IRONCORE LABS

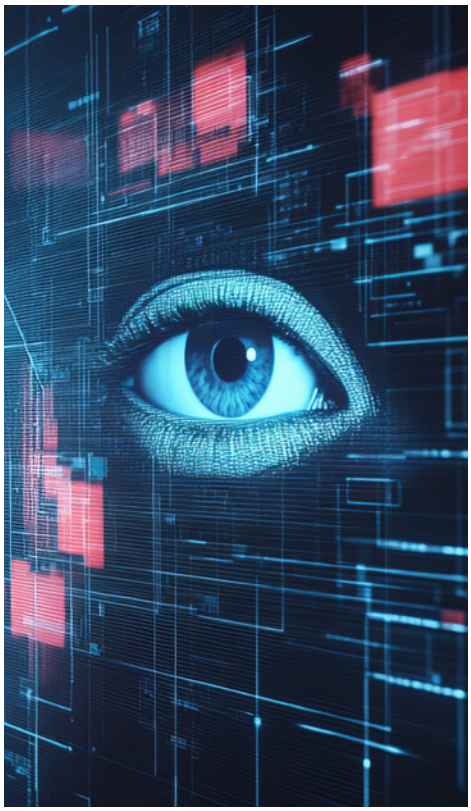
Training AI Without Leaking Data

How Encrypted Embeddings Protect Privacy

UNSUPERVISED AI

Generative-AI is mind blowing. Not only can it be used to create realistic text and images in response to open ended prompts, but the models can be trained without supervision. Historically, perfecting training sets by adding good data and weeding out garbage data was the main way to improve models. Now, given sufficient data, these steps can often be omitted. **More data means better results, but more data also means companies building models are hungry to use as much of YOUR data as they can get their hands on.**

THE RISKS OF PRIVATE DATA IN AI



We want our AI systems to be good, but we don't want our private data to leak out to others. It's no one else's business what I wrote in my diary or who I met with last week or what my doctor told me my prognosis is. Companies don't want internal discussions of predicted business results leaking outside of their executive group.

This creates a tension: what if I want an AI model to raise red flags when we are opening ourselves up to liability? Or what if I want a model to summarize my previous week or quarter or to suggest when I should meet with someone again based on prior patterns or meeting transcripts which may be private?

Undoubtedly, AI could be more useful to individuals and businesses with access to more data. Software providers can deliver more value if they can leverage AI using customer data either for per-customer models or aggregate models to everyone's benefit.

At the same time, anything an AI is trained on has a risk of showing up in outputs.

Even though all the training data is mixed together inside the neural network to produce outputs that theoretically generalize from the inputs, specific training inputs are often produced back verbatim, which is why the New York Times was able to prove their pay-walled articles were used to train ChatGPT, by careful prompting that ultimately extracted entire articles sentence for sentence.

If we let OpenAI train on our private data, there's a high risk that anyone using the model could see our information.

If your data is used to train a generative model, or even to fine-tune it (additional training on top of an existing model) then your data is at risk of extraction.

The risk of data leakage from a model is lessened when the model is not generative, meaning a model that isn't producing free-form text or images. For example, a model trained on detecting sentiment in text or categorizing subject matter (e.g., using a finite list of possible topics like finance and health) is not going to produce an output that is problematic.

But even non-generative models trained on private data introduce risk around the storage and handling of the training data.

WHAT IS INNOCUOUS?

Suppose you want to build your own AI model and train it to predict what's next in a sequence such as what video a person might watch next given their video viewing history. Before you can build the model, you need access to the viewing history of a large number of people, which means using information that some people might feel hesitant to share.

This is a fairly innocuous seeming scenario. We can assume the data doesn't include personally identifiable information or bank account info or medical records. At the same time, the information that Netflix and YouTube gather on people's viewing histories is likely to tell them a lot about political leanings, sexual orientation, medical diagnoses (people searching on stage 4 melanoma are probably diagnosed with it or know someone who is), and so on.

So is that information a privacy problem? Even if your exact watch history gets recommended to someone else, it isn't saying, "So-and-so liked similar things to you, so try this." It's aggregated and stripped down and anonymized. In this situation, is the data a problem? Not until more data is mixed in.

Imagine the AI engineers want to improve the models results by including location and gender and age. After all,

more data produces better results in AI, and it stands to reason that there are regional and generational preferences at play. Yet adding more information makes individuals more identifiable.

What if your recommendations are heavily weighted to what your neighbors are watching -- does that invade their privacy?

