Safe Multi-Party Semantic Agreement via FHE Word Embeddings

Maria Chrysafis
Massachusetts Institute of Technology
m4c@mit.edu

David Hu
Massachusetts Institute of Technology
davidhu@mit.edu

Miguel Tulla
Massachusetts Institute of Technology
mtulla@mit.edu

Edward Wan
Massachusetts Institute of Technology
edw21@mit.edu

1. Abstract

This paper presents an approach towards a multi-party semantic agreement protocol constructed using fully homomorphic word embeddings. It is not difficult for two or more parties to compare text for exact equality without revealing their text—the parties can simply compare a one-way hash of their samples through some suitable protocol. However, we aim to construct a protocol that relaxes the condition of exact equality into one of "semantic agreement."

In our approach, we quantify semantic agreement by comparing the word embedding values—as determined in LLMs (large language models)—of the party's text. We then employ pre-existing FHE (fully homomorphic encryption) computation libraries to compute word embeddings and semantic agreement on the ciphertext. By encrypting the entire end-to-end computation from text to embedding via FHE, our scheme ensures that sensitive information (even including the embeddings themselves) is not leaked during computation. Ultimately, we can securely gauge semantic agreement of samples of text by employing FHE in computing the dot product of two word embeddings to calculate their *cosine similarity*.

Our results demonstrate that FHE introduces minimal noise to the word embeddings, preserving the semantic information effectively. Despite the increased computational complexity of computing word embeddings using FHE compared to traditional methods, the resulting agreement protocol operates smoothly and efficiently.

Keywords: Multi-party computation, Semantic agreement, Fully homomorphic encryption, Word embeddings, Cosine similarity, Large Language Models.

2. Motivation

There are many situations where multiple parties would want to ascertain the similarity between individually produced texts without revealing said texts to each other. For instance, take the case of two parties determining whether they should collaborate when valuable ideas or intellectual property is involved.

Suppose two parties both have solutions or ideas for an open-ended problem, and they would like to determine if collaborating is in each of their best interest. Naturally, neither wants simply reveal their idea to the other for fear of it being replicated without acknowledgment and compensation. However, collaboration could be much more productive compared to each of them pursuing similar ideas independently. Our protocol provides a way for these parties to determine if collaboration is in their best interest by securely determining the semantic similarity of text that describes their ideas in detail.

In general, a secure semantic agreement protocol proves useful between a set of parties when the following conditions are met:

1. *Requirement for Secrecy*: It is disadvantageous for Party A to plainly share its text with Party B if Party B does not already have some knowledge that would allow it to produce text that could be considered similar to that of Party A.

2. *Bilateral Cost of Secret Leaking*: Condition 1 must be both ways; i.e., it must also be disad-

vantageous for Party B to plainly share its text to Party A.

3. *Mutual Advantage in Agreement*: There exists something mutually advantageous—be it collaboration or simply sharing knowledge—for Party A and Party B if they both have knowledge that prompts them to produce similar text.

3. Background on Fully Homomorphic Encryption

In traditional encryption schemes, encrypted data cannot directly be operated on without first decrypting it. However, fully homomorphic encryption (FHE) enables direct computation on encrypted data. In practice, this means that clients can outsource computations to external entities that incorporate FHE by passing in their encrypted data, having the external entity perform the requisite computations on this encrypted data, and then decrypting the output, all without leaking any sensitive data to the entity at any point throughout the computation.

A typical issue plaguing FHE schemes is that of error propagation. Many encryption schemes are based upon the *Learning with Errors* (LWE) assumption, and incorporate a small, random error term into the ciphertexts to ensure security. However, as computations (such as additions and multiplications) are applied to ciphertexts, the random errors compound, which often limits the depth of computation workflows that FHE schemes can support.

3.1. FHE Schemes with Arbitrary Operations

In [3], Gentry introduces a seminal procedure known as "bootstrapping," with which he can construct a fully homomorphic encryption scheme capable of supporting arbitrary-depth computations without sacrificing security. In practice, as any computable function can be represented as a Boolean circuit, Gentry's construction implies that any computation should be theoretically performable with fully homomorphic encryption.

