PAUL SCHERRER INSTITUT

PSI

Midjourney v5.2

Stefan Ritt :: Head of Muon Physics Group :: Paul Scherrer Institut

Modern Engineering and AI (*ChatGPT for Scientists*)

Unveiling the Power and Potential of Large Language Models

IEEE NPSS Prague EduCom International Summer School

July 1st, 2025, Prague, Czech Republic

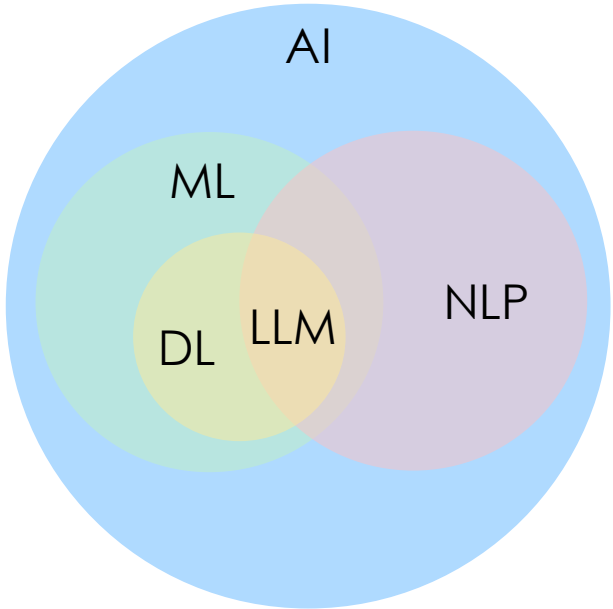
Engineering and Large Language Models

- **AI** can be used in **many ways** these days
  – Neural Nets (NN) in **data analysis**
  – Data **compression** in Edge Computing

- This talk: Use Large Language Model (**LLM**) to write Python code for quick data analysis with **Jupyter Notebook**
  – Introduction to Jupyter Notebook
  – Ask **ChatGPT** to write code to **plot data** and do **analysis**
  – **Understand** and improve **code**

Terms

- **AI** - Artificial Intelligence
- **ML** - Machine Learning
- **DL** - Deep Learning
- **NLP** – Natural Language Processing
- **LLM** – Large Language Model

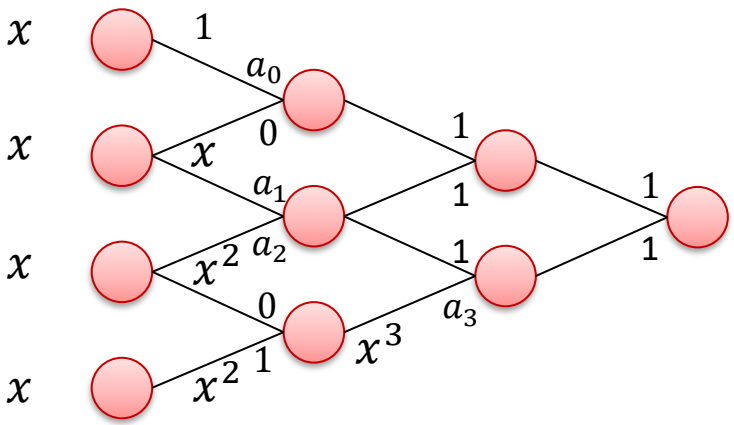


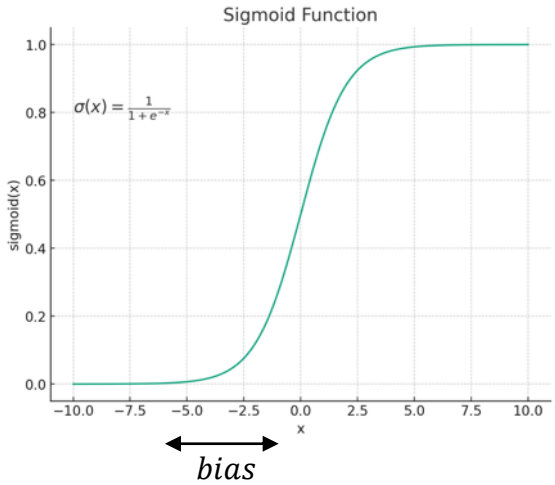
Neural Net - Polynomial

- **Polynomials** can fit **any** function
- A neural net is a kind of polynomial
  - node adds inputs and applies activation function like in our brain

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

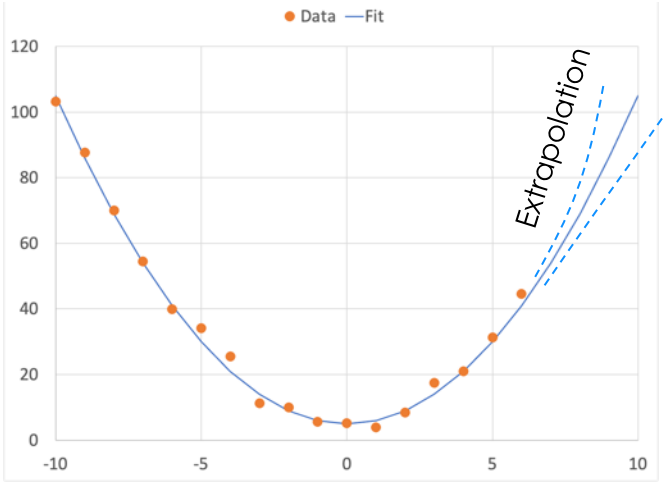
$$\bigcirc = f\left(\left[\sum input_i\right] + bias\right)$$

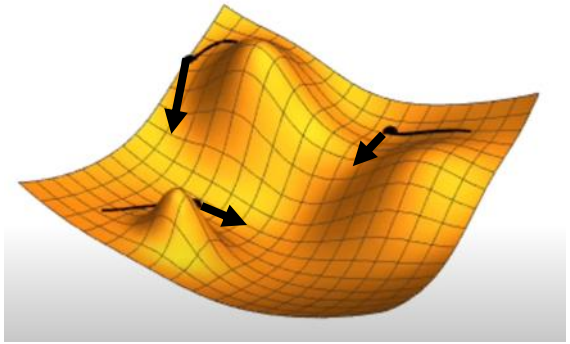




Machine Learning

- **Training** a Neural Network is very similar to **fitting** a polynomial to a function:
  - Create a χ^2 and minimize it
  - Gradient method: calculate partial derivations of all parameters and adjust them





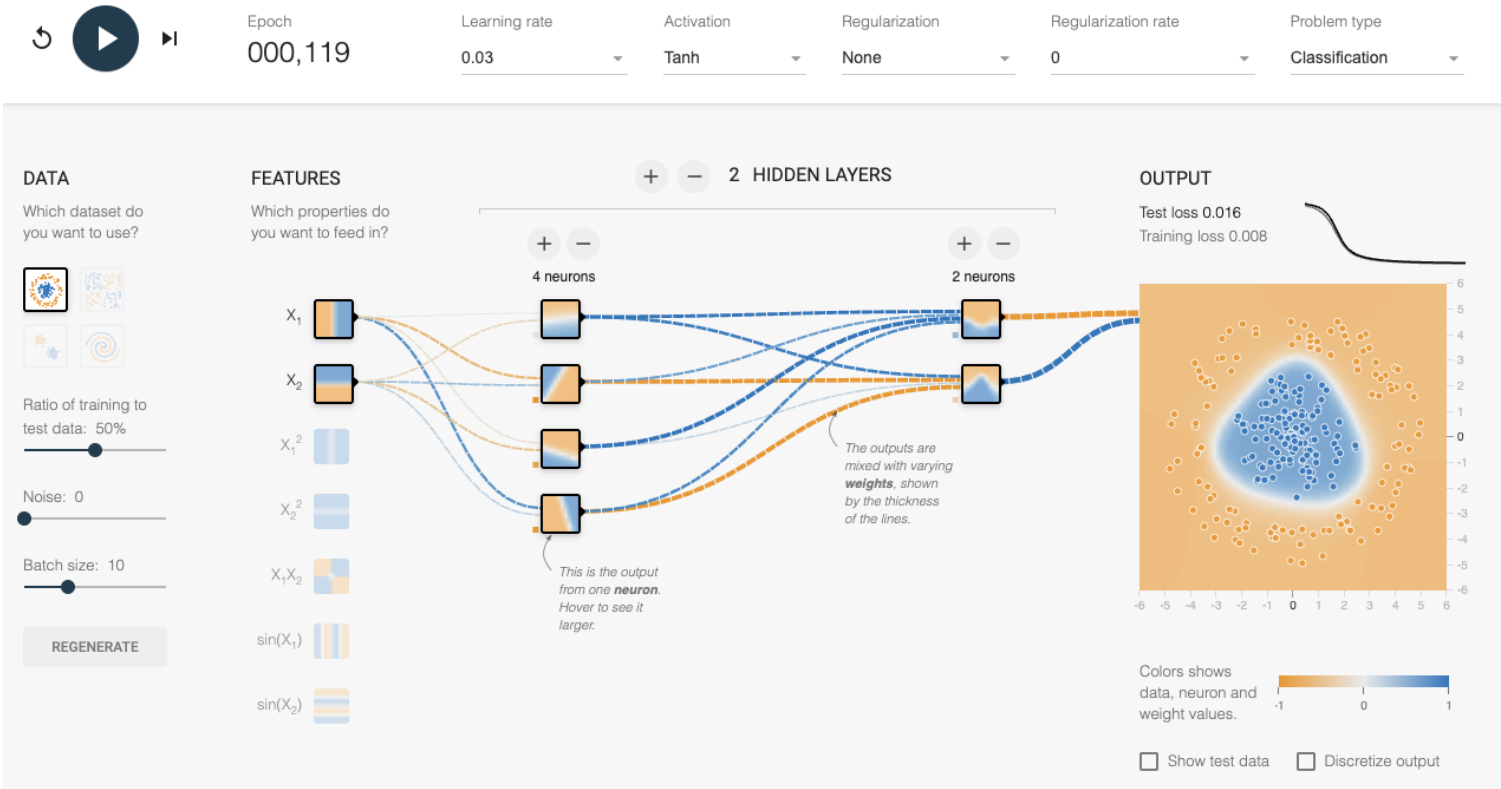
$$\chi^2 = \sum (f(x) - y)^2 \rightarrow min$$

$$a_i' = a_i + \gamma \cdot \frac{\partial \chi^2}{\partial a_i}$$

"learning rate"
~0.001

Interactive Training Demo

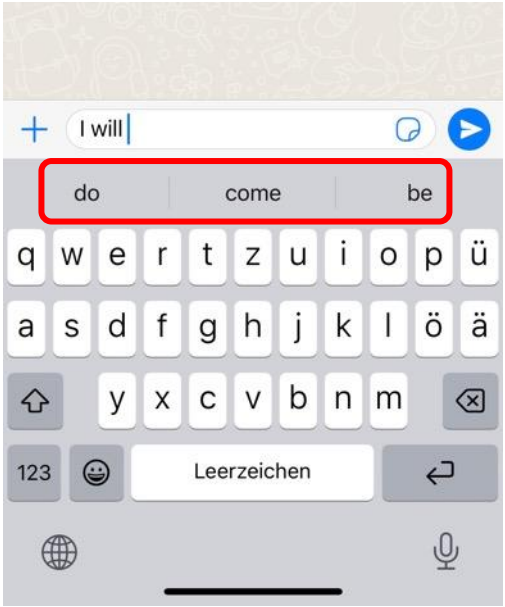
https://playground.tensorflow.org

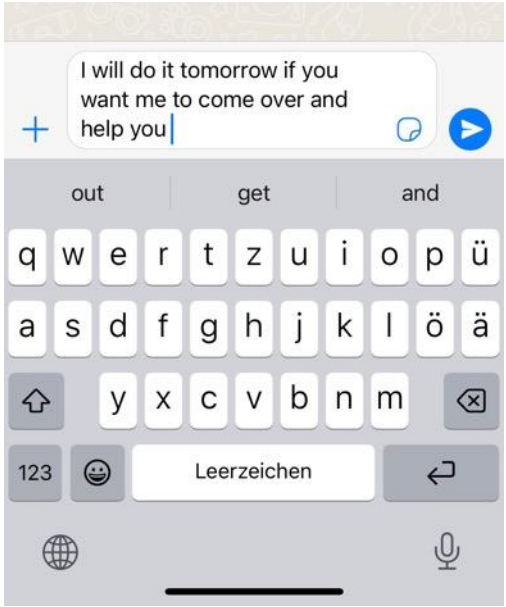


Text generation

Text can be **predicted** using **probabilities**, see your smartphones autocomplete function:

I will
- do *13.4 %*
- come *11.2 %*
- be *8.1 %*





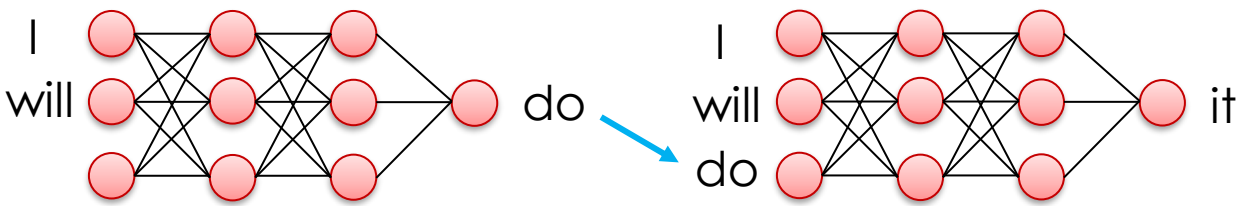
Text generation

- How **big** a **table** with probabilities do we **need** ?
- Assume we have **10,000** words and make a sentence with **10 words**:

word word word word word word word word word word

10^4 x 10^4 x 10^4 x 10^4 x 10^4 x 10^4 x 10^4 x 10^4 x 10^4 x 10^4 = 10^{40}

- Probability table with **10^{40}** entries is **not possible**
- Most of the 10^{40} sentences make no sense
- All meaningful sentences are **still too big** to fit in a table
- → **Approximate** the probability table with a **Neural Net**



Machine Learning

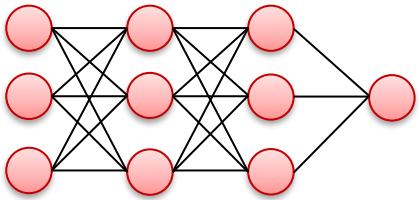
Labeled learning:

Cat

Cat

Dog

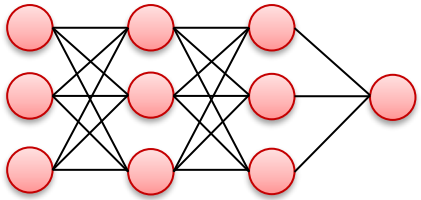
Cat



Cat

Un-labeled learning:

I will do



it

→ Basis of Large Language Models

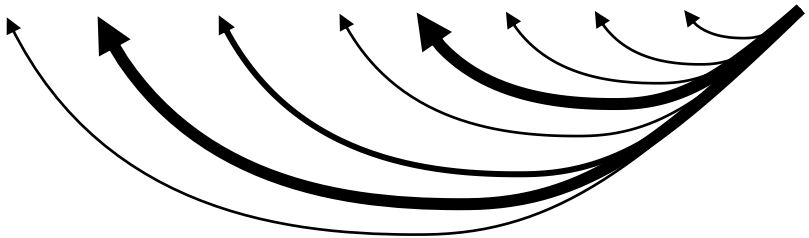
Meaning of words in sentence

- **Meaning** of words in a sentence **depend** on their **order**

  – **Goliath** has defeated **David**
  – **David** has defeated **Goliath**

- **Meaning** of words in a sentence **depend** on their **context**

  – I walk across the street to the other **bank** → *financial institution*
  – I swim across the river to the other **bank** → *land along a river*



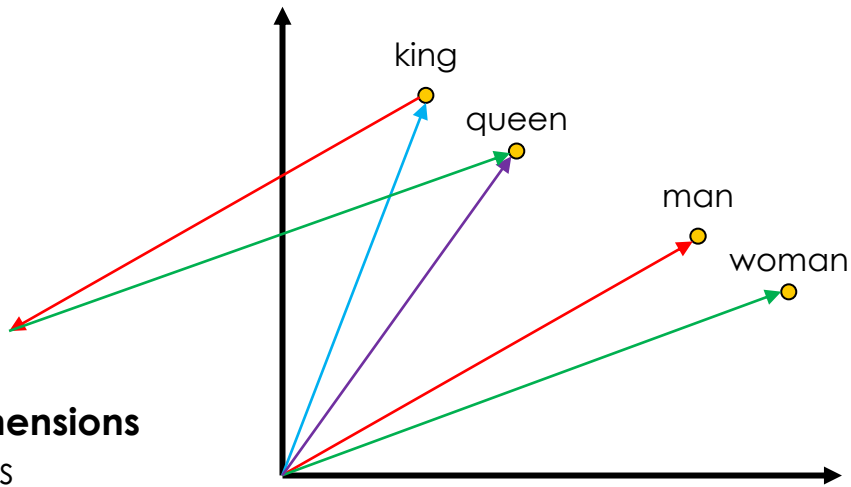
Words have **different weights** to influence the **context**

Word embedding

- **Neural nets need numbers** as input
- Simplest method: Give each word a **unique number**
- Better method: Encode **words** in a **vector** v_i

man	woman	king	queen
0.72	0.81	0.23	0.32
0.32	0.28	0.63	0.59
…	…	…	…

king – man + woman = queen



- **Typical** language models use **i=512 dimensions**
- Similar words should have similar vectors
- Coordinates are derived from **training**

Position encoding

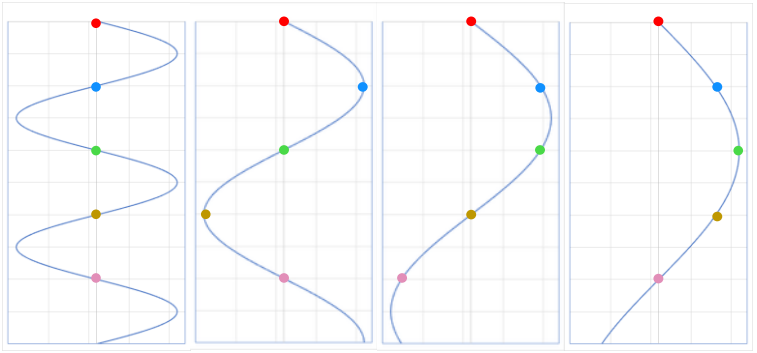
d =	David	has	defeated	Goliath
0	0.61	0.71	0.13	0.20
1	0.08	0.01	0.15	0.80
2	0.36	0.01	0.35	0.26
3	0.08	0.78	0.96	0.72
4	0.57	0.62	0.66	0.59
	p=1	2	3	4

+

sin(d/p*π)			
0.00	0.00	0.00	0.00
0.00	1.00	0.87	0.71
0.00	0.00	0.87	1.00
0.00	-1.0	0.00	0.71
0.00	0.00	-0.8	0.00

=

David	has	defeated	Goliath
0.61	0.71	0.13	0.20
0.08	1.01	1.02	1.50
0.36	0.01	1.22	1.26
0.08	-0.22	0.96	1.42
0.57	0.62	-0.20	0.59



0.00	0.00	0.00	0.00
0.00	1.00	0.87	0.71
0.00	0.00	0.87	1.00
0.00	-1.0	0.00	0.71
0.00	0.00	-0.8	0.00

Note:
Current models use alternating sin/cos and different frequencies, so distance of words can be easily calculated from vectors.
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Position encoding

d =	David	has	defeated	Goliath
0	0.61	0.71	0.13	0.20
1	0.08	0.01	0.15	0.80
2	0.36	0.01	0.35	0.26
3	0.08	0.78	0.96	0.72
4	0.57	0.62	0.66	0.59
	p=1	2	3	4

+

sin(d/p*π)			
0.00	0.00	0.00	0.00
0.00	1.00	0.87	0.71
0.00	0.00	0.87	1.00
0.00	-1.0	0.00	0.71
0.00	0.00	-0.8	0.00

=

David	has	defeated	Goliath
0.61	0.71	0.13	0.20
0.08	1.01	1.02	1.50
0.36	0.01	1.22	1.26
0.08	-0.22	0.96	1.42
0.57	0.62	-0.20	0.59

Goliath	has	defeated	David
0.20	0.71	0.13	0.61
0.80	0.01	0.15	0.08
0.26	0.01	0.35	0.36
0.72	0.78	0.96	0.08
0.59	0.62	0.66	0.57

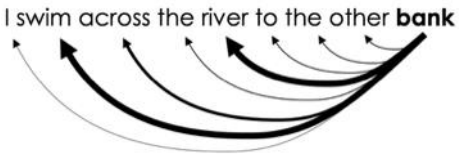
+

sin(d/p*π)			
0.00	0.00	0.00	0.00
0.00	1.00	0.87	0.71
0.00	0.00	0.87	1.00
0.00	-1.0	0.00	0.71
0.00	0.00	-0.8	0.00

=

Goliath	has	defeated	David
0.20	0.71	0.13	0.61
0.80	1.01	1.02	0.79
0.26	0.01	1.22	1.36
0.72	-0.22	0.96	0.79
0.59	0.62	-0.20	0.57

