Explore the Possibility of Fine-grained Non-encrypted Distributed Model Service: an Adversary View

*Course Project Type: Research

Bohan Cui
Department of Computer Science and Technology
Nanjing University
Nanjing, China
bohan.cui@outlook.com

***Abstract*—The increasing computational demands of large-scale machine learning models have made their training and deployment both expensive and resource-intensive. Traditional approaches to protect the intellectual property (IP) of these models, such as encryption, introduce significant overhead. This paper explores the possibility of deploying models in a fine-grained, non-encrypted distributed manner to untrusted devices. By partitioning model parameters into small blocks and distributing them across numerous devices, we aim to make the reconstruction of the complete model by adversaries computationally infeasible and financially unattractive. We propose a threat model to assess this approach and conduct experiments to evaluate its effectiveness. Our findings indicate that this fine-grained distribution can effectively protect model IP, provided the system can tolerate a certain level of additional overhead.**

***Index Terms*—IP protection, Privacy, Distributed system, MLaaS.**

I. Introduction

Nowadays, the consumption of the machine learning models becomes much larger and larger, which make the inference and training of these models expensive, time-consuming and power-consuming. [1] The training of GPT4 model consumed electricity equal to the consumption of medium-sized European country.

Calculation	Value (GPT-3)	Value (GPT-4)
Energy per Query	0.0003 kWh	0.0005 kWh
Total Queries per Day	10,000,000	10,000,000
Total Daily Energy Consumption	**3,000 kWh**	**5,000 kWh**

Fig. 1. The power consumption of the inference of GPT3 and GPT4 [3]

Under this circumstance, some cloud computing enterprises sell computing power to the AI companies, research institutes or governments to train a model or deploy the model service, which is named Machine Learning as a Service (MLaaS). In this way, the Machine learning model can be regard as a commodity with high business value. Therefore, the Intellectual Property (IP) of the models is expected to be protected well when the models are deployed on the third-party computing power suppliers, especially on untrusted entities.

In MLaaS aspect, to offer a more power-efficient and cheap solution for the MLaaS, we proposed a system *AwfulOwl*[1] to utilize the idle computing resources on the edge devices and mobile phone, in which we distribute the model to a certain amount of untrusted users' devices for execution.

In this article, we focus on the IP privacy concern of *AwfulOwl* or any other systems like *AwfulOwl*. Traditionally, when model are deployed on cloud computing server, the IP is usually protected by encryption [6]. [2]

The encryption technique will introduce quite a large overhead to the training and inference, which will improve the cost of some enterprise or hinder the model from being deployed on some scene. [2] Therefore, we explore the possibility to deploy the model to the devices without encrypted, by chopping the parameters into small blocks. With these fine-grained distribution, the pirate needs to take much longer time to reconstruct the model than training the model itself, thus the attack behaviour will become meaningless in financial aspect.

This project propose a threat model for this system and we combines the experiment, calculation and review to discuss the probability of this protection practice. And we find it can effectively protect the the IP if the system can afford a certain threshold of extra overhead.

All the codes of this project is published on Github[3].

[1]This project has not be published up to now. Once it is available, you can access to it at https://github.com/AwfulOwl-Community, if we decide to make it open-source.

[2]Techniques like TEE are not realistic on client-side devices.

[3]The repository is https://github.com/Jackcuii/CyberSecProj2024

II. Challenges and Modeling

Our problem is based on 4 assumptions of the system as below:

1) The model is not encrypted, so the controller of the device holding the parameters can access the data directly.
2) The pirate can only control part of the parameters, for the parameters are distributed to a large amount of devices.
3) All the devices are untrusted, which means **any** of the devices are possible to be hijacked by the pirate.
4) There is a dataset can valuate the model and be accessed by everyone.

And we define protecting the IP of the model as avoiding the attacker getting a duplicate of the whole model with similar performance. And the effort of the pirate should be lower than the original training or retraining the model.(We define effort as GPU hour here.)

We define S_i and P_i as the strategies or the facts of the service provider and pirate. i means the level of the strategies/facts. We assume the attacker and the service provider upgrade the techniques according to each other's. Therefore, S_k targets P_{k-1} and P_k targets S_{k-1}. Specially, S_0 refers to the system with no protection. Then we will give detailed definitions for the strategies/facts we are going to discuss.

