

# Python Numeric Types

*DS 5110: Big Data Systems*

*Spring 2025*

Lecture 5

Zhaoyuan Su, Yue Cheng



Some material taken/derived from:

• Wisconsin CS 544 by Tyler Caraza-Harter.

@ 2025 released for use under a [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.

# DS<sup>2</sup> Research Lab

(Data Systems for Data Science)

Director: Prof. Yue Cheng

Homepage: <https://ds2-lab.github.io/>

## Ph.D. Students

Yuqi Fu



- 5th Year Ph.D. Student in Computer Science
- **Research Interests:** Operating systems
- [fishercht1995.github.io](https://github.com/fishercht1995)

Ben Carver



- 4th Year Ph.D. Student in Computer Science
- **Research Interests:** Serverless parallel computing
- [scusemua.github.io/](https://github.com/scusemua)

Zhaoyuan (Alex) Su



- 4th Year Ph.D. Student in Computer Science
- **Research Interests:** Data reduction, Sys4ML, serverless computing
- [alexssu.github.io/](https://github.com/alexssu)

Rui Yang



- 4th Year Ph.D. Student in Computer Science
- **Research Interests:** Storage systems
- [mason.gmu.edu/~ryang22/](https://mason.gmu.edu/~ryang22/)

Zirui Wang



- 1st Year Ph.D. Student in Computer Science
- **Research Interests:** ML systems
- [https://jerryw35.github.io/](https://github.com/jerryw35)

Tingfeng Lan



- 1st Year Ph.D. Student in Computer Science
- **Research Interests:** ML systems
- [https://antlera.github.io/](https://github.com/antlera)

# Announcement

- A1 is out
  - Due in two weeks, on Feb 13.

# Is $0.1 + 0.2 = 0.3$ in Python?

```
zsu — python3 — python3 — Python — 70x9
→ ~ python3
Python 3.12.3 (main, Apr 9 2024, 16:03:47) [Clang 14.0.0 (clang-1400.0.29.202)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print(0.1 + 0.2 == 0.3)
False
>>> print(0.1 + 0.2)
0.30000000000000004
>>> █
```

# Learning objectives

- Know how machine stores numeric types, especially **floating points**
- Compare different numeric types in terms of **memory space cost, range, and precision**

# Python numeric types (built in)

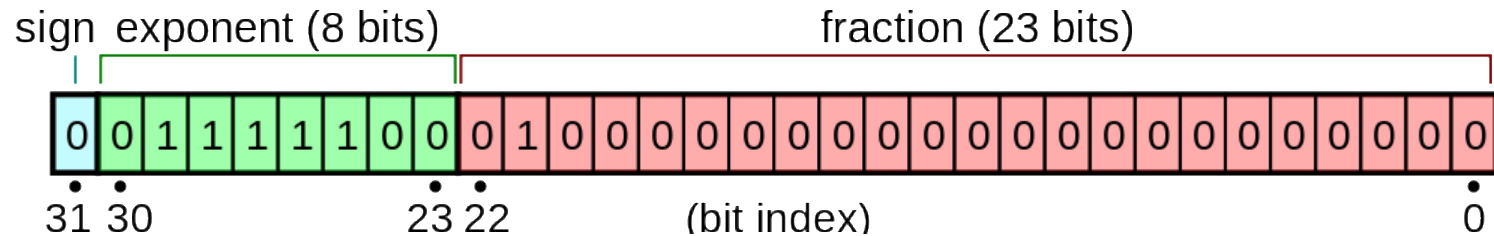
Official lib: <https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex>

## Python numeric types

- int
  - No max/min size (Python is unusual in this way)
  - Bigger values -> more bits necessary
- float
  - Defaults 64 bits (double precision)
    - You can also use float32 given a certain framework (e.g., PyTorch, numpy, etc.)
  - Most pre-trained ML models use float32 for parameters

# float32

- Standard IEEE format (float32)



$$\begin{aligned} p &= (-1)^s \times 2^{e-127} \times (1.m_1m_2\dots m_{23})_2 \\ &= (-1)^s \times 2^{e-127} \times \left(1 + \sum_{i=1}^{23} m_i \times 2^{-i}\right) \end{aligned}$$