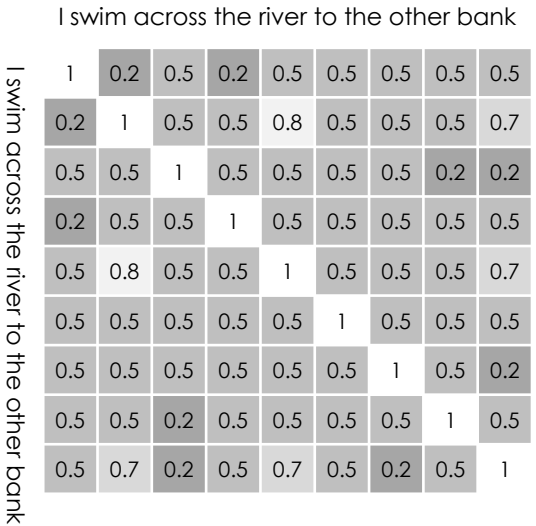
Context encoding

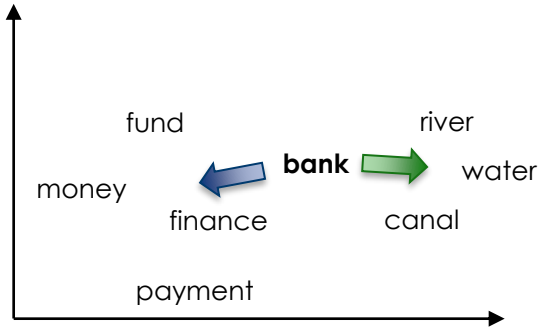


- **Context** matters!
- We need a **mechanism** that **considers** neighboring words to **enhance** the **meaning** of the word of interest
- Idea: use vector **dot product** of vectors
  → large for **similar word**
  → zero for perpendicular vectors

v: word vector
s: similarity

$$s_{ij} = v_i v_j^T$$



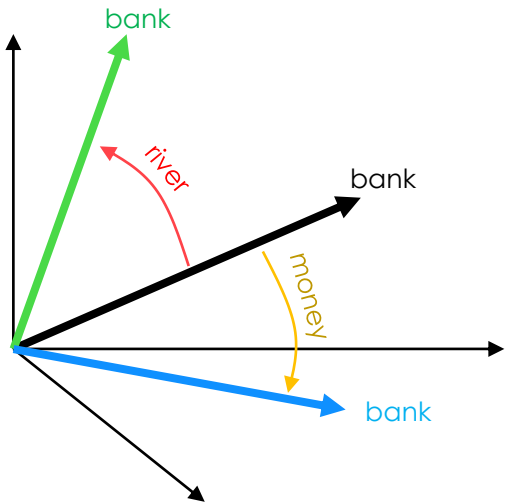


Vector rotation

- Word **vectors** v_i in 512-dim space
- **Similarities** $s_{ij} = v_i v_j^T$
- **Weights** $w_{ij} = softmax(s_{ij}) = \frac{e^{s_{ij}}}{\sum e^{s_{ij}}}$ $\sum w_{ij} = 1$
- Modify word vectors by **matrix multiplication**

$$y_i = v_i \times w_{ij}$$

- Word vectors are **rotated** in 512-dim space, influenced by neighboring words
- If a word has **similar neighbors**, it's **rotated similarly**



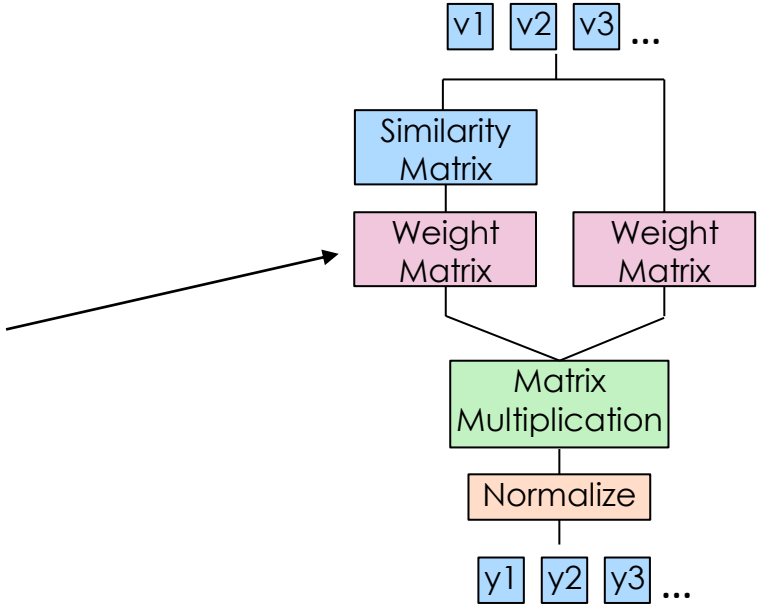
Self-Attention Mechanism

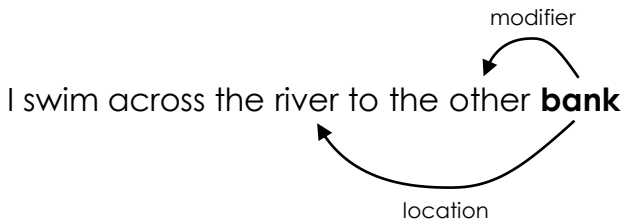
- Word vectors
  → Similarity Matrix
  → Rotation (Matrix Multiplication)
  → Normalize Matrix
  → **Contextualized representation of word vectors**

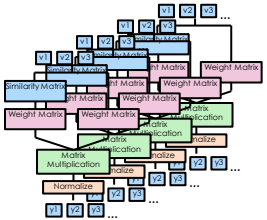
- Add Machine Learning by **weighting original** words with **matrix of weights** derived from training

  → **Self-Attention Mechanism**

- Put several Self-Attention Mechanism in parallel
  → **Multi-headed** Self-Attention Mechanism

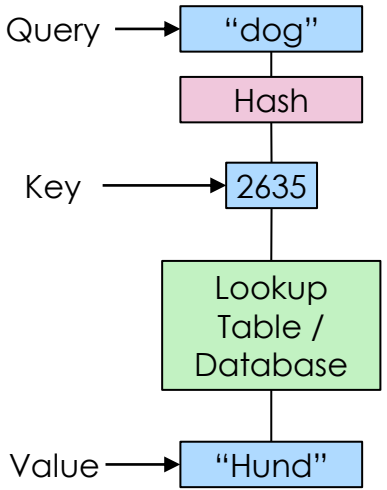




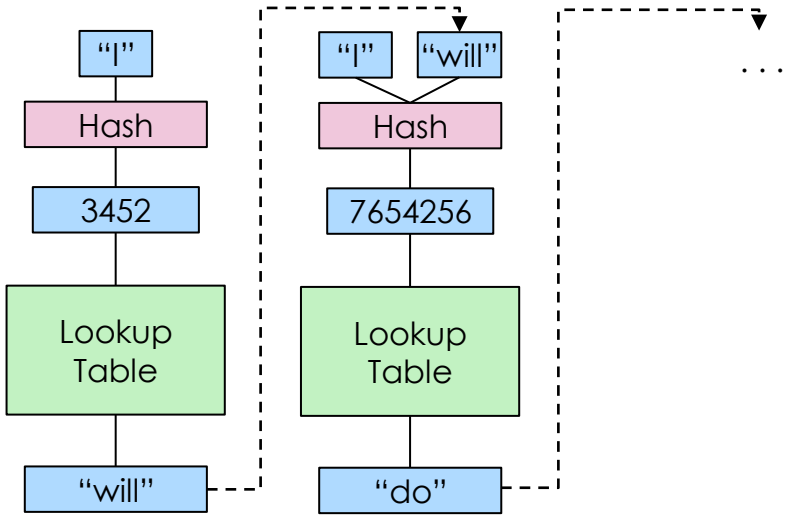


Traditional Database Lookup

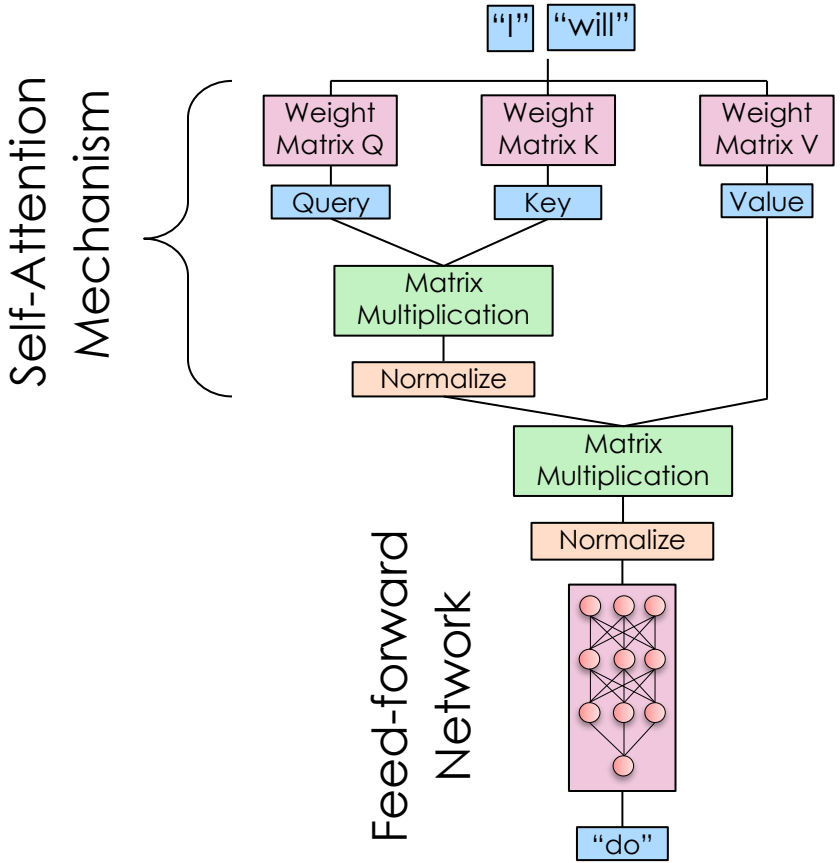
Lookup for translations



Lookup for text generation



Lookup with Neural Nets



Putting all together: **Transformer**

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

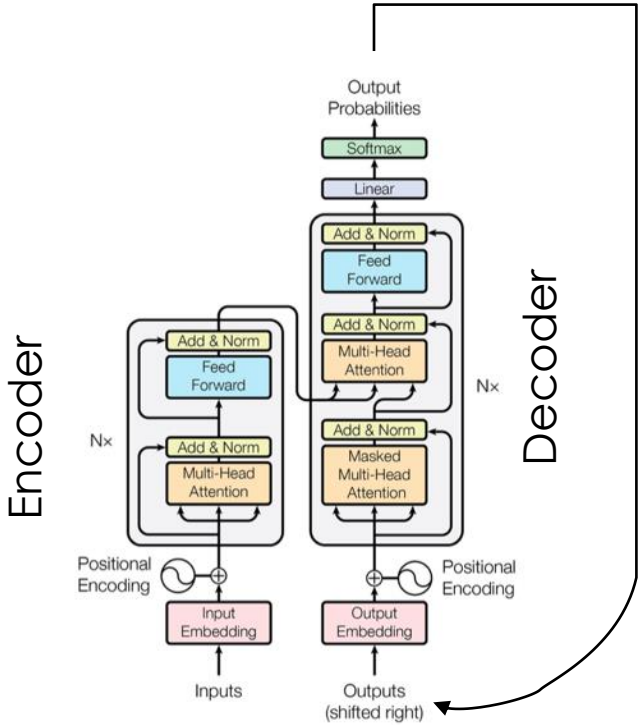
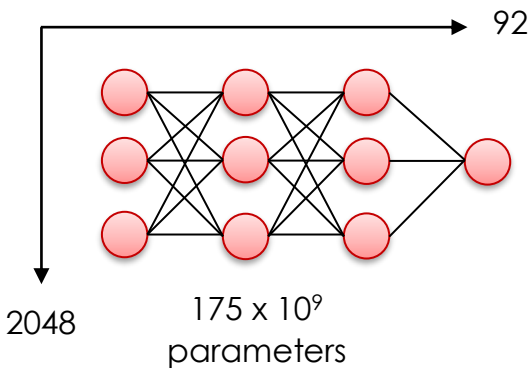