Original system (S_0) We define our original system as multiple untrusted devices, and the whole architecture of the model is mapped to the devices in this system in a static pattern. All the devices can be managed to communicate with others. The input of the model is flowed to some of the devices, and the result of the model can be gotten from some of the devices. When training the model, the input can also be the grad and the directions can be reversed. Here in Fig we display an system adopting pipeline parallel and tensor parallel as an example. The parameters belonging to the layer 1(blue) are stored in some of the devices and those belonging to layer 2(yellow) are stored in the others.

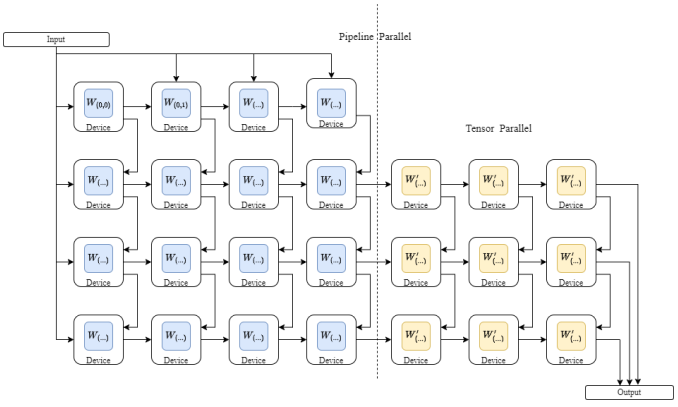

Fig. 2. Original System

Naive attack (P_1) In the following figures, the parts under the shadow are regarded as invisible or inaccessible to the pirates. And the devices and information acquired by the pirates are marked with red. We assume that the attacker can control part of all the devices and they know the exact position of the parameters hosted in the devices in the whole model (i.e. which layer they belong to and where they are in the parameter matrix of the layer). We will evaluate the performance of such a 'stolen' part in the experiment hereinafter.

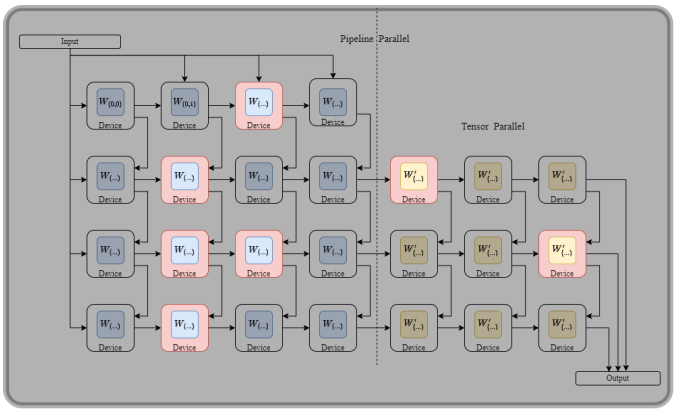

Fig. 3. Naive attack

Complexity of the operators (S_1) In the real scene, the attackers actually can hardly get the true position of the parameter blocks they control. Then we can have an approximation on it. We define G as the number of the blocks. Then the possibility of the having the blocks is $\frac{1}{A_N^N}$, which means the space of the solution is $O(N!)$. But what if they do not have the parameter blocks in the right place? We will introduce the experiment result in the next part.

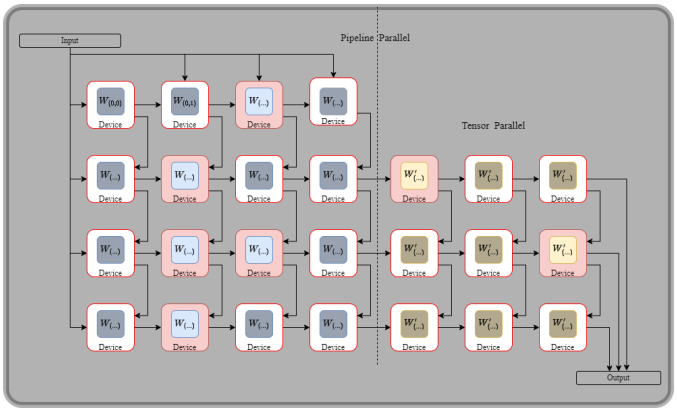

Fig. 4. Complexity of the operators

Solve the real position (P_1) Then we assume the attacker knows which layer the blocks belong to somehow, then they try to solve the real position of such blocks. Instead of Brute Force, the attackers actually can have more smart attacking methods, to search in the solution space, like heuristic algorithms or machine learning methods.