In practice, however, the computation and memory overhead associated with homomorphically encrypting operations becomes infeasible for larger and more complex Boolean circuits.

3.2. Concrete ML

The open-source startup Zama AI has constructed practical FHE solutions to many problems with their FHE compiler Concrete ML [1]. Concrete ML operates with TFHE, which stands for fully homomorphic encryption over the (discretized) torus, to speed up computations. Of particular interest to us is that Concrete ML can compile many common machine learning workflows into FHE.

It's perhaps easy to see how linear functions such as matrix multiplications can be compiled into FHE, as they can easily be expressed as a combination of additions and multiplications. However, much more challenging is the analogous task for arbitrary non-linear functions, which are highly prevalent in neural networks, and particularly in LLMs.

The approach employed by Concrete ML to compile an arbitrary function is as follows:

- First, apply *quantization* on the function. This means compressing the input and output space (typically consisting of $32-$ or $64-$bit float values) into a much smaller, discrete space (e.g. $8-$bit values corresponding to $[-128, 127]$).
- Express the resulting *quantized* function as a lookup table (TLU) over the reduced input space.
- Convert the TLU into a Boolean circuit and use the TFHE scheme to fully homomorphically encrypt the computation of the circuit.

Note the inherent tradeoff between having a smaller TLU (and resulting Boolean circuit) or a smaller quantization error incurred. Indeed, as we discretize the function more, the input space is reduced but the resulting quantized function becomes a rougher approximation of the original.

Also note that in Concrete ML's implementation of lookup tables, there is an inherent ± 1 error associated with each operation; the probability of these errors occurring can be controlled with the `p_error` parameter. Low `p_error` may increase accuracy, but they naturally decrease performance.

3.3. FHE LLMs

In [2], Frery describes a system to perform inference on the GPT-2 large language model that uses an implementation of an attention layer encrypted with FHE. Under this system, clients performing inference using GPT-2 are expected to take their input (a list of words or tokens), run it through the layers preceding the attention layer, encrypt the resulting output, pass it through the FHE-encrypted attention layer, decrypt the output, and then finally pass the result through the remaining layers.

This article served as the original inspiration for our project. We attempted to extend the implementation presented by Frery—which did not compile the embedding layer to FHE—and realized that we could leverage the FHE-compiled embedding into the semantic protocols we describe in this paper.

4. Background on Language Model Construction

4.1. n-gram Model

In the n-gram language model, we construct a function f which generates a probability distribution for x_i given the previous n words, $x_{i-1}, x_{i-2}, \ldots, x_{i-n}$, i.e. $\mathbb{P}[x_i | x_{i-1}, x_{i-2}, \ldots x_{i-n}]$.

For the trivial model $n = 0$, the $0-$gram language model simply learns the underlying frequencies of words. However, as n grows, the language model learns to incorporate contextual clues from the previous words, including semantic relationships between words. The typical tradeoff as n grows too large is that there are more possible $n-$grams, which translates to a larger potential vocabulary size and demands a larger training dataset as a result.

One strategy to allow $n-$gram models to learn semantic relationships between neighboring words is via *word embeddings*.

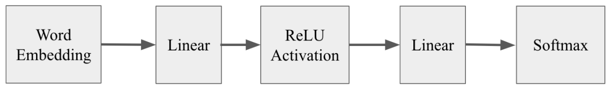
4.2. Word Embeddings

A word embedding represents words as a high-dimensional vector of floating point numbers in a way that conveys semantic meaning. By mapping from a discrete vocabulary of words to a higher-dimensional set of tunable vectors, the $n-$gram model can learn an embedding of the vocabulary that is semantically meaningful. For instance, the difference between the word embeddings of `queen` and `girl` should be similar to the difference between the word embeddings of `king` and `boy`. In general, similar words should correspond to embedding vectors which are approximately parallel (higher dot product), while unrelated words should correspond to embedding vectors that are more or less orthogonal (zero dot product).

Although word embeddings are often touted as being one-way, this is not entirely true: this is why A and B cannot just publicly post their word embeddings to compute their cosine similarity. See, for instance, [5] and [4] for potential algorithms to invert word embeddings.

4.3. Training

We trained an n-gram language model with the architecture given below:



This n-gram language model takes in the embeddings of the n previous words and passes it through the following layers described in the architecture to get a probability distribution over the entire vocabulary for the word that follows.

