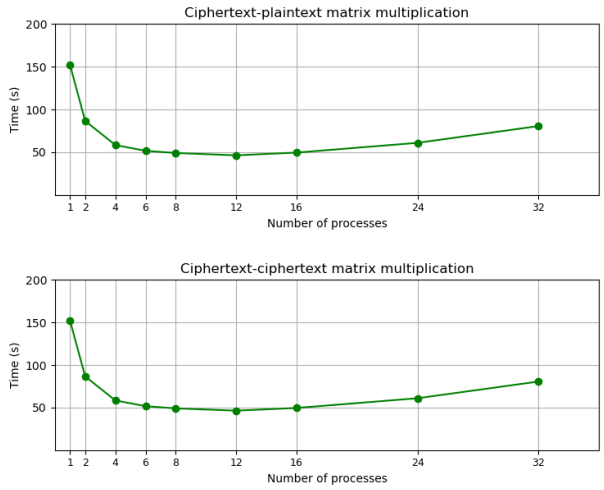

Figure 4: Parallelized matrix multiplications using operands that, before packing, are of shape (1500, 384) and (384, 384).

5.3. Scheme parameters

Accurately choosing the CKKS scheme parameters is essential in order to obtain an acceptable result from the encoder, and within a reasonable time. These parameters both influence the correctness and the performance of the computations, providing a trade-off between speed, precision, multiplicative depth allowed, memory and security. The adjustable parameters are the polynomial degree N, the scale Δ and the modulus chain q; the security level is $\lambda = \frac{N}{q_L}$, that in this work is fixed at minimum 128-bits. The proposed packing method is sparse and requires at least $N > 6000$ to work for the encoder use case; eventually $N = 8192$ is chosen, after some considerations where it is compared to $N = 16384$. The modulus chain q primes are defined such they are $\approx \Delta$, and the maximum depth L can be computed as $L = \lfloor \frac{218}{log_2(\Delta)} \rfloor$, where 218 is the maximum total size of the modulus chain for $\lambda = 128$-bits. To analyze the impact of the precision Δ on the privacy-preserving encoder, some tests are conducted by running inferences (using only one process), and the values reported are an average of the results over the four blocks. Note that since the privacy-preserving encoder consumes 41 multiplicative levels, it is necessary to refresh the ciphertexts every L products by sending them to the user holding the private key for decryption and re-encryption:

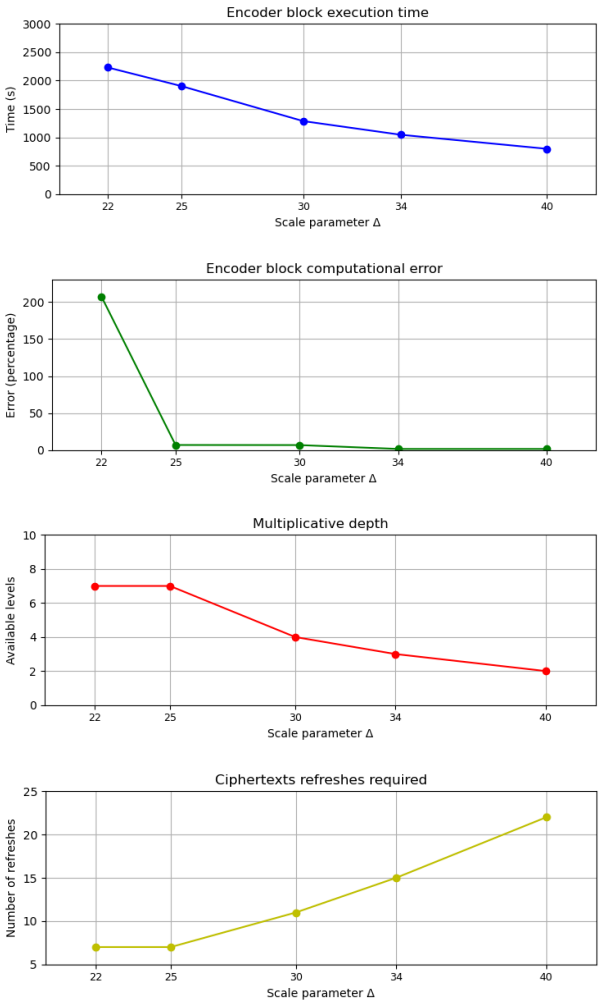

Figure 5: Effects of the scale parameter Δ.