Complexity of the architecture (S_2) Then we take the assumption in P_1 into account. For one thing, some commercial

companies will not publish the detailed architecture of their model. For another, the parameters are usually just floating point tensors, with no clues or pattern to show the layer they belong to. Therefore, the attacker can not directly do the searching on a subset of parameter blocks they acquire, which increases the difficulty of duplicate the model.

Stream hijacking ($\mathbf{P_2}$) To gain some knowledge of the architecture of the model, the attacker can do some analysis on the covert channels, to be more specific, the stream of the data. Due to the fact that the whole system is connected by the Internet, the pirates are likely to eavesdrop the streams between the devices. We will propose a method to analyze the streams.

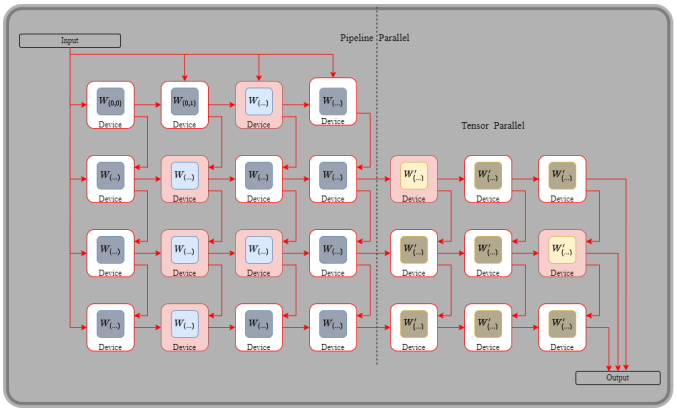

Fig. 5. Stream hijacking

Stream Obfuscation ($\mathbf{S_3}$) To hide the pattern of the communication between the devices, the system can add some extra fake-stream to the normal streams. So it can mix the pattern to hide the layers from analysis.

III. Design and Results of P_1

We choose the VGG16 CNN as the object of our experiment. We adapt the output to 100 to fit the CIFAR-100 dataset. We use pre-train the VGG-16 100 epoches on the CIFAR-100 from random initial values. Then the model achieves 61& accuracy on the CIFAR-100 test set. We use this snapshot as the baseline of the experiment.

The device we used was NVIDIA RTX-4060 GPU, and we use about 2 hours to pre-train the model. Therefore, the *cost* of the model is 2 GPUhours on RTX-4060.

We focus on the largest layer of the MLP, which is a 25088×4096 matrix. The granularity of our experiment is 256×256. So the layer of the model will be distributed to 16×98 devices. So we will test the situation that the pirates control different proportions of the devices.

So we only reserve part of model blocks, and set other parts to zero and run the testset on it to get the accuracy of the model. Actually, the process is equivalent to a coarse-grained *dropConnect*. We ranged the drop rate from 50% to 100% and ran 3 times for each rate to get the average accuracy. Then we plot the figure as below (Fig.6).

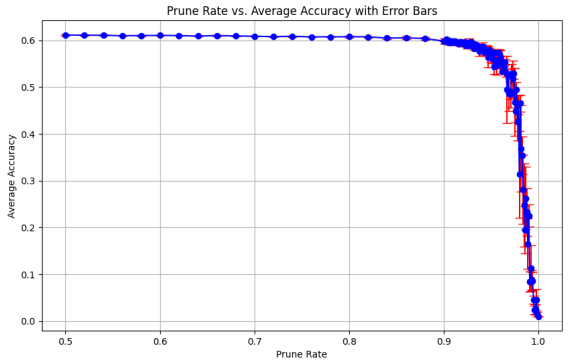

Fig. 6. Experiment Result

We can observe an obvious decrease of the accuracy when the drop rate is larger than 90% and a sharp drop when it is larger than 95%. Therefore, if the pirate get control of 10% of the devices, given they know the exact position of the parameter blocks.

IV. Design and Results of S_1

Then we will test whether the performance decrease is acceptable if the parameter blocks are not at the right position. We simulate this situation by randomly exchange some of the given blocks. So we have two variables in this part, the drop rate and the exchange pair sum. The drop rate ranges from 85% to 100% and exchange pair sum ranges from 0 to 350. The result is displayed below.

The 3D plot is shown on the left, and to more intuitionally display the result, we adopt a contour plot to show the relationship between position-correctness and accuracy(Fig.7).