Below are the key parameters of our model:

- Vocabulary Size = 7472
- Embedding Dimensions = 100
- Context Size = n = 2

Our model was trained to learn the semantic relationships between words by analyzing $n-$grams from a large text. Our dataset was the book *Great Gatsby*, which has $47,094$ words.

Training took 2 hours and 38 minutes, suggesting that with more computational power we could train our model on significantly larger texts and with larger context sizes.

5. Semantic Agreement Protocols

With the requisite background on word embeddings and fully homomorphic encryption schemes, we now move on to discuss our multi-party semantic agreement protocol constructions. In our assumed setup, there are two parties A and B who wish to measure the *cosine similarity* of their plaintext in a way such that the following conditions are met:

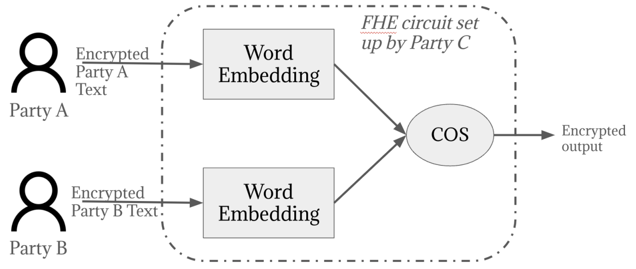
- *Inter-party Security*: Neither A nor B learns anything other than the cosine similarity between their words.

- *External Security*: No external parties learn any information about the words of parties A and B, including their plaintext values, embeddings, or even the cosine similarity between them.

- *Correctness*: At the end of the protocol, parties A and B each receive a value which is the correct cosine similarity value, within some acceptable margin of error.

5.1. A Simple System with a Trusted Third Party

We first present a simple system that allows two parties A and B to determine the cosine similarity of their word embedding through the help of an honest third party. The protocol is as follows:



- A and B share a secret key vector sk with each other via some key-sharing algorithm, e.g. Regev's public key encryption scheme.

- A and B both encrypt their texts using sk and send their encrypted texts to a trusted third party C. Note, A and B must send C their ciphertext messages $\mathsf{ct}_A = \mathsf{Enc}(\mathsf{sk}, \mathsf{pt}_A)$ and $\mathsf{ct}_B = \mathsf{Enc}(\mathsf{sk}, \mathsf{pt}_B)$ through an additional secure layer (e.g. using TLS) to protect against man-in-the-middle attacks from one another.

- A and B also send C an encryption of the bits of their secret key, $\mathsf{ek} = (\mathsf{Enc}(\mathsf{sk}, \mathsf{sk}_1), \ldots, \mathsf{Enc}(\mathsf{sk}, \mathsf{sk}_n))$. This encryption key ek will be used by C for bootstrapping the FHE circuit. C only really needs one of the parties to send ek, so it can arbitrarily decide which one to use and which one to discard.

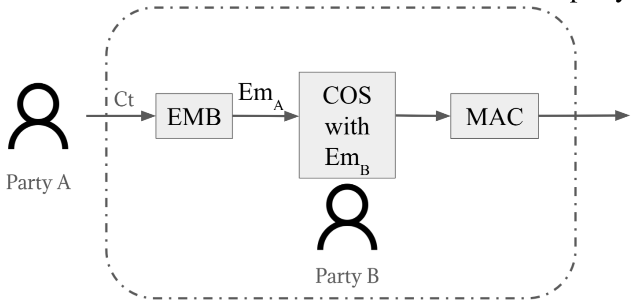
- C uses an FHE circuit to compute the (encrypted) word embeddings of A and B's ciphertexts and then their cosine similarity. C then send A and B the output of this computation, which should be the encrypted cosine similarity $\mathsf{ct}_C = \mathsf{Enc}(\mathsf{sk}, \cos(\theta))$.

- A and B decrypt C's output to determine the cosine similarity $\cos(\theta) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_C)$. Thus, they are both able to recover the cosine similarity of their ciphertexts without revealing the other party's plaintext.

Unfortunately, this system relies on the assumption that C is honest. A dishonest party C could simply, send A's ciphertext ct_A to B who can decrypt it and learn the plaintext.