The results have an acceptable numerical error for Δ between 2^{25} and 2^{40}. The ideal value varies depending on the tolerable number of communications between the holder of the private key and the entity hosting the model. If there is a need to communicate as little as possible then $\Delta = 2^{25}$ is the best choice, but it also provides the slowest and least accurate inference. If communicating is not an issue, then, it is necessary to find the scale with the least execution time

+ communication overhead, which depends on the connection speed. In this work, to run the other tests, the precision scale is arbitrarily chosen to compromise between the two scenarios: $\Delta = 2^{30}$, meaning $L = 4$, for which the ciphertexts have to be refreshed 11 times.

6. Results

After the integration with HE, the obtained privacy-preserving encoder is tested, employing the proposed packing method and the multiprocessing variants of the matrix multiplications. The required time and the resulting WER are compared against the approximated fully polynomial encoder found in section 4, which is the same model but working on plain data, and the original fine-tuned *Whisper* architecture.

Encoder	Time (s)	WER
Privacy-preserving	3992.0	44.54
Fully polynomial	0.009	39.62
Whisper	0.011	22.78

Table 2: Computational time and WER comparison.

A complete end-to-end execution of the privacy-preserving encoder, requires, on average, approximately 1 hour, 6 minutes and 32 second, and its WER increases by 4.92 compared to its counterpart computing on plain data, because of the errors introduced by the encryption scheme. The average computational time of each operation type during one inference is visualized in the following figure:

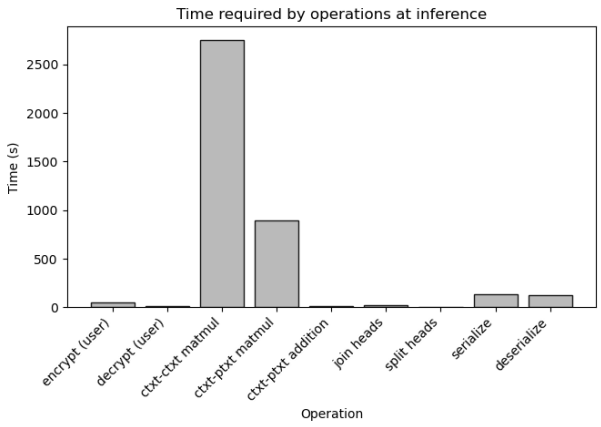

Figure 6: Total time required by each operation type during one inference.

7. Conclusions

In conclusion to this work, some considerations are necessary. First of all, the approximated polynomial encoder, constructed in section 4, in the final test attained a WER of 39.62, which is not remarkable; commercial solutions for ASR usually have a WER lower than 30. Nonetheless, since the approximation uses only additions and products, and the replacement of the softmax in the attention mechanism is currently an open problem, it can be considered arguably satisfying. On the other hand, the results regarding the proposed packing method and the parallelization are more rewarding; it is compatible with the transformer multihead attention module, and by leveraging the existing properties of the CKKS scheme it drastically reduces both the computational time and the memory requirements of the HE operations. While the objectives of this thesis were addressed successfully, one major inconvenience with the developed solution is that the user, holding the private key of the encryption scheme, has to cooperate with the entity hosting the privacy-preserving encoder to refresh the ciphertexts and allow the computation to continue, making it impractical for some real-world use cases. A future work might address this point by testing the impact of adopting the bootstrapping operation, allowing to avoid mid-inference communication between the two parties.

References

[1] A. Radford et al. Robust speech recognition via large-scale weak supervision, 2022.

[2] Babenko et al. A comparative study of secure outsourced matrix multiplication based on homomorphic encryption. *Big Data and Cognitive Computing*, 7(2):84, 2023.

[3] J. Cheon et al. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017*, pages 409–437. Springer, 2017.

[4] T. Ishiyama et al. Highly accurate cnn inference using approximate activation functions over homomorphic encryption, 2020.