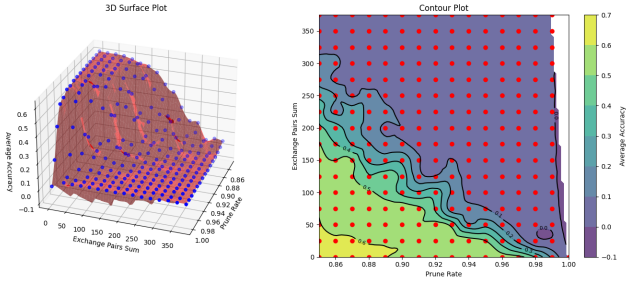

Fig. 7. Experiment Result

The contour plot shows that when the drop is from 85% to 90%, the left model is only tolerant to only 25 pairs' exchanges of the blocks. Otherwise, the accuracy will drop significantly (by at least 10%). Therefore, the performance is not acceptable to the attacker if they can not figure out most of the right positions of the parameters.

V. Design and Results of P_1

The formal description of the searching of the right positions is as below.

Given a function Val, a set of values S, among all the orders of S, find $max(Val(order(S)))$

This is a combination optimization problem. We can adopt Reinforcement Learning and Heuristic Searching.

Genetic Algorithm: Any position combination of the blocks is an object in the population. And the fitness function is the accuracy of the model. Objects evolve by exchange the position of the blocks.

Reinforcement Learning: The agent is the pirate. The environment is the model evaluation. The action is the exchange of the position of the blocks. The reward is the accuracy of the model.

The population of the genetic algorithm is tested with the range from 10 to 100 and iterated 10000 generations to get the best combinations. The results show that all the groups only have 1% (equals to randomly yield output from any input).

Then we will give explanation to the result of the experiment. To search an *informative* solution, we must have some clue/pattern from the beginning and mix them to generate *more informative* solutions.

- The space of the solution in the example is very large. Go(AlphaGo): 361!, This example: 1568! So a very powerful structure is needed.
- The probability of having some right clues in the initial state of the genetic algorithm.
  $P(\text{ clue }) = \frac{1}{16\times 98} \times \left(\frac{15}{16}\right)^{(156-1)}$
  $E(1 \text{ clue in all population }) = 0.000452$
  So it naturally has only 1% (equals to randomly) accuracy

Thus, the attack will fail if the system divides the model *finely enough*. When it comes to a certain system, there exists an acceptable upper bound of the division.

From the experiments above, there is an threshold of the granularity of the system. And we will calculate this value below for this scene.

We assume the size of the training set is $Size(Tr)$, test set is $Size(Ev)$, Epoch is Ep and the time of doing one propagation is T.

Therefore, the training time is

$$T_{tr} \approx 2T \cdot Size(Tr) \cdot Ep$$

The attacking algorithm's generation is gen, the block sum is N, the population is Pop

Then

$$P(clue) = \frac{1}{N} \cdot (1 - \frac{1}{\sqrt{N}})^{N-1}$$

$$E(1 \text{ clue in all population}) = Pop \cdot P(clue)$$

The attacking time is

$$T_{ak} \approx gen \cdot Pop \cdot Size(Ev) \cdot T$$

The most optimistic approximation is

$$E(1 \text{ clue in all population}) \geq 1$$

There is another important factors of T_{ak}, the precision of the accuracy when test the combination. If the precision is too low, the subtle increase of the correctness of the combination will not cause one more prediction to become right, then it will not lead to the variation of the accuracy. Then the algorithm will not converge. (A figure to show this situation Fig.8) So we should take the precision Pr into account.

$$Pr = \frac{1}{Size(Ev)}$$

The condition of successful attack is $T_{ak} \leq T_{tr}$ and $E(1 \text{ clue in all population}) \geq 1$

Then

$$gen \cdot Pop \cdot \frac{1}{Pr} \leq 2 \cdot Size(Tr) \cdot Ep$$

$$Pop \cdot \frac{1}{N} \cdot (1 - \frac{1}{\sqrt{N}})^{N-1} \geq 1$$

Set the constants according to the real situation and solve the binary inequality. Get the result minimum of the N and check if it is possible to distribute the model into N part. Then the architect can very roughly determine if this practice is useful on a certain system.