# Python numeric types (built in)

Official lib: <https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex>

## Python numeric types

- int
  - No max/min size (Python is unusual in this way)
  - Bigger values -> more bits necessary
- float
  - Defaults 64 bits (double precision)
    - You can also use float32 given a certain framework (e.g., PyTorch, numpy, etc.)
  - Most pre-trained ML models use float32 for parameters
  - Min/max, Inf, -Inf, NaN have special bit combinations

# Python numeric types (built in)

Official lib: <https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex>

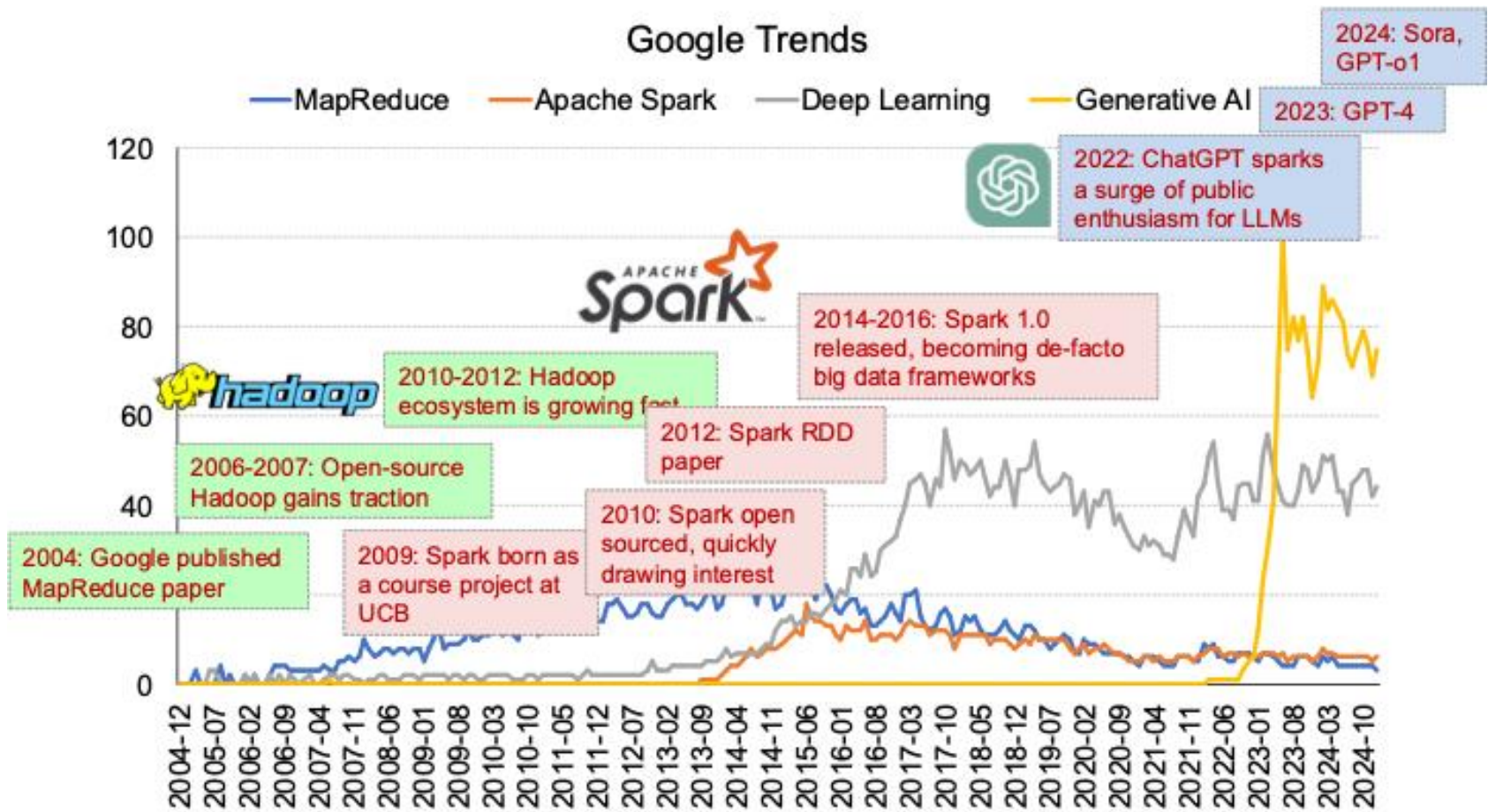
