Modern notes

Nareli Cruz Cortés

October 2, 2025

1 Modern C in the context of PosQuantum Cryptography

Of course I understand the logic and tricks to code in C, but it has been a while since the last time.

1.1 Language Standard

- C99: Introduced fixed width integers as uint32_t and uint64_t, inline functions and variable-lenght arrays.
- C11 Better multithreading support, atomic operations, and bounds-checked functions.
- C17 bug-fix of C11

1.2 Safe coding practices

- avoid old unsafe constructs like raw string handling with strcpy
- uses stdint.h fixed-width types for portability across platforms
- use static inline functions instead of macros when possible

1.3 Security-aware coding

It is important to ensure constant-time implementations to prevent side-channel attacks. Also memory cleansing functions as wiping secrets from the RAM. All these should be found somewhere as good practices, or so.

1.4 Compiler support and tooling

Use modern compilers such as Clang, GCC, MSVC and enabling warnings and sanitizers -Wall -Wextra -fsanitize=address, undefined Also use static analyzers like Coverity, clang tidy to catch memory issues.

2 Project for Practicing Python and Modern AI to be in the loop

Beginner-Level: Core Python + AI Fundamentals

- Build a Perceptron from Scratch: Implement the simplest neural network without libraries, just NumPy. Helps you understand gradient descent, activation functions, etc.
- Handwritten Digit Classifier (MNIST): First implement it with only NumPy, then re-do with PyTorch/TensorFlow.
- Text Sentiment Analyzer (Bag-of-Words → Transformer): Start with a classic method (logistic regression with scikit-learn), then move to fine-tuning a transformer like DistilBERT.

Intermediate-Level: Modern AI Practices

- **Image Classifier with Transfer Learning**: Use a pretrained model (e.g., ResNet, EfficientNet) and fine-tune on a custom dataset (e.g., cats vs. dogs, or your own photos).
- **Chatbot with Retrieval-Augmented Generation (RAG)**: Combine a large language model (like GPT via Hugging Face) with a vector database (e.g., FAISS, ChromaDB) to answer domain-specific questions.
- **Time-Series Forecasting with Deep Learning**: Predict stock prices, weather, or energy consumption using LSTMs, GRUs, or Transformers.

Advanced-Level: Cutting-Edge AI

- **Implement a Tiny Transformer from Scratch**: Write your own mini GPT in Python + NumPy. Great for mastering attention mechanisms.
- **Diffusion Model for Image Generation**: Implement a simplified diffusion model, then use the `diffusers` library for stable diffusion fine-tuning.
- **Reinforcement Learning Agent (Gymnasium)**: Train an AI agent to play CartPole, then scale up to Atari games using PPO or DQN.
- **AI Deployment Pipeline**: Build an end-to-end pipeline: model training → Docker container → REST API with FastAPI → deployment on cloud (AWS/GCP/Render).

Extra Practice for Staying in the "Modern AI Loop"

- Experiment with LLM APIs (OpenAI, Hugging Face, etc.): Wrap them in apps (chatbots, assistants, summarizers).

- Explore Vector Databases + RAG: This is currently huge in enterprise AI.
- MLOps Mini-Project: Automate training, evaluation, and deployment using tools like MLflow or Weights & Biases.

Tip: Cycle between

1. From-scratch coding (deep understanding),
2. Using modern libraries (staying current),
3. Deployment/Integration (real-world practice).

Don't forget pip install scikit-learn

3 Perceptron

See the code perceptronsimple.Py
Observations:

- Weights initialization: all zeros, last weight is bias.
- Activation: step function (threshold at 0).
- Training: perceptron rule:

$$w \leftarrow w + \eta(y - \hat{y})x$$

- Bias: added by augmenting input with constant '1'.
- Example: Trains to learn logical AND.
- The order of the input affects the final output.
- The input size is 2 and the size of the training set is 4.
- Inputs in functions are only defined by name, not type.
- **The model is drawn in 2D!**

4 MNIST

I need to program a classifer for supervised dataset MNIST. For very well-known datasets, like this one, TensorFlow-Keras or Pytorch the dataset comes built-in, which means the functions to download and fetch the dataset are in the library already. To prepare the data:

- Reshape : do it a 1D Tensor
- Normalize: COnvert the pixel values range values from 0-255 into 0-1 to get better managment and faster convergence.
- One-hot encode labels, example 2 becomes [0,0,1,0,0,0,0,0,0,0]

I will try to solve it with a multilayer neural network with the following configuration:

- 784 - 64- 10
- activation funtions: ReLU in the hidden and Softmax in the output.
- Loss function categorical cross=entropy and the optimizer Adam.

To evaluate test on unseen test data and measure the accuracy. I can chose between Pythorch and TensorFlow-Keras. This time I get TensorFlow.

- Which are the models in Keras? Sequential (Sequential) and Functional (Model)
- Dense, Flatten , Input are part of layers
- To compile, three elememts are need, at least optimizer, loss, and metrics (accuracy)
- To fit the model: x,y to train of course, epochs, batch sizes, and validation split ratio
- Then, evaluate the model, by computing evaluate and getting lost and accuracy.
- The categorical_crossentropy requieres the target labels in one-hot format.