Perhaps more importantly, who is building the models? What can these companies and the engineers they employee see? How much of an issue is it that they can comb through the raw data points? It's a slippery slope. Private data should be kept private to the individual who produced it as a default stance, always, because even innocuous-seeming data can be very invasive.

A MORE PROBLEMATIC EXAMPLE

Let's switch to an example that carries more obvious risk: suppose we're building a model based on people's interactions with customer support. We take the thread of interactions between a customer and a support agent and associate it to the rating that the customer later gives their experience. The idea is to be able to look at future interactions across all customers and predict those ratings so that supervisors can be alerted of interactions that are problematic before they blow up. For now, we'll assume that the model being built is specific to a particular company, though a customer support SaaS company could easily choose to train across all of their customers and offer a "red flag" feature to everyone, which compounds the risks.

The issue with this data is that customers share lots of personal information. They send passwords, give details about their private lives, send credit card numbers, and sometimes they mention what's going on in their life and how little they want to be talking to customer support at the moment when they're going through chemotherapy or whatever. User generated content that isn't intended to be public is a privacy nightmare.

It would be an invasion to let data scientists browse through these conversations and we don't want to propagate potentially sensitive data to more places where it could be seen or stolen. As an added risk, this data, once available to the machine learning engineers, could get tapped for other purposes, such as for fine tuning generative models that recommend responses to customers.

In short, privacy is endangered when data is used to train AI models.

Instead of weighing those risks versus the benefits of the models that could be created, **let's look at ways to make the models while locking up the data** to minimize the number of people who can view it and to reduce the risk of theft.

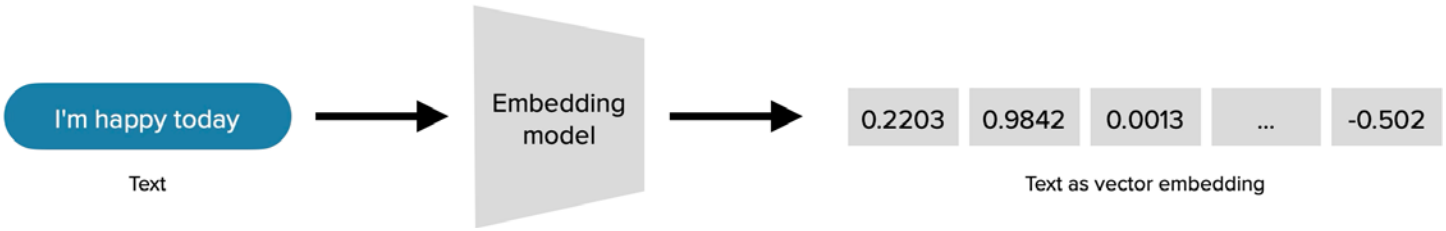
VECTOR EMBEDDINGS AS TRAINING DATA

Any input in AI has to first be reduced into a mathematical representation. The input is often a fairly direct representation. For example, an image might be reduced to a grid (matrix) of color values for each pixel. A sentence may be reduced to a list of numbers where each number represents an index to a dictionary entry.

But there's another option that's more rich and meaningful, and that's a vector embedding.

Vector embeddings are produced by embedding models, which are trained to extract the concepts, meaning, or intent of inputs and encode that information into a vector (a long list of small numbers). We can measure the similarity between two inputs by finding the distance between the two vectors.

If we generate embeddings from images, two that are similar might both be photos of an empty beach on a sunny day. A very different (as measured by a distance calculation) image might be a portrait of a person with a birthday cake at a restaurant.



Vectors also capture intent and meaning in sentences. The sentences "I'm happy today" and "this morning I feel great" are similar despite using different words and sentence structures and they'd have a very small distance between them. The sentence "I need to fix the sprinklers" would be mathematically distant from the other two.

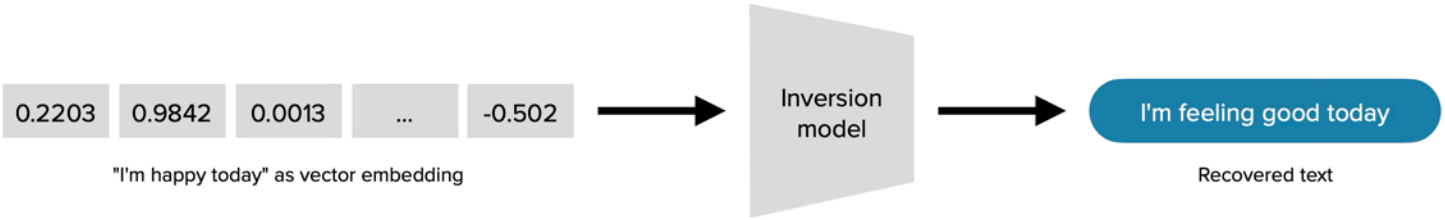
Embedding vectors are often used in natural language search to find text with similar meaning or images that could be similarly described. They have other uses, though, too, such as for recommendation engines where the similarity measures can be used to find similar products or products that others with a similar purchasing history have also bought, for example.