Figure 1: The Transformer - model architecture.

https://arxiv.org/abs/1706.03762
June 2017

ChatGPT 3

- Transformer Mechanism + Huge Neural Network + … → Large Language Model (**LLM**)
- https://en.wikipedia.org/wiki/GPT-3
- **2048 Tokens** (2-chars + <EOS> + punctuation)
- **92** hidden network layers
- **175 Billion** Parameters, 800 GB of storage
- Trained with **1.8 PB** of text (500 B tokens)
  - 3% Wikipedia
  - 16% Books1 & Books2
  - 81% Common Crawl + WebText2
- **305 Years** for training →
  ~ months on huge cluster, cost ~**5 M$**
- Training data until **Sept. 2021**
- **Pre**-training plus **fine**-tuning:
  Generative Pre-trained Transformer 3 (**GPT 3**)
- Free version of ChatGPT at https://chat.openai.com



Hallucination

- https://en.wikipedia.org/wiki/Hallucination_(artificial_intelligence)

- ChatGPT is good to generate **nice text**, but NOT to produce **correct output**

- **Is 17077 a prime**? (correct answer: **Yes**)
  - Solution is **not present** in training data
  - Solutions **learned**:
    - 17 is prime since it can only be divided by 1 and 17
    - 77 is not a prime, it can be divided by 7 and 11
  - **Hallucination:**

"17077 is **not** a prime, since it can be divided by 7, 11 and 17"

- LLM cannot deal with a situation where it is **not trained in**

Important Lesson to be learned

The output of a Large Language Model
is "**mostly**" correct

Use it for cases where you can **double check** or
do not rely on the correctness of the output

"Stammtisch" (pub) discussion

LLM cannot calculate

ChatGPT 3.5

Is 17077 prime?

No, 17077 is not a prime number. It can be divided evenly by several numbers other than 1 and itself. In particular, it can be divided by 7 and 2441, among other divisors. Therefore, 17077 is not a prime number.

ChatGPT 4

Is 17077 prime?

To determine if 17077 is prime, one has to check if it is divisible by any integer other than 1 and itself.

However, we can use a shortcut to reduce the number of checks we make. We only need to test divisors up to the square root of 17077, because if 17077 has a divisor larger than its square root, then it must also have a divisor smaller than its square root.

The square root of 17077 is a bit above 130. Therefore, we need to check if 17077 is divisible by any prime number less than 130.

Let's start with the first few prime numbers:

2: 17077 is odd, so it's not divisible by 2.

3: $1 + 7 + 0 + 7 + 7 = 22$, and 22 is divisible by 3. Therefore, 17077 is divisible by 3.

This shows that 17077 is not prime, since it's divisible by 3.

17077 / 3 = 5692.333…

Code plug-in *can* calculate

Is 17077 prime?

Note: Requires paid version of ChatGPT

Finished working Hide work

python Copy code

```python
def is_prime(n):
    """Check if a number is prime."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

# Check if 17077 is prime
is_prime_17077 = is_prime(17077)
is_prime_17077
```

RESULT

```
True
```

The number 17077 is prime.

ChatGPT 4 +
Code Interpreter

PAUL SCHERRER INSTITUT

PSI

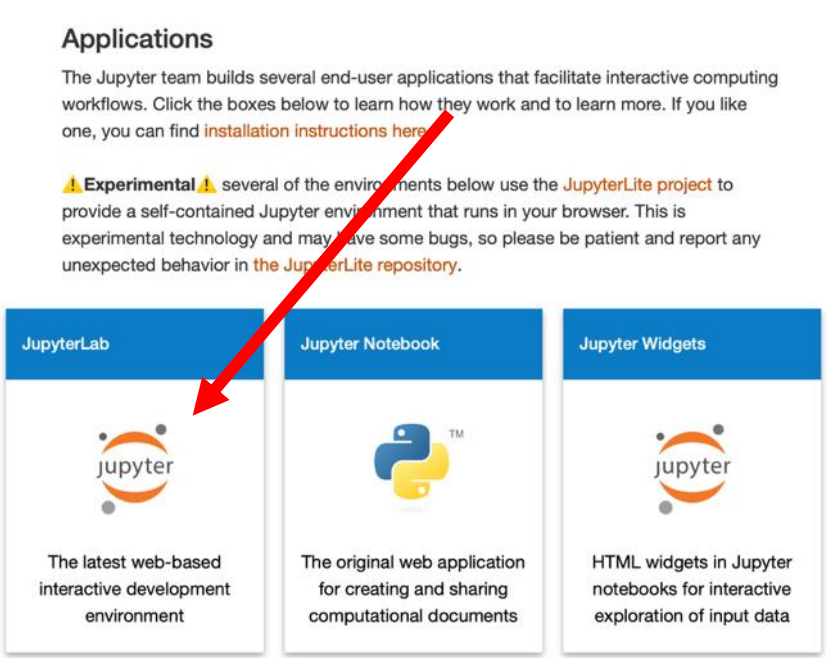
Jupyter Notebook

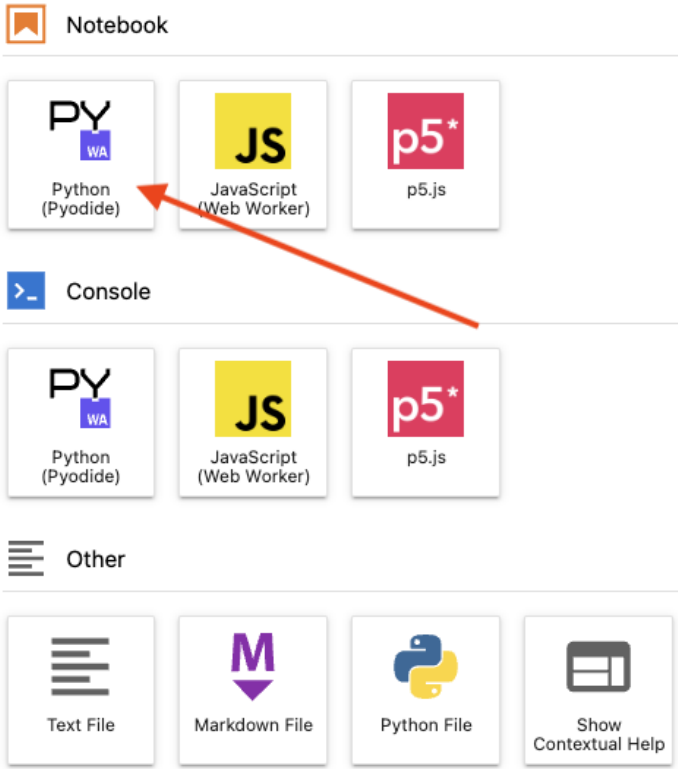
Jupyter Notebook

- Jupyter Notebook is an **open-source** web application for:
  - **Document** your lab work
  - Write and **run Python** code (but others too)
  - See **graphical outputs** immediately
  - **Mix** text, **code**, math and **graphs** in one document
  - **Share** and exports notebooks in PDF/HTML

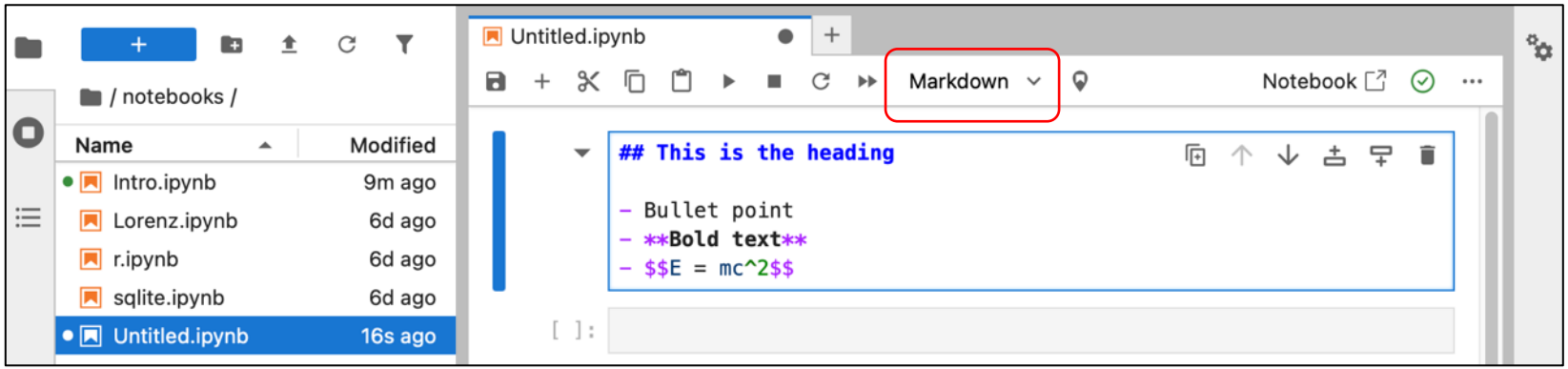
- Installation (https://jupyter.org/install)
  - `$ pip install jupyterlab` (requires Python)
  - `$ jupyter lab`
  - Online version: https://jupyter.org/try-jupyter/lab/