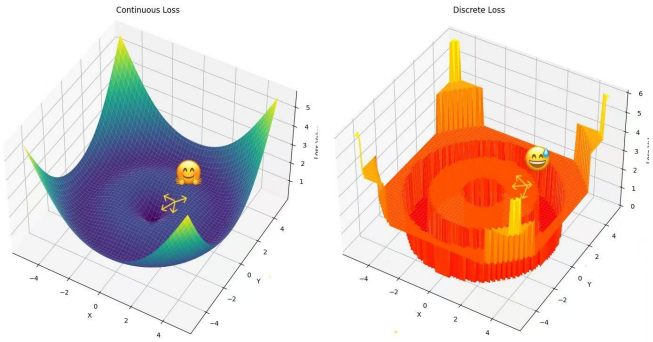

Fig. 8. Continuous and Discrete

VI. Methods for P_2 and S_3

The stream information can be classified into sequential or flatten. Examples are given to elaborate the two kinds respectively. We proposed an algorithm to analyze the streams to divide the nodes int layers/operators for each of these two types.

Sequential Information We can assume a *MapReduce* [5] like system as below (Fig.9).

And we assume the attacker can get the sequential information of the stream, which means he knows the timestamp of all the communications. In this system, 3 parts from the 3 layers respectively are hosted in each device. Therefore, given the

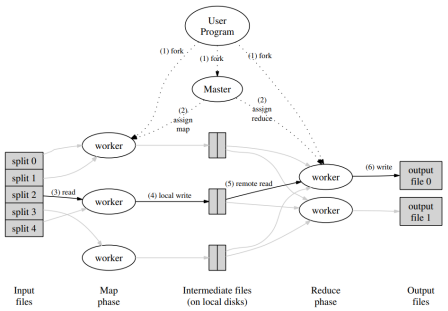

Fig. 9. Diagram in *MapReduce* Article [5]

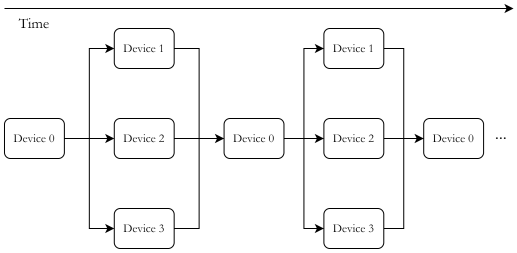

Fig. 10. Sequential Information

attacker knows the sequence of communication, he can figure out the corresponding part for each layer, as below(Fig.10).

According to the pattern of the information, we can find that it is easy to find the *joint point* in the topology. So we find the *k-cut vertex* for the graph, then we use these cut vertices to divide the graph (the entry/exit are also regarded as pivots). Then each part of the system has large probability to be in the same layer/operator. The pseudo codes are shown in **Algorithm 1**.

Algorithm 1 Divide by Sequential Information

Require: $G = (V, E)$ is a graph with designated entry and exit vertices
Ensure: A sequence of vertex layers

function DIVIDESEQ(G)
  $k \leftarrow$ SOLVECONNECTIVITY(G)
  $vertices \leftarrow \emptyset$
  $cutVertices \leftarrow$ FINDKCUT(G, k)
  $vertices \leftarrow \{entry\} \cup cutVertices \cup \{exit\}$ ▷ Include entry and exit
  $layers \leftarrow \emptyset$
  for all consecutive vertices $v_i, v_{i+1} \in vertices$ **do**
    $layer \leftarrow \{v \in V \mid v$ lies between v_i and $v_{i+1}\}$
    $layers \leftarrow layers \cup \{layer\}$
  end for
  return $layers$
end function

Flatten Information The information gotten by the attacker can also be flatten, which means the streams do not have timestamps so that the attacker can not know the order of the streams. So the 'bottleneck' is not obvious for these information. We assume there exists a system exploiting the pipeline(Fig.11).

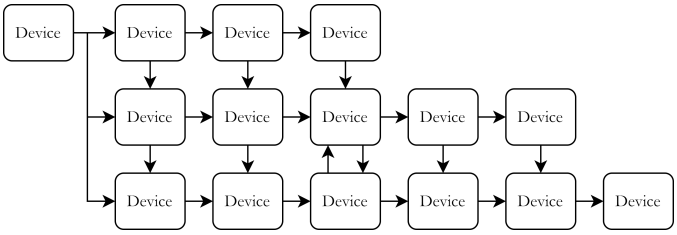

Fig. 11. System

A better solution is to use the transitive closure. We can observe the *proportion of the parallelism*. We can solve transitive closure for the graph and get the sum of the *reachable nodes* for each node. In dataflow driven systems, along the flow of the data, the *reachable nodes* decreases monotonically. Therefore parallelism can be regarded as the same *reachable nodes*. We can find the *sum* which the least nodes have, so the nodes with with these *sum* can be regarded as *not so parallel* and likely to be the pivots of the layer/operator. **Algorithm 2** shows the algorithm described above.