5 Text Sentiment Analyzer

TF-IDF: Term Frequency - Inverse Document Frequency: Term Frequency (TF) measures how often a word appears in a document. And Inverse Document Frequency (IDF) This measures how important a word is across a whole collection of documents. TF-IDF helps computers "see" which words are important in a document compared to a whole collection, by balancing frequency and uniqueness.

- The linear_model options are: logistic regression
- I am using scikit-learn.

end-to-end scikit-learn example that builds a text sentiment analyzer using a Transformer (TF-IDF vectorizer) + Logistic Regression inside a Pipeline. It includes train/test split, simple hyperparameter search, evaluation, and saving/loading the model. Notes Transformer here refers to scikit-learn's feature transformer (TfidfVectorizer), which converts raw text to numerical features. If your labels are not binary, Logistic Regression still works for multi-class (via one-vs-rest or multinomial with s solver='lbfgs' and multi_class='auto').

For real projects, feed a larger dataset (e.g., your CSV of reviews), and consider cleaning text, handling emojis, lowercasing, etc. If you need probability thresholds (e.g., abstain when uncertain), use predict_proba and set a custom cutoff (default decision threshold is 0.5 for binary). If you want a version that streams data from a CSV and does cross-validation on a bigger grid, say the word and I'll adapt this to your files.

Pipeline is useful to do something like encapsulating the preprocessing and model to get a uniform consistent process. It contains the preprocessing instructions, and the fit as well, so when training or testing the pipeline applyed exactly the same treatment to all the processes.

6 Image Classifier with Transfer Learning 

I will use DeiT-S (2020). In this Strategy the goal is to take pretrained transformers and used the already trained weights. It can be executed with PyTorch because it has preloaded the data, It can be done using GPU or cpu.

The library timm is a PyTorch Image Models library (by Ross Wightman) with a huge zoo of pretrained vision models, plus training utilities, losses, schedulers, and data pipelines that make transfer learning and benchmarking fast:

- Hundreds of pretrained models: ResNet, ConvNeXt, EfficientNet, RegNet, MobileNet, ViT, DeiT, Swin, EVA, CaiT, MaxViT, etc.
- One-line model factory: timm.create_model(name, pretrained=True, num-classes=...)
- Ready transforms matched to each model's weights: input size, mean-std, interpolation.
- Feature extraction: create models that output penultimate features for classic ML or heads you add.
- Training bits: optimizers, schedulers (cosine, one-cycle, etc.), mixup-cutmix, EMA, AMP helpers.
- Utilities: list/search models, pretrained weight download, model export helpers.

7  SCIkit-Learning and PyTorch

They complement each other nicely because they shine at different parts of the ML workflow. Why combine them?

- Preprocessing & feature engineering (scikit-learn) Robust, declarative tools: Pipeline, ColumnTransformer, StandardScaler, OneHotEncoder, imbalanced sampling (imblearn), etc.
- Model selection & evaluation (scikit-learn) Easy train-val splits, cross-validation, Grid Random Bayes search, and rich metrics curves: roc_auc_score, precision_recall_curve, confusion_matrix, calibration, permutation importance.
- Flexible modeling (PyTorch) When you need deep nets, custom losses, multi-task heads, or GPU acceleration, PyTorch is the go-to.
- Best of both worlds Use sklearn to manage data & search space; use PyTorch for the model. This gives you clean, leak-free pipelines plus state-of-the-art models.

8 Transformer

9 Vector Databases + RAG (Retrieval-Augmented Generation)

The Core Challenge

Large Language Models (LLMs) like GPT are powerful, but they:

Don't know everything (knowledge cutoff). Can "hallucinate" facts. Struggle to stay up to date with fast-changing information.

So we need a way to ground the model in reliable, external knowledge at query time.

Vector Databases: The Memory Engine

A vector database is a specialized system that stores and retrieves information not as plain text, but as vectors (dense numerical representations of meaning).

A piece of content (e.g., a document, paragraph, or image) is converted into a vector embedding by an embedding model. Similar items live close together in this high-dimensional space. This enables semantic search: finding things based on meaning, not keywords.

Example: "doctor" and "physician" - vectors close together. "doctor" and "dog" - far apart, even though textually similar.

3. RAG: Retrieval-Augmented Generation

RAG is an AI pattern that couples **retrieval (search) with **generation** (LLM).

Workflow:

1. User asks a question - “What are the side effects of drug X?” 2. System searches the vector DB for relevant documents (clinical notes, FDA docs, etc.). 3. Top results are fed into the LLM** as additional context. 4. LLM generates an answer that’s grounded in retrieved information.

Result: The model is not just guessing — it’s reasoning **with context** retrieved in real time.

4. Why They Work Well Together Vector DB = external knowledge store, efficient at semantic search. RAG = method to inject that knowledge into the LLM at the right moment.