You can build an embedding model for just about anything, but there are a lot of pre-trained models for handling text, image, video, audio, and other inputs. And you can use the vectors these models produce as training inputs for making new models, which is a technique that can lead to faster inferences, smaller more efficient models, and shorter training times.

⚠️SECURITY OF EMBEDDINGS

Although many people think converting inputs into vector embeddings is enough to protect the data, that's not the case. There are thousands of academic papers on ways to "invert" embeddings back to near approximations of their original forms – images, faces, text, or whatever. While a human looking at a series of numbers can't directly make much sense out of it, there are techniques to make those numbers very meaningful.

We've covered some of these attacks in previous blogs, such as this one on recreating faces from facial recognition embeddings and this one on recreating text from sentence embeddings.



This paper will not cover these attacks, but suffice to say that a vector can be returned to a very near approximation of its original form and specifically with text data, often an exact reproduction of the original. These attacks can be conducted with off-the-shelf open source software and a minimum of expertise. Here are some example attacks using vec2text:

Original Sentence	Inversion Result (10 steps)	Notes
Dear Omer, as per our records, your license 78B5R2MVFAHJ48500 is still registered for access to the educational tools.	Dear Omer, As per our records, your license 78B49VH5R2M5R7D is still registered for access to the educational tools.	Nailed the name, including spelling (after refinement), but missed the nonsense "license" though first three letters are correct. 17⁄18 words exactly recovered.
The parent-teacher meeting regarding Alana Masterson (birthdate 4/10/2018) has been rescheduled to October 21 due to a conflict.	The parent/teacher meeting regarding Alison Masterton (birthdate 4/20/2018) has been rescheduled to 21 October due to a conflict.	Missed the first name and got month and year of the birthdate. Got the reschedule date, but reversed the order of month and day. 19⁄21 recovered exactly (counting m/d/y as three and ignoring date order and hyphen vs. slash)
Dear Carla, Please arrive thirty minutes early for your Orthopedic knee surgery on Thursday, April 21, and bring your insurance card and co-payment of $300.	Dear Carla, Please arrive thirty minutes early for your Orthopaedic knee surgery on Thursday, April 21, and bring your insurance card and your co-payment of $300.	Basically nailed everything, but used an alternate spelling of orthopedic and added a "your" to the sentence, which didn't change the meaning. 25⁄25 ignoring added "your" and alternate spelling of orthopedic.

TRAINING ON ENCRYPTED VECTOR EMBEDDINGS

What we need is a way to render the training data safe from a human who has access, including Machine Learning engineers, admins, and hackers. The training data should not be readable or invertible, but should still be useful for creating models.

It turns out that encrypting vector embeddings using approximate-distance-comparison-preserving encryption is a way to do just that: protect the training data while allowing data scientists to create powerful models.

Approximate-distance-comparison-preserving encryption (DCPE) is a way to take a vector embedding and pro-

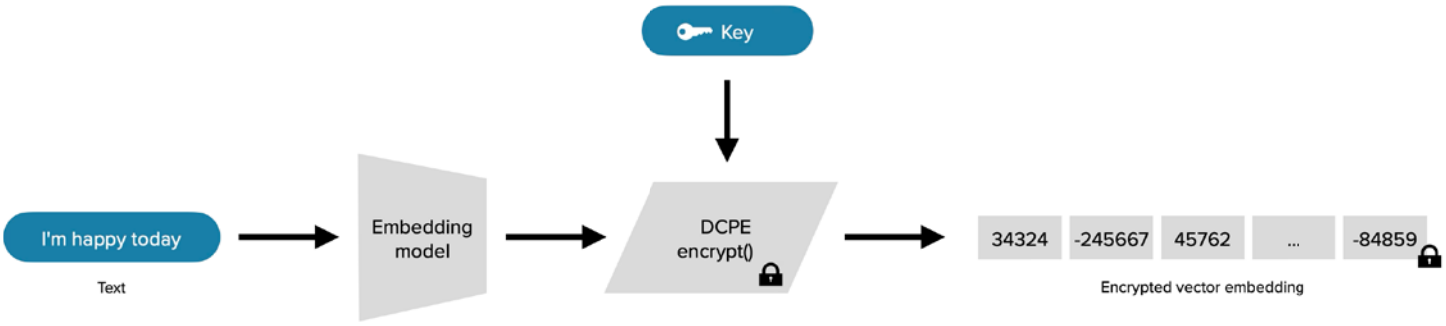
tect the underlying meaning of the vector by preventing inversions. But while encrypted, the vectors still allow comparison of distances so questions like whether two faces are the same person can still be answered. It's a powerful tool that's mostly used in vector search use cases, but can also be applied to training data.