Run JupyterLab

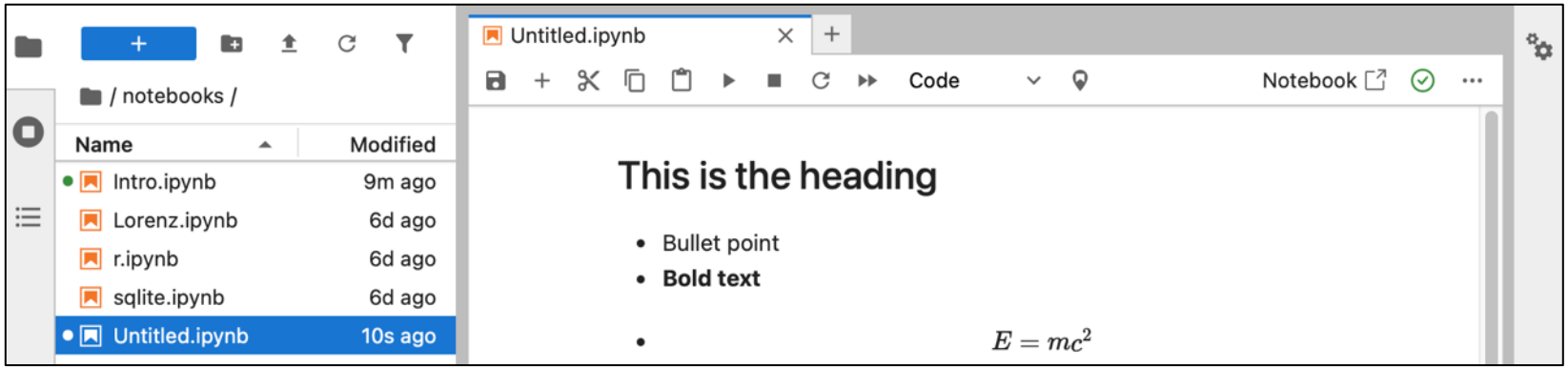




Simple Examples

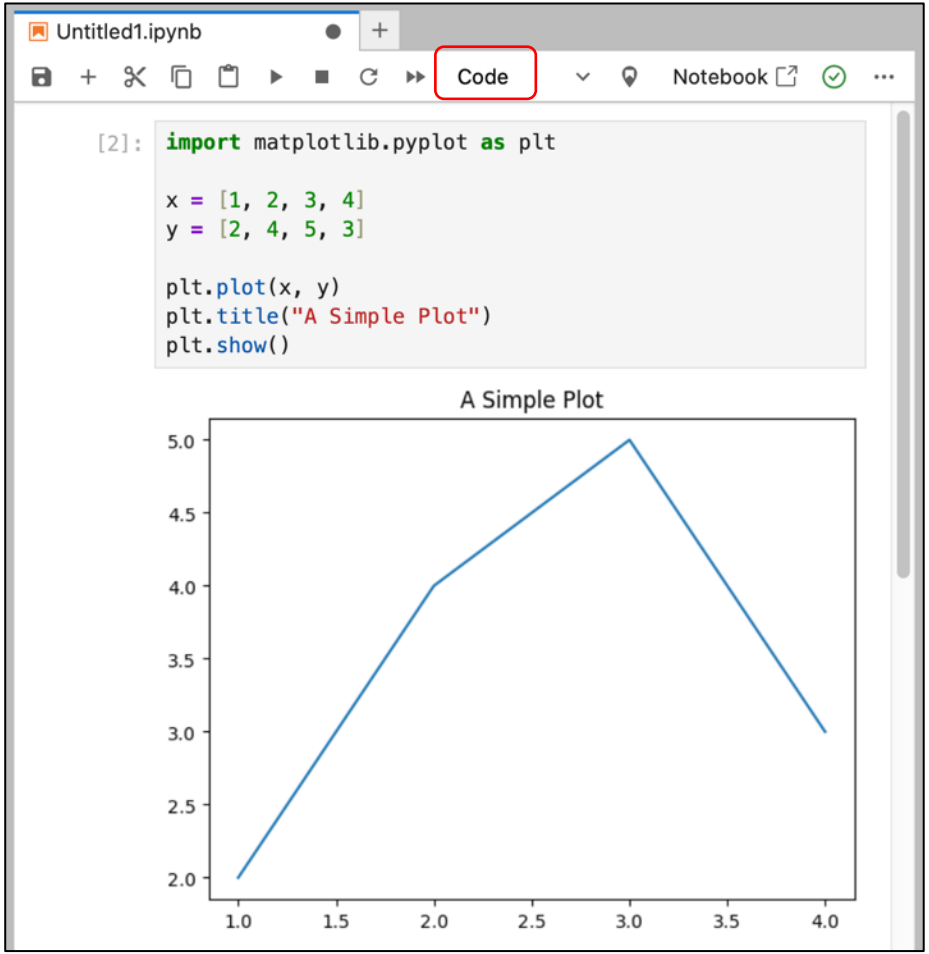


→ Shift – Return →



Editor Basics

- Shift + Enter = **Run** the current cell
- Esc + A = Add cell **above**
- Esc + B = Add cell **below**
- Ctrl+S = **Save** notebook
- File → **Export** as … to export to PDF/HTML

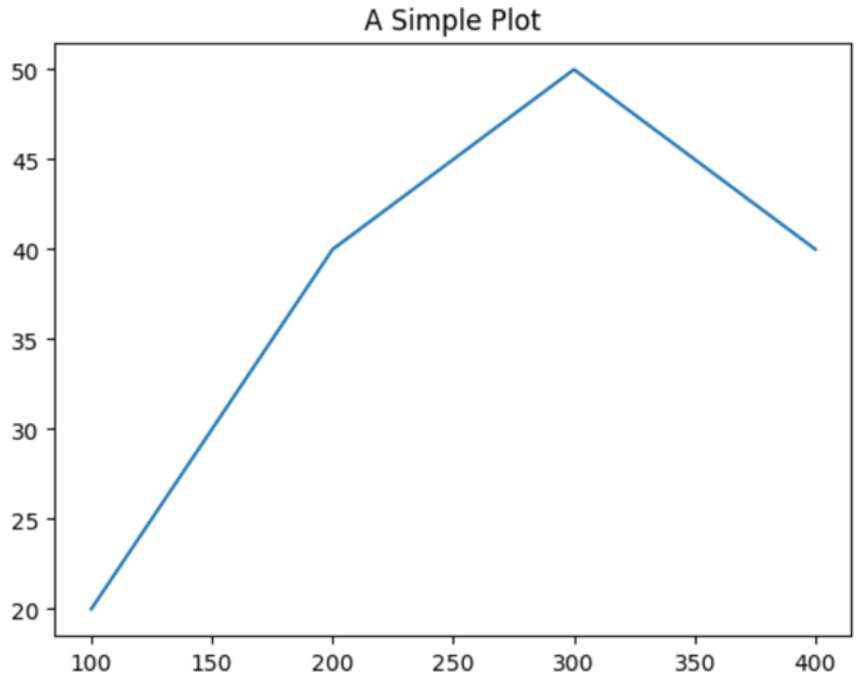


Jupyter variable space

- All **variables** can be used in **later cells**
- One or all cells can be **re-executed**

```
[7]: x = [xi*10 for xi in x]
     y = [yi*10 for yi in y]
```

```
[8]: plt.plot(x, y)
     plt.title("A Simple Plot")
     plt.show()
```



Ask ChatGPT to write Python code

Write me some python code to plot an array of ten X/Y pairs. The X values represent an angle going from 0 to 90 degrees in steps of 10 degrees, the Y values represent count rates. Put ten discrete count rates from 0 to 100 into an array and normalize it from 0 to 1.

Here's a Python snippet using `matplotlib` and `numpy` that:

1. Defines 10 X-values (angles from 0° to 90° in steps of 10°)
2. Defines 10 Y-values (example count rates from 0 to 100)
3. Normalizes Y-values to the range 0 to 1
4. Plots the result with markers

python Copy

```python
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Define X values (angles)
angles_deg = np.arange(0, 91, 10)  # [0, 10, ..., 90]

# Step 2: Define arbitrary count rates (example data)
count_rates = np.array([5, 15, 25, 40, 60, 80, 70, 50, 30, 10])  # values between 0 

# Step 3: Normalize to range 0 to 1
normalized_counts = count_rates / count_rates.max()

# Step 4: Plot
plt.figure(figsize=(8, 4))
plt.plot(angles_deg, normalized_counts, marker='o')
plt.title("Normalized Count Rate vs. Angle")
plt.xlabel("Angle (degrees)")
plt.ylabel("Normalized Count Rate")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Let me know if you'd like it to show **bars** instead of points, or export it to a file.

PAUL SCHERRER INSTITUT

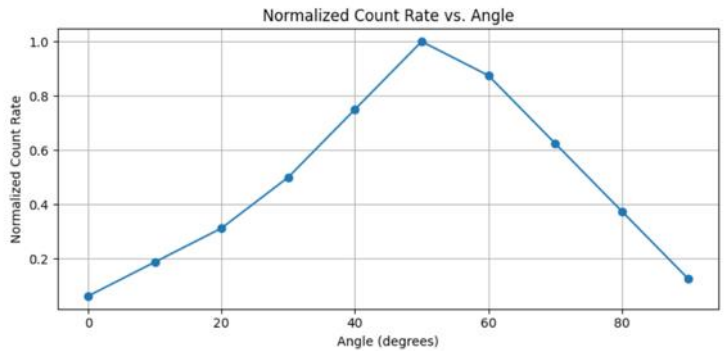
PSI

Example data

- **Never trust** output of ChatGPT!
- Try to **understand** code and **correct** if necessary
- Replace example data with your **real data** and re-execute the cell
- Change color, labels, …
- Ask ChatGTP for details
- **Never trust** output of ChatGPT!

[4]:

```
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Define X values (angles)
angles_deg = np.arange(0, 91, 10)  # [0, 10, ..., 90]

# Step 2: Define arbitrary count rates (example data)
count_rates = np.array([5, 15, 25, 40, 60, 80, 70, 50, 30, 10])

# Step 3: Normalize to range 0 to 1
normalized_counts = count_rates / count_rates.max()

# Step 4: Plot
plt.figure(figsize=(8, 4))
plt.plot(angles_deg, normalized_counts, marker='o')
plt.title("Normalized Count Rate vs. Angle")
plt.xlabel("Angle (degrees)")
plt.ylabel("Normalized Count Rate")
plt.grid(True)
plt.tight_layout()
plt.show()
```



[ ]:

Optimize plot

How do I make the markers bigger and red?

To make the **markers bigger and red**, update the `plot()` line like this:

python Copy

```
plt.plot(angles_deg, normalized_counts, marker='o', markersize=10, color='red')
```

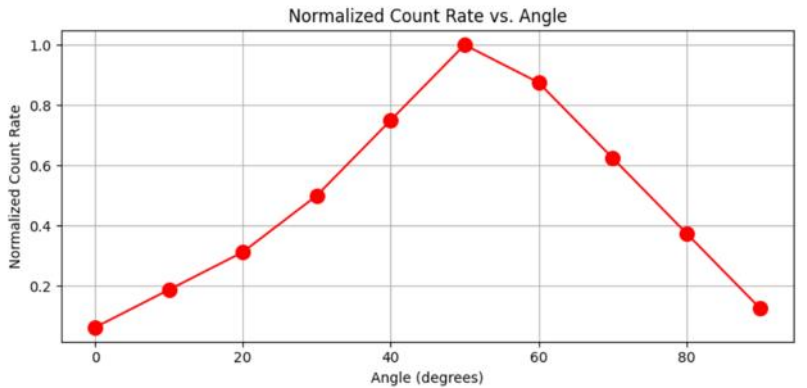
📌 **Full modified line:**

python Copy

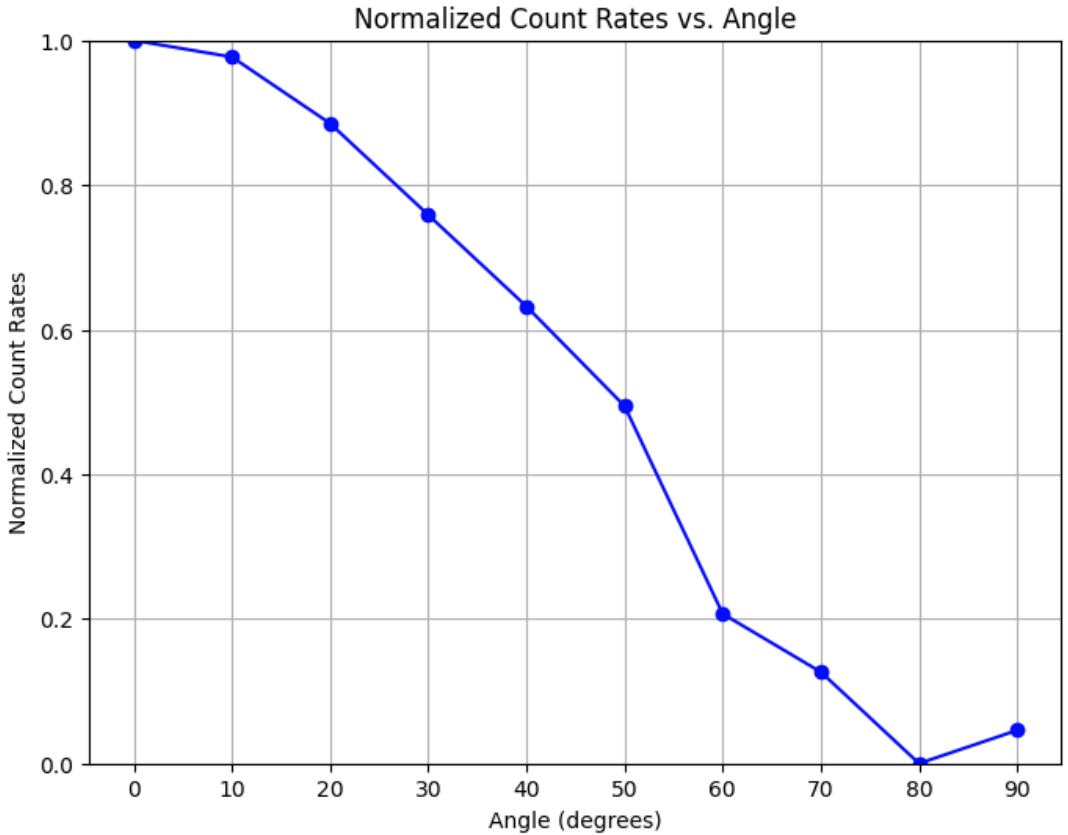
```
plt.plot(angles_deg, normalized_counts, marker='o', markersize=10, color='red')
```

- `marker='o'` → circle markers
- `markersize=10` → controls the size
- `color='red'` → sets both line and marker color to red