Together, they form a feedback loop:

Ask -Retrieve relevant facts -¿ Generate grounded answer.

5. Benefits

Freshness**: You can update the database without retraining the LLM. Accuracy**: Reduces hallucinations by citing real sources. Control**: You decide what knowledge is trusted (internal docs, curated data). Scalability**: Efficiently handles millions/billions of data chunks.

High-level analogy**: Think of the LLM as a brilliant student with a strong reasoning brain but **no access to books after 2023**. A vector database is like a **super-fast library catalog** where books are organized by meaning. RAG is the act of saying: *“Go fetch the 3 most relevant books, skim them, and then answer my question using those sources.”*

DistilBERT

What is DistilBERT?

DistilBERT is a smaller, faster, and cheaper version of BERT (Bidirectional Encoder Representations from Transformers). It was introduced by Hugging Face in 2019 and is trained using a technique called **knowledge distillation**.

In short, DistilBERT retains about 95–97% of BERT’s performance, while being 40% smaller and 60% faster.

Background: BERT

BERT is a large Transformer-based model trained on massive text corpora. It achieves state-of-the-art performance on many NLP tasks (e.g., sentiment analysis, text classification, question answering). However, BERT is computationally expensive: it has many parameters, slow inference, and is difficult to deploy on small devices.

How DistilBERT is Made

DistilBERT is created using **knowledge distillation**:

- The *teacher model* is BERT.
- A *student model* (smaller Transformer) is trained to mimic the teacher.
- The student learns not only from the raw labels but also from the teacher's "soft" outputs (probability distributions).

This allows DistilBERT to preserve most of BERT's semantic understanding, but with fewer layers (6 instead of 12).

Advantages

- **Faster inference** - useful for real-time applications.
- **Smaller size** -can run on mobile and edge devices.
- **Cheaper computation** - reduced training and serving costs.

Python Example

```
from transformers import pipeline

Load pretrained sentiment analysis model
classifier = pipeline("sentiment-analysis",
                      model="distilbert-base-uncased-finetuned-sst-2-english")

print(classifier("I love using DistilBERT!"))
[{'label': 'POSITIVE', 'score': 0.999}]
```

Applications

DistilBERT can be used in most tasks where BERT is applied:

- Sentiment analysis
- Spam detection
- Topic classification
- Named Entity Recognition (NER)
- Question answering

Summary: DistilBERT is essentially BERT's lightweight twin — almost as accurate, but much faster and easier to deploy.

Table 1: Widely used pretrained image classifiers for transfer learning (typical 224×224 input).

Model	Year	Params	FLOPs	Top-1*	Strengths	Best for
AlexNet	2012	∼61M	∼0.7G	∼57%	Historic baseline; simple to fine-tune	Teaching, quick prototypes
VGG-16	2014	∼138M	∼15.5G	∼71%	Very simple blocks; strong features	Feature extraction baselines
Inception-v3	2015/16	∼24M	∼5.7G	∼78%	Efficient via Inception modules	Mid-size tasks with limited compute
ResNet-50	2015	∼25.6M	∼4.1G	∼76–77%	Stable training; great general baseline	General-purpose transfer learning
DenseNet-121	2016	∼8M	∼2.9G	∼74–76%	Parameter-efficient; strong features	Compact models with good accuracy
MobileNet-v2	2018	∼3.4M	∼0.3G	∼71%	Very small/fast; depthwise convs	Edge/mobile, real-time apps
EfficientNet-B0	2019	∼5.3M	∼0.39G	∼77–78%	Excellent acc/compute trade-off	Accuracy + efficiency balance
EfficientNet-B4	2019	∼19M	∼4.2G	∼82%	Scales well (width/depth/res)	Higher-accuracy with moderate cost
ViT-B/16	2020	∼86M	∼17G	∼79–81%	Transformer flexibility; long-range deps	Large-data fine-tuning, research
DeiT-S	2020	∼22M	∼4.6G	∼79%	Data-efficient ViT; ImageNet-only pretrain	Strong modern baseline without huge data

Notes: Params/FLOPs are approximate and vary by implementation, input resolution, and head. **Top-1*** refers to vanilla ImageNet-1k single-crop accuracy reported in common repos; your mileage will vary with training tricks and fine-tuning setup.

Table 2: Recommended pretrained image classifiers by transfer-learning scenario.

Scenario	**Recommended pick(s)**
General-purpose baseline	ResNet-50
High accuracy with good efficiency	EfficientNet-B0/B3/B4
Mobile / edge deployment	MobileNet-v2/v3
Very small model (fast prototyping)	MobileNet-v2, EfficientNet-B0
Compact but strong features	DenseNet-121
Limited compute, mid-size tasks	Inception-v3
Cutting-edge transformer baseline	ViT-B/16, DeiT-S
Tiny datasets (freeze most layers)	ResNet-50, VGG-16 (feature extractor)
Large-data fine-tuning / research	ViT-B/16 (or larger), EfficientNet-B4/B7
Real-time inference constraints	MobileNet-v3, EfficientNet-B0