5.2. A Low Trust Two Party System

We also propose a more complicated system that eliminates the need for a trusted third party.



- A generates a secret key vector sk which is not shared with anyone else.

- A encrypts their text into ciphertext $\mathsf{ct}_A = \mathsf{Enc}(\mathsf{sk}, \mathsf{pt}_A)$ and sends it to B. A also sends an encryption of the bits of their secret key, $\mathsf{ek}_A = (\mathsf{Enc}(\mathsf{sk}, \mathsf{sk}_1), \ldots, \mathsf{Enc}(\mathsf{sk}, \mathsf{sk}_n))$. This encryption key ek will be used by C for bootstrapping the FHE circuit.

- B precomputes the embedding of their plaintext $\mathsf{em}_B = \mathsf{Emb}(\mathsf{pt}_B)$.

- B generates a secret key $\mathsf{sk}_B = (a, b)$ for $a, b \in \mathbb{Z}_p$, which will be used by B to generate a one-time message authentication code (MAC).

- B sets up an FHE circuit that does the following:

- Computes the embedding of A's text $\mathsf{em}_A = \mathsf{Emb}(\mathsf{pt}_A)$.
- Computes the cosine similarity between em_A and em_B. Note that the embedding of B's text is *hard coded* into the circuit, and it is precomputed by B before compiling the circuit.
- Signs the cosine similarity value by appending a MAC. We will do this with a simple one-time MAC scheme where $\mathsf{MAC}(\mathsf{sk}_B, m) = am + b \pmod{p}$. This is done as described in past lecture notes from this class [8].
- Outputs the results, which are the encrypted values $\mathsf{Enc}(\mathsf{sk}, \cos(\theta))$ and $\mathsf{Enc}(\mathsf{sk}, \mathsf{MAC}(\mathsf{sk}_B, \cos(\theta)))$ respectively.

- B send the results of the FHE computation to A, where A will decrypt them and obtain $\cos(\theta)$ and $\mathsf{MAC}(\mathsf{sk}_B, \cos(\theta))$.

- A sends the decrypted result $\cos(\theta)$ and $\mathsf{MAC}(\mathsf{sk}_B, \cos(\theta))$ to B, and B can confirm that this is in fact the cosine similarity it calculated by verifying the MAC.

This system allows for semantic agreement with much less trust involved. However, it comes at the cost of additional complexity in the FHE circuit due to the signing computation.

6. Security Evaluations

6.1. Threat Model

Under a fairly loose set of assumptions, our systems from Section 5 succeed at providing secure multi-party semantic agreement. Namely:

- The third-party computation entity is to be trusted and will not, for instance, simply send party B's encrypted ciphertext to party A (who would then be able to decrypt it).

- Both parties A and B are incentive-aligned in their goal of obtaining a true semantic agreement measure. Therefore, neither A nor B will be dishonest and, for example, send an encryption of intentionally bad text to alter the output. This assumption should follow from Condition 3 as discussed in the Motivation section.

- All parties can be considered *curious*. That is, our system must protect against parties snooping on communications and intentionally trying to determine the secret texts.

Moreover, the hypothetical system proposed in 5.2 eliminates the need for a third party and provides a fairly strong notion of security without the need for the first assumption.

6.2. Analysis of Possible Attacks

We note the existence of a coordinated attack by which a group of parties $A_1, A_2, \cdots, A_n$ may each query their degree of semantic agreement with the same counterparty B, each thereby obtaining a linear equation on the embedding of party B from the received vector dot product. As n becomes larger than the dimension of the word embeddings, it becomes possible for the group $A_1, \cdots, A_n$ to pool their information to deduce B's true word embedding, which is definitely undesirable.

However, this is not truly that problematic for several reasons:

- Counterparty B has control over the rate at which semantic agreement queries are processed, and can simply refuse to answer more than $m << n$ queries.

- We could easily restructure our system to output a binary yes/no with some threshold for similarity instead of the exact cosine similarity, which would make it much more difficult for the aforementioned attack to succeed without sacrificing too much functionality.