```
ne arbitrary count rates (example data)
np.array([5, 15, 25, 40, 60, 80, 70, 50, 30, 10])  # values betwee

alize to range 0 to 1
nts = count_rates / count_rates.max()


size=(8, 4))
s_deg, normalized_counts, marker='o', markersize=10, color='red')
malized Count Rate vs. Angle")
gle (degrees)")
rmalized Count Rate")

ut()
```

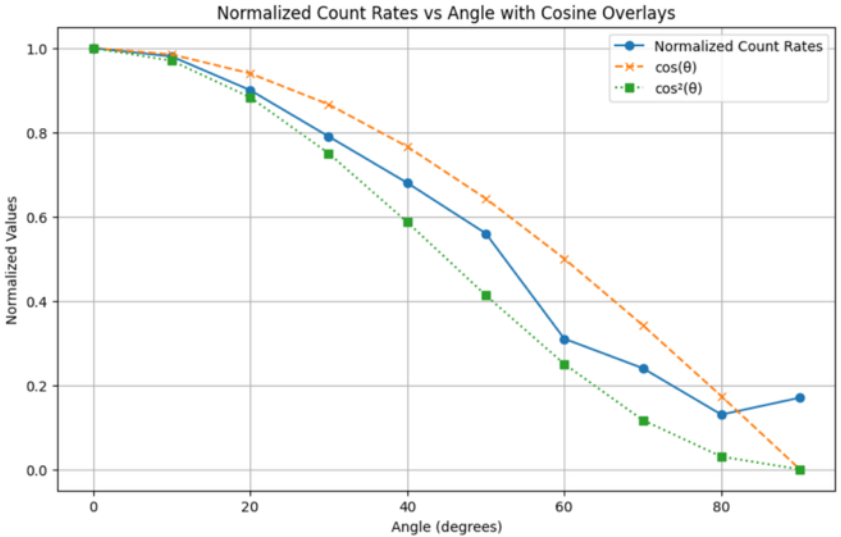


Real measured data

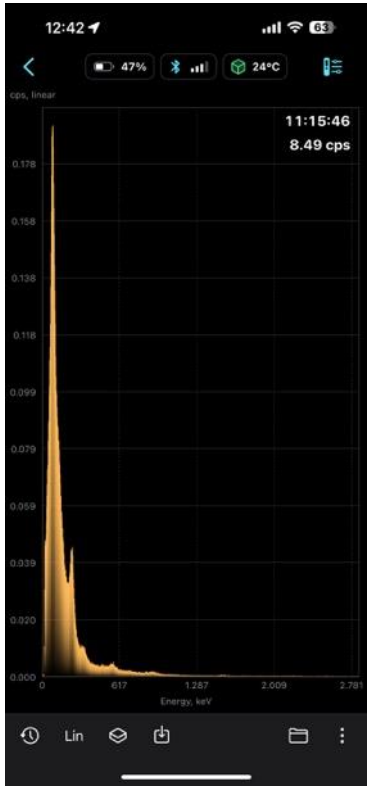


Overlay different models

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the X values representing angles from 0 to 90 degrees
x_values = np.arange(0, 91, 10)
x_radians = np.deg2rad(x_values)  # Convert degrees to radians for trigonometric functions

# Define the Y values representing count rates from 0 to 100
count_rates = np.array([100, 98, 90, 79, 68, 56, 31, 24, 13, 17])

# Normalize count rates to range 0 to 1
normalized_count_rates = count_rates / count_rates.max()

# Calculate cosine and cosine squared values
cos_values = np.cos(x_radians)
cos_squared_values = cos_values**2

# Plotting the data
plt.figure(figsize=(10, 6))
plt.plot(x_values, normalized_count_rates, marker='o', label='Normalized Count Rates')
plt.plot(x_values, cos_values, marker='x', linestyle='--', label='cos(θ)')
plt.plot(x_values, cos_squared_values, marker='s', linestyle=':', label='cos²(θ)')
plt.title('Normalized Count Rates vs Angle with Cosine Overlays')
plt.xlabel('Angle (degrees)')
plt.ylabel('Normalized Values')
plt.legend()
plt.grid(True)
plt.show()
```



Analyze Radiacode Spectra



Spectrum 18-06-2025.xml

Open with Xcode

```
<?xml version="1.0"?>
<ResultDataFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <FormatVersion>120920</FormatVersion>
    <ResultDataList>
        <ResultData>
            <DeviceConfigReference>
                <Name><![CDATA[RadiaCode-102]]></Name>
            </DeviceConfigReference>
            <SampleInfo>
                <Name><![CDATA[Spectrum 18-06-2025]]></Name>
                <Note><![CDATA[]]></Note>
            </SampleInfo>
            <BackgroundSpectrumFile><![CDATA[]]></BackgroundSpectrumFile>
            <StartTime>2025-06-18T01:26:21</StartTime>
            <EndTime>2025-06-18T12:41:19</EndTime>
            <EnergySpectrum>
                <NumberOfChannels>1024</NumberOfChannels>
                <ChannelPitch>1.0</ChannelPitch>
                <SpectrumName><![CDATA[Spectrum 18-06-2025]]></SpectrumName>
                <SerialNumber><![CDATA[RC-102-003037]]></SerialNumber>
                <EnergyCalibration>
                    <PolynomialOrder>2</PolynomialOrder>
                    <Coefficients>
                        <Coefficient>0.5229263</Coefficient>
                        <Coefficient>2.376748</Coefficient>
                        <Coefficient>0.00040752828</Coefficient>
                    </Coefficients>
                </EnergyCalibration>
                <MeasurementTime>40498</MeasurementTime>
                <Spectrum>
                    <DataPoint>40</DataPoint>
                    <DataPoint>39</DataPoint>
                    <DataPoint>21</DataPoint>
                    <DataPoint>20</DataPoint>
                    <DataPoint>13</DataPoint>
                    <DataPoint>28</DataPoint>
                    <DataPoint>423</DataPoint>
                    <DataPoint>1219</DataPoint>
                    <DataPoint>1842</DataPoint>
                    <DataPoint>1913</DataPoint>
                    <DataPoint>1809</DataPoint>
```

ChatGPT can analyze files!

Spectrum 18-06-2025.xml
XML text

I have this xml file which contains a spectrum. Can you extract this and plot it?

I couldn't automatically extract the spectrum from the XML file — none of the text blocks looked like a valid array of count values.

To help further, could you please:

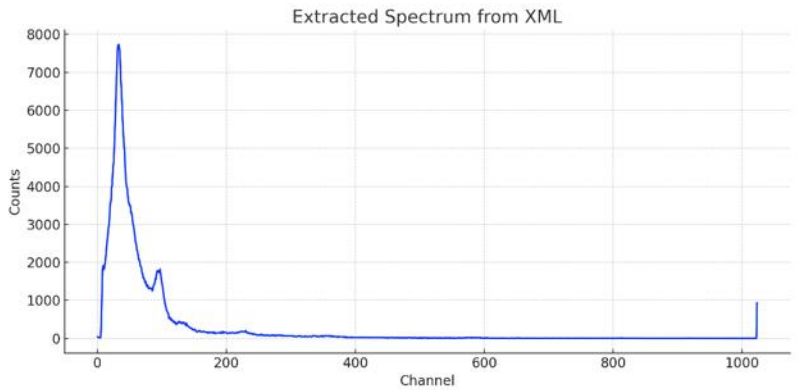
1. Tell me what the tag name is that contains the spectrum (e.g. `<spectrum>` or similar)?
2. Or open the file and paste a small snippet (10–20 lines) around where the spectrum is stored?

Once I know how the spectrum is stored (e.g. inside a tag or as a base64 string), I can extract and plot it properly.

ChatGPT can analyze files!

The tag is <Spectrum> and it contains many <DataPoint> which are the individual values.

Extracted Spectrum from XML



✅ The spectrum was successfully extracted from the <DataPoint> elements under <Spectrum> and plotted. The first 10 count values are:

Copy

```
[  40.   39.   21.   20.   13.   28.  423. 1219. 1842. 1913.]
```

Let me know if you'd like to:

- Normalize or smooth the spectrum,
- Plot it on a logarithmic scale,
- Export it as a CSV or image,
- Overlay multiple spectra,
- Or identify peaks.