It works like this:

1. **Vectorize:** the production data being used for training gets turned into a vector embedding using an embedding model. This happens in the production environment that has access to the source data.

2. **Encrypt:** in that same environment, a single encryption key is used to encrypt all of those embeddings. If you were building different models on different segments of data, you'd use different keys for each model that was to be created.

3. **Export:** once you have the encrypted vector embeddings, they can be exported into a less secure environment where data scientists can "see" and use the data.



Note 1: in this case, we don't ever want to be able to decrypt the vectors. To make this a one-way operation, when encrypting the vectors, the per-vector random number that is created (which we refer to as an IV), should be discarded. Without that, the vectors can't be decrypted even if you have the key.

From here, a model can be trained off of the encrypted vector embeddings and the encrypted training data can move into the machine learning environment from production without jeopardizing privacy.

Note 2: you can't use encrypted vectors to fine-tune an LLM that was previously trained on unencrypted data. Encrypted vectors are best suited for classification or prediction models rather than text or image generation ones.

USING A MODEL TRAINED ON ENCRYPTED VECTOR EMBEDDINGS

In order to use this new model built on encrypted data, any inputs used for inference must go through the same process as the training data before being run through the model. **Importantly, the inputs must be encrypted using the same key that was used to encrypt the training data.**

Without access to the correct key, the model is useless. Even with the key, the training data is safe.

BENEFIT: REDUCE RISK OF AI DATA LEAKAGE AND THEFT

In most infrastructures – particularly production ones – encryption keys are the most secure part of the infrastructure. They typically live inside key management systems that are tightly controlled, locked down, and backed by hardware security modules (HSMs). When managed properly, keys aren't exportable so there's no mechanism for an angry admin or hacker to copy them.

An intruder cannot use a model unless they also have access to the key, which means the theft of a model is a much less consequential issue provided the key was not also stolen. The same is true of the training data, except then even the theft of the key is not a problem (assuming the IVs were discarded).

This greatly reduces the risk of having these extra copies of private data (counting the training data and the model both as copies).

PROTECTING AGAINST INSIDERS

One potential difficulty in this pattern is that the data scientists need to be able to test the model they create. A simple and secure way to do this is to divide the training data into a training data set and a test data set used to measure the accuracy of the learned model.

Note: additional information associated with each bit of training data would not be encrypted, so in our example, the vectors would be encrypted, but the ratings for the customer support interactions the vectors describe would be unencrypted.

When done in this way, the machine learning team can build and test models without ever seeing any real data, and the key for the model can be restricted to the production environment.

However, if the engineers need to be able to test more organically by running the model in a test environment with ad hoc inputs to more qualitatively assess effectiveness, then we run into an issue. The model is useless without the key, which means the key needs to be copied or made available to the test environment.

While this isn't ideal, the one-way encryption pattern does allow for this to happen without any direct way to recover private data. However, access to the key does open up some attacks, which are discussed below.

ATTACKS ON DCPE

The most effective known attack against DCPE is a "chosen plaintext attack." This is where an attacker is able to generate encrypted vectors from source text or to otherwise assemble a large list of associations between unencrypted text and encrypted vectors under a particular key. (This holds true for other use cases, like facial recognition, images, etc., but we'll stick with our text example to make things clear.)

If an attacker has access to the key, a way to use the key, or a way to observe and record inputs and outputs after use of the key, then this is an attack to understand. And it's because of this attack that keys should be restricted as far as possible.

With a large enough sample (in the tens of thousands at least) of these associations between plaintext and encrypted vectors under a specific key, an attacker can build an inversion model that can attack other vectors encrypted under that key.

The good news here is that the attack results are, at best, fuzzy; and the time and cost of building an attack model specific to a single key are high.