- The party evaluating the FHE computation could intentionally add some small randomized error to the cosine similarity value. This would lead to the attacking parties trying to solve a system of equations that looks like

$$A\vec{b} + \vec{e} = \vec{s}$$

where A is a matrix consisting of the normalized embeddings of $A_1, \ldots, A_n$ plaintexts' in each row, $\vec{b}$ is Party B's normalized embedding, $\vec{e}$ are the n randomized errors, and $\vec{s}$ are the returned cosine similarities. Note that, with valid parameters, the LWE assumption should protect B from having their embedding leaked.

- Depending on the word embedding used, it may still be very difficult for the group $A_1, A_2, \cdots, A_n$ to deduce B's exact plaintext. This is especially true if the context length of the embedding is large. Let V be the vocabulary size, C be the context length (i.e. how many tokens are allowed to be embedded), and N the number of dimensions in the embedding. The embedding is then a function:

$$\mathsf{Emb} : V^C \to R^N$$

Given parameters can easily be on the order of $N \approx 1000$ and C can essentially be unbounded (as described in Open AI's paper on text embeddings [6]), it becomes essentially impossible to determine the exact text that would produce any given word embedding.

7. Implementation

We fully implemented a system that is capable of calculating single-word embeddings in FHE. As mentioned in Section 4.3, we trained our own n-gram language model with a context size of 2 and 100 embedding dimensions following the model architecture outlined earlier. The model is implemented in `pytorch` below; we remark that the final softmax layer to extract the final word probabilities is integrated into the training loop instead.

n-gram Model Implementation

```
class QuantNGramLanguageModeler(nn.Module):
    def __init__(self, vocab_size, embedding_dim,
          context_size):
        super(QuantNGramLanguageModeler, self).__init__()
        self.embeddings = qnn.QuantEmbedding(vocab_size,
              embedding_dim)
        self.linear1 = qnn.QuantLinear(context_size *
              embedding_dim, 128, bias=True, bias_quant=None)
        self.linear2 = qnn.QuantLinear(128, vocab_size,
              bias=True, bias_quant=None)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out = torch.relu(self.linear1(embeds))
        out = self.linear2(out)
        return out
```

After training the language model above for several epochs on *The Great Gatsby* for two hours, we extracted the word embedding layer `self.embeddings`. Even though the word embedding layer was already quantized with the Brevitas quantization library, it's not possible yet to directly compile this layer into FHE with Concrete-ML, as it doesn't currently support compilation on lookup tables (which is what an embedding layer is implemented as under `pytorch`).

To transform this into an equivalent quantized layer that can be compiled into FHE, we started by extracting a matrix E of dimensions 100×7472 (embedding dimension by vocabulary size), with the ith column containing the embedding of corresponding word i.

Then, we linearized the computation by expressing the embedding lookup operation as

$$\mathsf{Emb}(\vec{w}) = E\vec{w}$$

where $\vec{w}$ is a one-hot encoding of the word we are embedding. To compile our circuit, we quantized (with Brevitas) the matrix multiplication into `int16` computations as required by Concrete-ML.

Compiling the Embedding into FHE

```
# Load model and other variables.
logger.info("Importing model and other variables...")
vocab_size, embedding_dim, context_size =
      pickle_from_path("model/params.pkl")
model = QuantNGramLanguageModeler(vocab_size, embedding_dim,
      context_size)
model.load_state_dict(torch.load("model/model.pth"))
model.eval()
word_to_ix = pickle_from_path("model/word_to_ix.pkl")
vocab = pickle_from_path("model/vocab.pkl")

# Linearize the embedding.
logger.info("Linearizing the embedding...")
qembedding = qnn.QuantLinear(vocab_size, embedding_dim,
      bias=False, bias_quant=None,
      input_quant=Int8ActPerTensorFloat)
qembedding.weight =
      nn.Parameter(model.embeddings.weight.transpose(0, 1))

# Compile the embedding.
logger.info("Compiling the embedding...")
input_data = torch.stack([word_to_tensor(word, word_to_ix,
      vocab_size) for word in vocab])
input_data = np.array(input_data, dtype=float)
compiled_embedding = compile_brevitas_qat_model(
    qembedding,
    input_data,
)
```

In the above, `word_to_ix` is simply a mapping from the vocabulary space to indices, and `word_to_tensor` is a function that converts a word to the corresponding one-hot encoding. `compiled_embedding` provides the end product–a (slightly transformed) embedding layer taking words to embeddings, compiled with FHE.