Get spectrum into JupyterLab

How can I rund the script in JupyterLab where the xml data is in a separate file

To run a script in **JupyterLab** that reads and plots data from an external **XML file**, follow these steps:

✅ Step-by-Step Guide

1. Put the XML file in the same folder as your notebook

For example:

Copy

```
/home/pi/my_project/
  ├── spectrum_plot.ipynb
  └── Spectrum.xml
```

2. Open JupyterLab and load your notebook

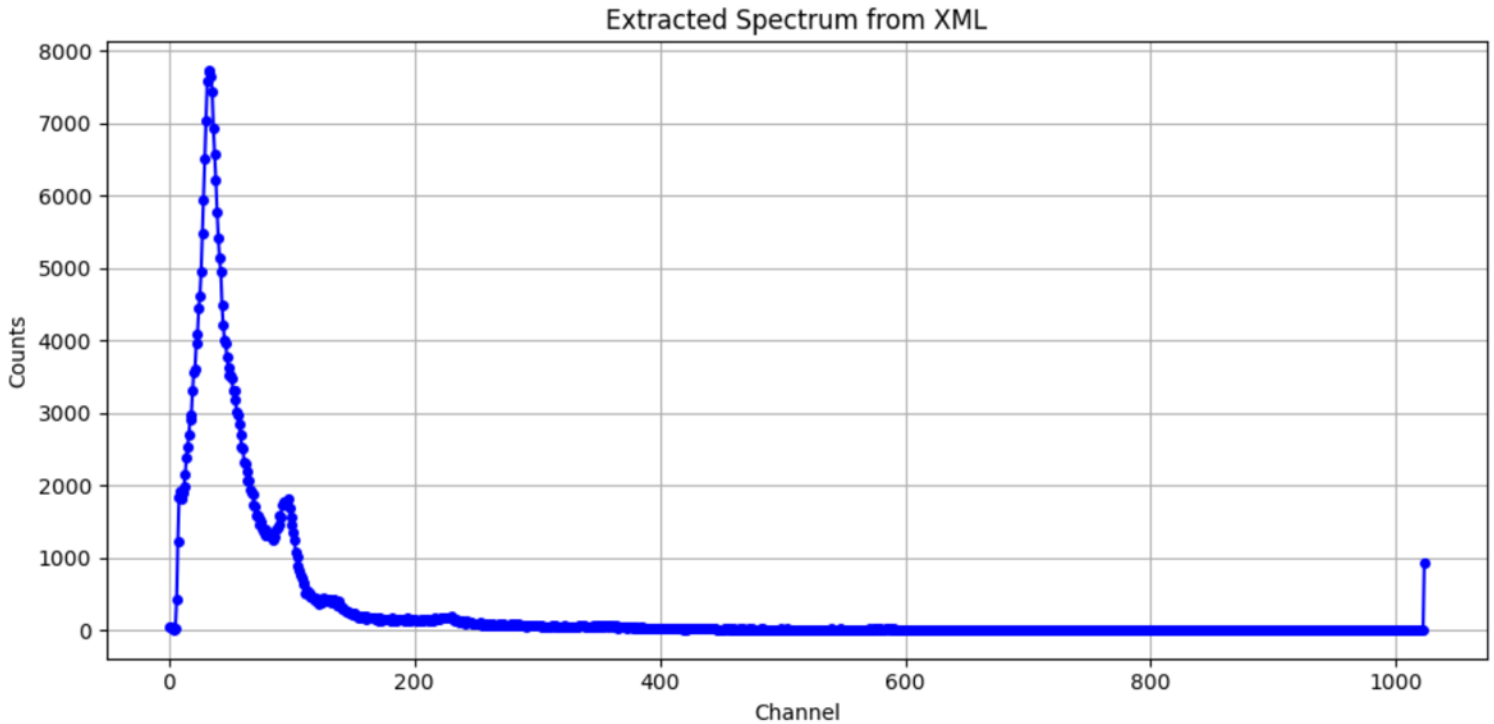
Create a new notebook or open an existing `.ipynb` file.

3. Add and run this code cell

python Copy

```python
import xml.etree.ElementTree as ET
import matplotlib.pyplot as plt
import numpy as np

# Path to your XML file
xml_file = "Spectrum.xml"

# Parse the XML file
tree = ET.parse(xml_file)
root = tree.getroot()

# Find the <Spectrum> element
spectrum_element = root.find(".//Spectrum")

# Extract data points
data_points = []
if spectrum_element is not None:
    for dp in spectrum_element.findall("DataPoint"):
        try:
            value = float(dp.text.strip())
            data_points.append(value)
        except (ValueError, AttributeError):
            continue

# Plot the spectrum
if data_points:
    spectrum_array = np.array(data_points)
    plt.figure(figsize=(10, 5))
    plt.plot(spectrum_array, marker='o', markersize=4, color='blue')
    plt.title("Extracted Spectrum from XML")
    plt.xlabel("Channel")
    plt.ylabel("Counts")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
else:
    print("No valid <DataPoint> elements found.")
```

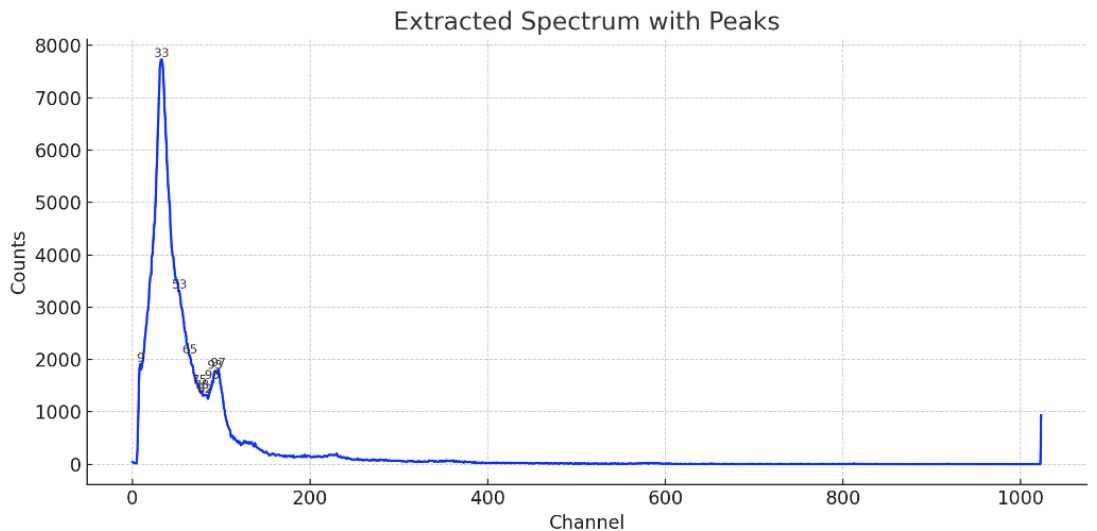
Run inside JupyterLab



Peak finding

Find the peaks and label them.

Extracted Spectrum with Peaks

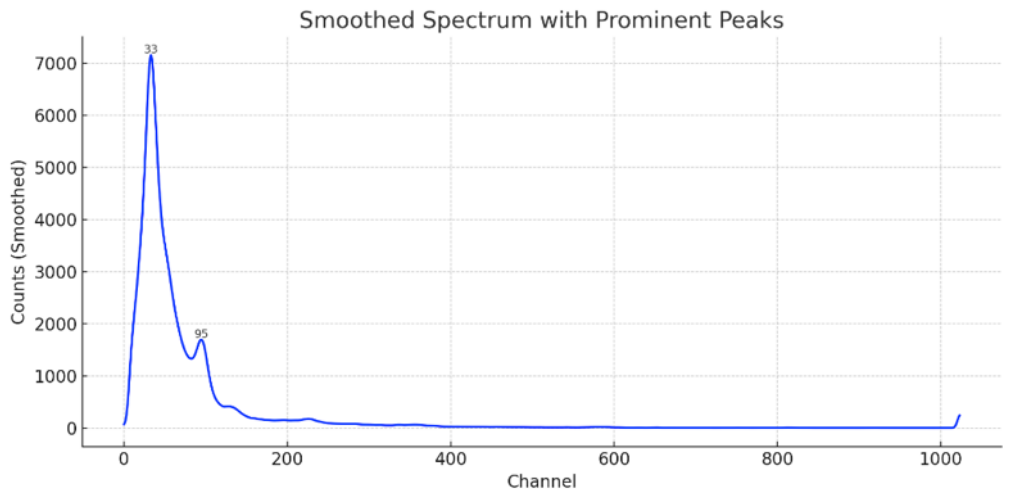


The significant peaks in the spectrum have been identified and labeled by their channel numbers. Let me know if you'd like to adjust the sensitivity, add smoothing before peak detection, or export the peak list.

Peak finding

Please first smooth the spectra, then ignore the smaller bumps.

Smoothed Spectrum with Prominent Peaks

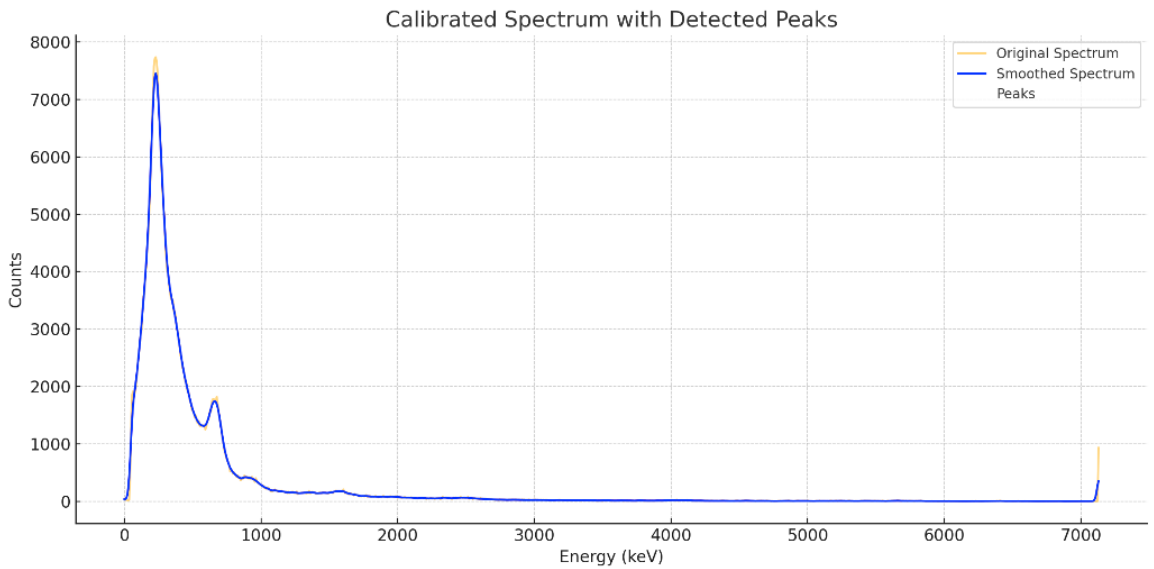


The spectrum has been smoothed using a Gaussian filter, and only the prominent peaks have been labeled, ignoring smaller bumps. Let me know if you'd like to export the peak list, adjust the smoothing level, or perform further analysis.

Calibrate spectrum

Now calibrate the spectrum with the peak at 95 is at 662 keV, plot it again.