The result of the example is shown in the figure below. The threshold is set to 2. The *reachable sum* is marked on the blocks. And the pivots are marked with star. The figure 12 shows that the algorithm works well on the given diagram.

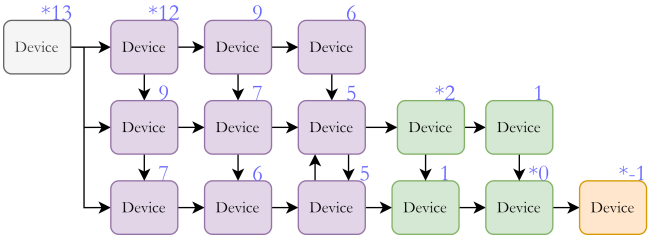

Fig. 12. Result of the Algorithm

Obfuscation To obfuscate the streams between the devices, we can use a Machine Learning method.[4].

We assume the model is M, the input $G < E, V >$ is the original info graph. The $G' < E', V >$ is the output of M. The labels are $S = \{S_1, S_2, S_3 ... S_k\}$. The predictions are $P = \{P_1, P_2, P_3 ... P_m\}$. The function Div is the function to divide the graph into subsets, which is mentioned above.

Therefore

$$G' = M(G)$$

And

$$P = Div(G') = Div(M(G))$$

the loss can be

[4]Because I really lack device to find the hyper-parameter for the model, so I only offer a theoretical formula here

$$Loss(M) = \frac{\Sigma_{i,j \in G, i \neq j, i \in P_u, S_x, j \in P_v, S_y} \ credit(i,j)}{Vk} + \Delta G$$

$$credit(i,j) = \begin{cases} b, & \text{if } (x \oplus y) \oplus (u \oplus v) \\ a, & \text{otherwise} \end{cases}$$

$$\Delta G = c \cdot \mathbf{Diff}(G, G')$$

a and b are hyper-parameters indicates the reward of correct prediction and penalty of the wrong prediction.

ΔG is like a regularization term and **Diff** can be any function to evaluate the increasing of the streams. For example, it can just be

$$\mathbf{Diff}(G, G') = \Sigma_{e \in E' \oplus E} \ 1$$

And c is a hyper-parameter for the penalty for unit increasing of stream. This regularization term serves as the penalty of the extra communication overhead the obfuscation poses on the system.

Then we can train the M with the given $Loss$ function to train a transformation model to generate obfuscated plan.

Algorithm 2 Divide by Flatten Information

Require: $G = (V, E)$ is a directed graph
th is a threshold value
Ensure: A sequence of vertex groups divided by pivots

function DIVIDEFLATTEN(G, th)
  $closure \leftarrow$ TRANSITIVECLOSURE(G)
  for all $v \in V$ **do**
    $reachable[v] \leftarrow |\{u \in V \mid (v,u) \in closure\}|$ ▷ The reachable of nodes are monotonic along the 'dataflow'
  end for
  $sorted\_nodes \leftarrow$ SORT(V, by $reachable[v]$ in descending order)
  $freq \leftarrow$ COUNTFREQUENCY($reachable$)
  for all $n \in$ UNIQUEVALUES($reachable$) **do**
    if $freq[n] < th$ **then**
      $pivots \leftarrow pivots \cup \{v \in V \mid reachable[v] = n\}$
    end if
  end for
  ▷ Entry/Exit node is always a pivot(Unique Max/Min of reachable)
  $groups \leftarrow \emptyset$
  for all consecutive pivots $p_i, p_{i+1} \in pivots$ **do**
    $group \leftarrow \{v \in V \mid v$ lies between p_i and $p_{i+1}\}$
    if $group \neq \emptyset$ **then**
      $groups \leftarrow groups \cup \{group\}$
    end if
  end for
  return $groups$
end function

VII. Related Work

AwfulOwl-like system being an obscure architecture, its IP protection is never mentioned. So we list some researches on normal IP protection and the system itself[5]: [7] [8] [9] [10]

VIII. Conclusion

In our research, we offer basic proofs to prove that it is possible to protect the intellectual property of the model by divide the parameters into small parts, as long as the provider can bear the overhead of the fine-grained distribution.