That's because the inversion model is impacted by the per-vector randomness that comes from the IV and is bounded by a parameter of the algorithm called the "approximation factor." This is where the "approximate" in "approximate distance-comparison-preserving encryption" comes into play.

The technical details are out of scope for this paper, but the takeaway is that the higher the approximation factor, the more fuzzy the results of an inversion model attack.

If implementors follow guidelines on minimums, then things like people's names will be nearly impossible to recover accurately, but general themes of text will be recoverable. The higher the approximation factor, the less effective an inversion model will be. But there's a trade-off because higher approximation factors lead to less precise comparison results and less precise models.

Note: allowable precision drops depend on the type of model and types of inputs, so a classification model using broad classifications like health and finance probably won't be impacted by precision loss, but if the model is meant to categorize financial derivative types, which are concepts that may be close together in vector space to begin with, then the "fuzziness" of the vectors will need to be minimized to preserve good results.

We talk about guidelines for approximation factors elsewhere, but there is no easy "just use X" answer since the right approximation factor and the allowable minimum depends heavily on the embedding model used, the range of per-element values that it produces, and how fine-grained comparisons need to be for a given model's domain.

Luckily, even at minimum approximation-factor values, specific details in private conversations are obscured (names, addresses, dollar amounts, and so forth likely can't be retrieved).

MINIMIZING THE RISK OF A CHOSEN-PLAINTEXT ATTACK

In any case, when using DCPE, the key(s) used to encrypt training data should be tightly controlled, and engineers (data scientists and developers) should not be given direct access to it.

Instead, keys should only be usable by code running in specific environments, and there should be review processes in place where multiple team members know what code is being deployed into the environment. This will greatly minimizes the risk that someone could create a model to attack a specific key.

As with most real-world security, there's a need for a mixture of technical, procedural, and policy measures to provide a robustly secure and privacy-preserving system. For the more sensitive data, an organization may choose to only allow access to the relevant key from production code in a production environment where the strictest access controls and policies are enforced.

ALTERNATE APPROACHES

There are other ways to encrypt data in models that also require a specific key to use the model. The most notable alternate approach comes from the family of fully homomorphic encryption (FHE).

This is cool technology and a viable approach for many use cases. The trade-off versus the approach outlined above is one of performance (meaning here the amount of time it takes to train a model or generate an inference). FHE requires quite a lot more processing power, introduces latency, and is not feasible for large models (no one has yet produced an FHE LLM that is usable). The FHE approach also requires special engines to run and use the model, which restricts choices for libraries, types of models, and so on.

The partially homomorphic encryption technique used by DCEP has these advantages over FHE:

- The training data contents (and by implication, the data embedded into the model) are encrypted, not just the model itself
- Models can be run anywhere they would be without encryption, using any software stack (pytorch, tensorflow, rust's burn, and so on)
- The performance is strong, with the overhead being mostly around generating embeddings. The encryption takes negligible time in comparison
- It works with arbitrarily large models and training sets

CONCLUSION

Building AI models on private data doesn't need to introduce new risks for the training data. SaaS companies can use these approaches to make models using customer data without risking that data (provided the model results can't potentially leak data). One-way encrypting vector embeddings and using them as training data is a viable approach to preserve privacy and increase security for many use cases, particularly in classification-style domains.

For those interested in this sort of encryption, take a look at IronCore Labs' Cloaked AI, which is open source under the AGPL license or available under a paid commercial license. It makes it pretty easy to encrypt vectors in a variety of programming languages and it's free to try and test. The engineers hang out in a Discord channel and answer questions there, too.

IRONCORE LABS

ABOUT IRONCORE

IronCore Labs, the leading provider of application data security for modern cloud applications, offers a platform to help software developers and businesses rapidly add application-layer encryption into the heart of their software. The platform protects data wherever it lives and allows it to still be used with encrypted search solutions for keyword indices and vector databases. By protecting the "memory of AI," IronCore Labs is at the forefront of ground breaking new technology to handle quickly emerging security issues while empowering cloud software companies to move fast in adopting new technologies. IronCore's solutions help customers meet data privacy regulations, address international transfer limitations, and fulfilling hold-your-own key (HYOK/BYOK) requests to build trust and profitable relationships with their customers.

IronCore Labs
1750 30th Street #500
Boulder, CO 80301, USA

Inquiries
Email: info@ironcorelabs.com
Phone: +1.415.968.9607

CONNECT WITH US

- ironcorelabs.com/blog
- linkedin.com/company/ironcore-labs
- twitter.com/ironcorelabs
- ironcorelabs.com