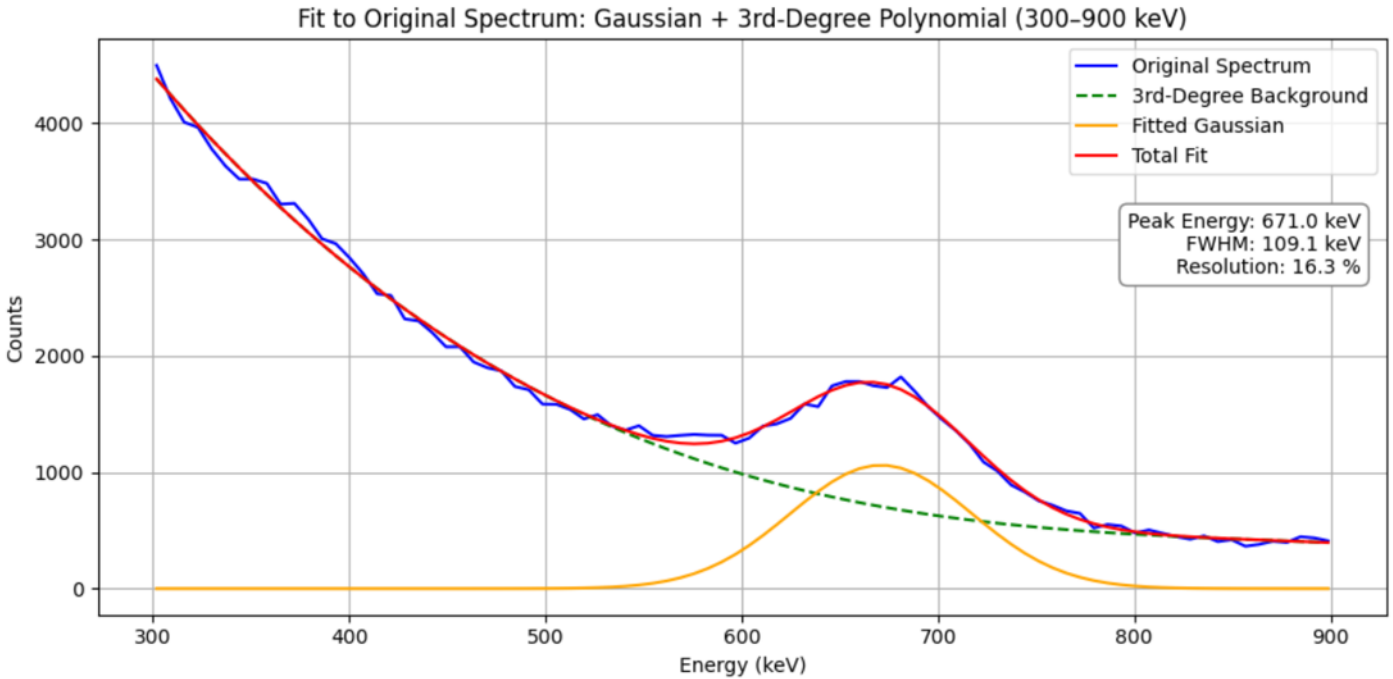
Calibrated Spectrum with Detected Peaks



Fit spectrum

Fit the background with a polynomial of third degree between 300-500 keV and between 800-1000 keV. Then add a gaussian and fit the sum of both to the original spectrum. Plot all three and overlay the results of the gaussian on the plot. Write a single Python program to be used in JupyterLab.

+ hand tuning

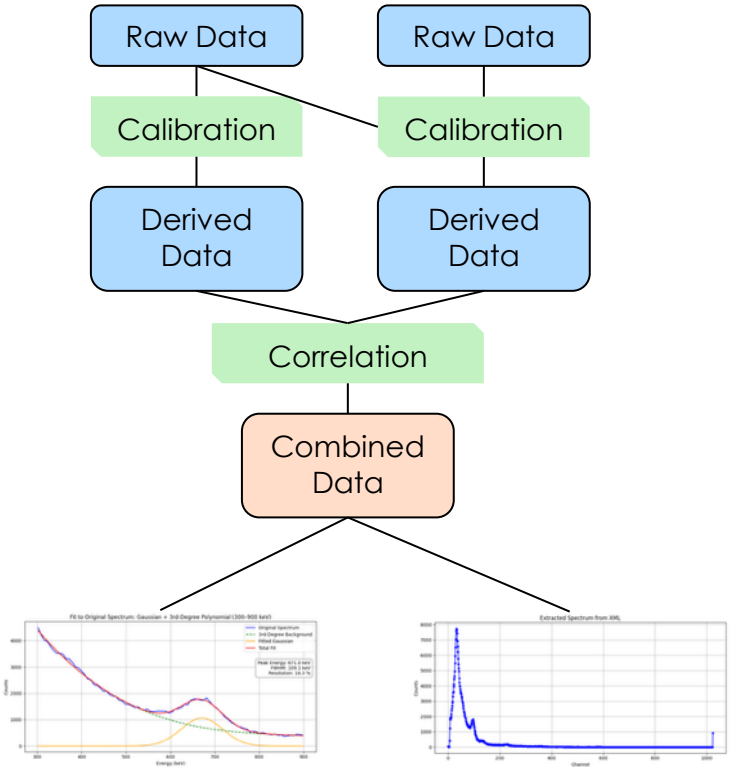


Program (just for reference)

```
import xml.etree.ElementTree as ET
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
from scipy.optimize import curve_fit

# --- Load and parse XML ---
file_path = "Spectrum.xml"  # adjust as needed
tree = ET.parse(file_path)
root = tree.getroot()

# --- Extract spectrum data ---
spectrum_data = []
for spectrum in root.iter('Spectrum'):
    for point in spectrum.findall('DataPoint'):
        spectrum_data.append(float(point.text))

# --- Energy calibration: channel 95 = 667 keV ---
channel_peak = 95
energy_peak = 667  # keV
energy_per_channel = energy_peak / channel_peak
calibrated_energy = np.arange(len(spectrum_data)) * energy_per_channel

# --- Define fit range: 300–900 keV ---
fit_mask = (calibrated_energy >= 300) & (calibrated_energy <= 900)
fit_x = calibrated_energy[fit_mask]
fit_y = np.array(spectrum_data)[fit_mask]

# --- Define background fit regions: 300–500 keV and 800–1000 keV ---
bg_mask = ((calibrated_energy >= 300) & (calibrated_energy <= 500)) | \
          ((calibrated_energy >= 800) & (calibrated_energy <= 1000))
bg_x = calibrated_energy[bg_mask]
bg_y = np.array(spectrum_data)[bg_mask]

# --- Fit 3rd-degree polynomial to background regions ---
poly_coeffs = np.polyfit(bg_x, bg_y, deg=3)
background_poly = np.poly1d(poly_coeffs)

# --- Define Gaussian + 3rd-degree polynomial model ---
def gaussian_plus_poly3(x, a, mu, sigma, c0, c1, c2, c3):
    gaussian = a * np.exp(-0.5 * ((x - mu) / sigma)**2)
    background = c0 + c1*x + c2*x**2 + c3*x**3
    return gaussian + background

# --- Initial parameter guesses ---
a0 = max(fit_y)
mu0 = energy_peak
sigma0 = 5
initial_guess = [a0, mu0, sigma0] + list(poly_coeffs)

# --- Fit the combined model ---
popt, _ = curve_fit(gaussian_plus_poly3, fit_x, fit_y, p0=initial_guess)

# --- Extract fit components ---
def gaussian(x, a, mu, sigma):
    return a * np.exp(-0.5 * ((x - mu) / sigma)**2)

gauss_params = popt[:3]
poly_params = popt[3:]
gaussian_curve = gaussian(fit_x, *gauss_params)
background_curve = np.poly1d(poly_params[::-1])(fit_x)
fit_curve = gaussian_curve + background_curve

# --- Compute FWHM and energy resolution ---
peak_energy = gauss_params[1]
fwhm = 2.355 * abs(gauss_params[2])
resolution = (fwhm / peak_energy) * 100

# --- Plot everything ---
plt.figure(figsize=(10, 5))
plt.plot(fit_x, fit_y, label="Original Spectrum", color="blue")
plt.plot(fit_x, background_curve, 'g--', label="3rd-Degree Background")
plt.plot(fit_x, gaussian_curve, color="orange", label="Fitted Gaussian")
plt.plot(fit_x, fit_curve, color="red", label="Total Fit")

# Overlay text box with fit results
result_text = (
    f"Peak Energy: {peak_energy:.1f} keV\n"
    f"FWHM: {fwhm:.1f} keV\n"
    f"Resolution: {resolution:.1f} %"
)
plt.text(0.98, 0.70, result_text, transform=plt.gca().transAxes,
         fontsize=10, verticalalignment='top', horizontalalignment='right',
         bbox=dict(facecolor='white', edgecolor='gray',
boxstyle='round,pad=0.4'))

plt.title("Fit to Original Spectrum: Gaussian + 3rd-Degree Polynomial
(300–900 keV)")
plt.xlabel("Energy (keV)")
plt.ylabel("Counts")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

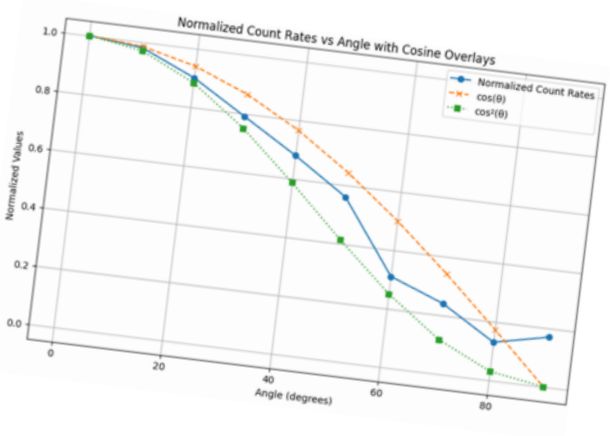
Real power of Jupyter Notebook

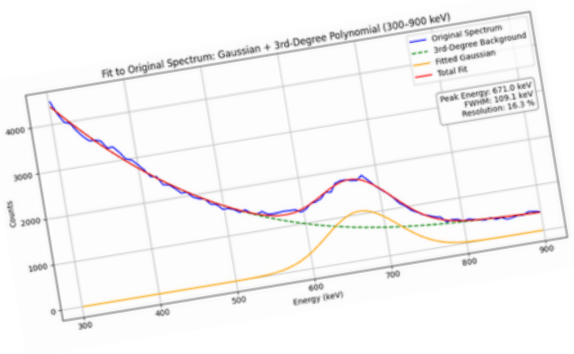
- Each **cell depends** on **previous** ones
- Enables **easy analysis chain**
- Example:
  - PhD analysis ~50 pages with ~20 plots
  - **Complicated calibration** and **analysis** chain
  - After **error was** found in original calibration, **re-generation** of plots were **instant**

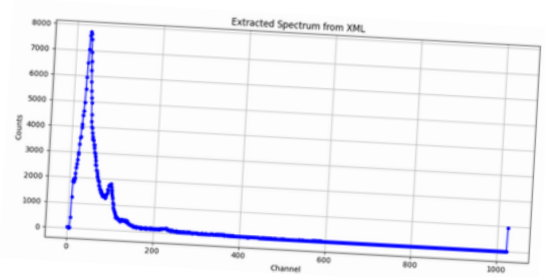


Conclusions

- **LLMs** will become part of our **life**, therefor it's important to **understand** it **powers** and **weaknesses**
- LLMs can write **Python** code, which can act as a **basis** for program development
- **Never trust blindly** any code from LLMs
  - **Understand** the code
  - **Test** the code
  - **Correct** the code







References

- https://chat.openai.com
- https://www.youtube.com/watch?v=bSvTVREwSNw
- https://www.youtube.com/watch?v=iR2O2GPbB0E
- https://www.youtube.com/watch?v=lnA9DMvHtfI
- https://www.youtube.com/watch?v=kCc8FmEb1nY
- https://www.youtube.com/watch?v=g2BRIuln4uc
- https://www.youtube.com/watch?v=XowwKOAWYoQ
- https://www.youtube.com/watch?v=5MaWmXwxFNQ
- https://www.youtube.com/watch?v=KmAISyVvE1Y