Our full implementation can be found in our Github repository [9].

Compilation time of FHE circuit	23.1 s
Avg. time per embedding	0.990 ms
Avg. time per embedding in FHE	2.85 ms
Avg. % error of FHE embeddings	1.45%

Table 1. Performance results

8. Performance

We tested the performance and accuracy of our FHE word embedding implementation. Remember that due to the quantization that is required when using the Concrete-ML library, there will be a slight error when comparing word embeddings computed in FHE and those computed in the clear.

All performance results were determined by running our programs on a full node of the MIT SuperCloud system [7]. The node had 48 Intel Xeon Platinum 8260 CPUs and 192 GB of total RAM. We were able to use these same computing resources for the training of our n-gram language modeler.

We used a sample of 1000 words to determine the average performance and error associated with FHE computations. The percent error of our word embeddings was calculated using the expression

$$\%\text{ error} = \frac{||\vec{e}_{\mathsf{FHE}} - \vec{e}||}{||\vec{e}||} \qquad (\dagger)$$

where $\vec{e}$ is the trained word embedding and $\vec{e}_{\mathsf{FHE}}$ is the FHE computed word embedding.

Our results are shown in Table 1. In particular, the requisite quantization procedures only introduced 1.45% error on average to the word embeddings. Mathematically, this implies a fairly satisfactory upper bound on the percent error of our cosine similarities as well, as implied by the following lemma.

Lemma 1. *Given two word embeddings $\vec{e}, \vec{f}$ and their corresponding FHE-computed embeddings $\vec{e}_{\mathsf{FHE}}, \vec{f}_{\mathsf{FHE}}$, write χ_e, χ_f for the associated errors as defined in $(\dagger)$. Then, the FHE-computed cosine similarity*

$$\vec{e}_{\mathsf{FHE}} \cdot \vec{f}_{\mathsf{FHE}}$$

is within $\chi_e + \chi_f + \chi_e \cdot \chi_f$ percent error of the true cosine similarity

$$\vec{e} \cdot \vec{f}.$$

Proof. Without loss of generality scale such that $\|\vec{e}\| = \left\|\vec{f}\right\| = 1$.

Note that we have

$$\vec{e} \cdot \vec{f} - \vec{e}_{\mathsf{FHE}} \cdot \vec{f}_{\mathsf{FHE}} = \vec{e} \cdot (\vec{f} - \vec{f}_{\mathsf{FHE}}) +$$

$$\vec{f} \cdot (\vec{e} - \vec{e}_{\mathsf{FHE}}) - (\vec{e} - \vec{e}_{\mathsf{FHE}})(\vec{f} - \vec{f}_{\mathsf{FHE}}),$$

and so from Triangle Inequality

$$\left|\vec{e} \cdot \vec{f} - \vec{e}_{\mathsf{FHE}} \cdot \vec{f}_{\mathsf{FHE}}\right| \leq \left|\vec{e} \cdot (\vec{f} - \vec{f}_{\mathsf{FHE}})\right|$$

$$+ \left|\vec{f} \cdot (\vec{e} - \vec{e}_{\mathsf{FHE}})\right| + \left|(\vec{e} - \vec{e}_{\mathsf{FHE}})(\vec{f} - \vec{f}_{\mathsf{FHE}})\right|$$

$$\leq \chi_e + \chi_f + \chi_e \cdot \chi_f.$$

Rescaling produces the desired result.

□

Qualitatively, Lemma 1 implies that if the FHE-computed embeddings are fairly accurate (as they appear to be in Table 1), the FHE-computed cosine similarities should be accurate to a comparable degree. Quantitatively, we expect them to be accurate up to an error of at most $0.0145 + 0.0145 + 0.0145 \cdot 0.0145 = 2.92\%$ on average.

All in all, our FHE implementation suffers from a factor of $\frac{2.85 \text{ ms}}{0.990 \text{ ms}} \approx 3$ multiplicative overhead from running on slower FHE operations and a $\approx 3\%$ approximation error incurred from quantization on an average cosine similarity computation–not a particularly large price to pay for the security of semantic agreement.