And we propose a method to attack/protect the structure of the model by analyze/obfuscate the stream of the model, which can threat/strength the IP privacy of the model.

IX. Discussion

IP protection of ML models is much more urgent than the normal software commodity. The competitiveness of regular software industry is always market share and user retention. So sometimes small-scale spread of pirate version is not so severe. But the barrier-to-entry of AI company is the large cost of training model itself, so leak of model will make the market into a chaos.

Then I tend to have a negative prediction on the value of this problem in the long run. The cost of training a model is always not affordable for small companies. But with the progress of the scaling, the model will finally be too large and too expensive to be trained by any single company [4]. At that time, the state-of-art model can only be trained by a union of a large bunch of AI companies and be open-source. (If not open-source, they will be Cartel which is not allowed by any stable government.) If the model becomes open-source, IP protection issue is not so important.

X. Limitations

Note: This part seems not supposed to appear in a normal academic article. But this is a course report and exactly some of the experiments can not be conducted due to the limitation of the hardware resource and time, So I believe it is necessary to list them here.

- Different from most of researches in C&P, which find/target a *valid* attacking method, this project is related to an attacking method proved *unsuccessful*. However, from the commercial aspect, proving to be safe is a key factor to attract investment and secure a commercial success.
- We assume the cost of training increase linearly relatively to scale. This is not the most general case.
- Due to the tight budget, the model may not reach the extreme performance of a certain structure (and the VGG structure is not state-of-art) in the experiment, which is different from the common case. So the approximation of accuracy loss can be a bit optimistic.
- The conclusion is mostly based on CNN. Proofs on other classic architecture are limited.

[5]More generally, it is called Crowd Computing

- Some of the experiments are not conducted due to limited GPU devices.(e.g. P_2: ML method can be a better method, S_3: not really train a model to evaluate the method.)
- For P_2 and S_3, if we both adapt ML method, a GAN may be a better solution.
- An interesting idea was encountered when conducting this project that if the attacker get full knowledge of part of the model , to what extent can the attacker accelerate the duplicating of the model with existing methods. (Simulate the scene that part of the database of an AI company leaks.)

ACKNOWLEDGMENT

Thanks for Prof.Wu, Prof.Xu and TA Wendi's (NJU, National Key Lab for Novel Software Technology) kind help and instructions in this semester.

Thank my friend Rong Zhang (THU, Department of Automation) for borrowing me his RTX-4060 to conduct the experiments.

Some codes of the experiment are written with the assistance of GPT.

REFERENCES

[1] J. Getzner, B. Charpentier, and S. Günnemann, "Accuracy is not the only Metric that matters: Estimating the Energy Consumption of Deep Learning Models," arXiv preprint, arXiv:2304.00897, 2023. [Online]. Available: https://arxiv.org/abs/2304.00897

[2] D. Peringanji, "Unlocking the Future: Privacy-Preserving ML Experimentation," International Journal for Research in Applied Science and Engineering Technology, vol. 12, pp. 350-356, 2024. doi: 10.22214/ijraset.2024.60969.

[3] M. Rebuzzi, "Energy Consumption of ChatGPT Responses," Baeldung on Computer Science, Jul. 8, 2024. [Online]. Available: https://www.baeldung.com/cs/chatgpt-large-language-models-power-consumption. [Accessed: Jan. 18, 2025].

[4] I. Sutskever, "Sequence to sequence learning with neural networks," NeurIPS 2024 Test of Time Award, YouTube, Dec. 2024. [Online].

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008.

[6] M. Xue, Y. Zhang, J. Wang, and W. Liu, "Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations," IEEE Transactions on Artificial Intelligence, vol. 3, no. 6, pp. 908–923, 2022.

[7] Pramanik, P.K.D., Pal, S. & Choudhury, P. Mobile crowd computing: potential, architecture, requirements, challenges, and applications. J Supercomput 80, 2223–2318 (2024). https://doi.org/10.1007/s11227-023-05545-0

[8] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[9] L. Gomez, A. Ibarrondo, J. Márquez, and P. Duverger, "Intellectual property protection for distributed neural networks," 2018.

[10] J. Zhang, Z. Gu, J. Jang, H. Wu, M. Ph. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS '18*, New York, NY, USA, 2018, pp. 159–172. Association for Computing Machinery.