9. Our Attempt at a FHE LLM

At the start of this project, our goals were more ambitious: we hoped to fully quantize all the layers of the GPT-2 model and perform all computations in the entire model in FHE. However, we ran into multiple issues. Unlike our existing system, running inference with the GPT-2 model requires quantization of non-linear computations such as the softmax layer. This is currently not implemented in the Concrete-ML library.

We briefly tried to implement a softmax layer manually by applying an exponential, summing over the final axis, and dividing the resulting matrices. Although each of these operations is supported by Concrete-ML individually, they were only supported up to tensors of dimension at most 3 while the `minGPT` model we studied operated on tensors of dimension 4. The authors believe that it should nonetheless be possible to quantize and compile a large language model such as `minGPT` into FHE.

10. Conclusion and Future Work

Our papers present a proof of concept for safe multi-party semantic agreement schemes. We presented novel protocols that leverage FHE and the word embedding techniques of NLP to successfully allow multiple parties to come to a semantic agreement without exposing confidential text.

Moreover, we trained an n-gram language model to create our own word embedder. Using the existing FHE library Concrete-ML, we were able to successfully implement a system that computes single-word embeddings on encrypted words.

Our current system is limited by the constraint that parties may only measure semantic agreement on *single words*. This is simply a limitation of the language model we decided to implement. Many text embedding models on the market can work on arbitrarily large texts. Nevertheless, our semantic agreement protocol stands as a compelling proof-of-concept—it extends easily to operate on embeddings of text with arbitrary length—showing the potential for implementation of more powerful agreement schemes using FHE.

We hope in the future to be able to fully implement our proposed scheme in software. While we were successful in evaluating FHE embeddings, our current system does not perform the extra step of determining cosine similarities.

11. Contributions

Miguel was responsible for the original idea of FHE LLMs which later became this project. He also implemented the FHE word embeddings in software and evaluated their performance. Finally, he developed the two-party semantic agreement protocol in Section 5.2.

David analyzed various codebases to assess their viability for integration with FHE, found the initial baseline codebases for the existing language models that could be converted to FHE, and implemented a majority of the (ultimately unsuccessful) quantization of the softmax layer.

Maria helped motivate the system discussed in 5.2 and helped find security issues in the systems the authors discussed (e.g. the need for a MAC in 5.2 & vulnerabilities when repeating the scheme many times). She also analyzed the zama.ai codebase to ascertain how linear & nonlinear operations were implemented.

Edward assisted in efforts to quantize `minGPT`, helping the authors ultimately succeed at quantizing the entire multi-head attention module apart from the softmax layer. He was also involved in discussions about security evaluations of the three-party semantic agreement protocol in section 5.1 which ultimately led to the theoretical two-party agreement system.

References

[1] Ayoub Benaissa. Concrete — zama's fully homomorphic encryption compiler. https://www.zama.ai/post/zama-concrete-fully-homomorphic-encryption-compiler, 2023. 2

[2] Jordan Frery. Towards encrypted large language models with fhe. https://huggingface.co/blog/encrypted-llm, 2023. 2

[3] Craig Gentry. Fully homomorphic encryption using ideal lattices, 2009. 2

[4] Kai Kugler, Simon Münker, Johannes Höhmann, and Achim Rettinger. Invbert: Reconstructing text from contextualized word embeddings by inverting the bert pipeline. *arxiv*, 2022. 3

[5] Haoran Li, Mingshi Xu, and Yangqiu Song. Sentence embedding leaks more information than you expect: Generative embedding inversion attack to recover the whole sentence. *ACL Anthology*, 2023. 3

[6] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. Text and code embeddings by contrastive pre-training. *CoRR*, abs/2201.10005, 2022. 6

[7] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew

Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018. 7

[8] Ron Rivest. 6.857 lecture 3: Unconditionally secure authentication. https://web.mit.edu/6.857/OldStuff/Fall97/lectures/lecture3.pdf, 1997. 5

[9] Miguel Tulla Lizardi, Edward Wan, Maria Chrysafis, and David Hu. 6.5610 Final Project. https://github.com/mtulla/65610_project. 6