## Python numeric types

- int
  - No max/min size (Python is unusual in this way)
  - Bigger values -> more bits necessary
- float
  - Defaults 64 bits (double precision)
    - You can also use float32 given a certain framework (e.g., PyTorch, numpy, etc.)
  - Most pre-trained ML models use float32 for parameters
  - Min/max, Inf, -Inf, NaN have special bit combinations
- complex,
  - e.g. `complex('-1.23+4.5j')`

# Other (commonly used) numeric types

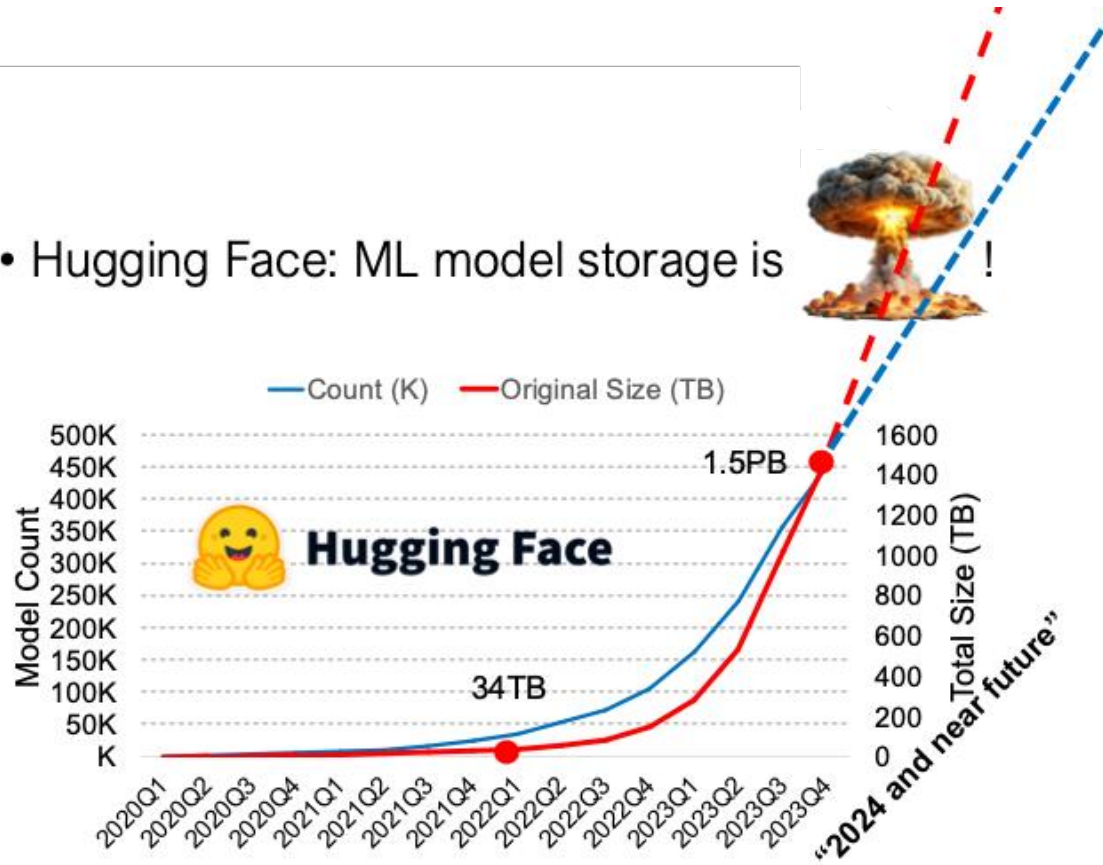
- Common numeric types that (a) CPU can directly manipulate and (b) popular Python frameworks (e.g., PyTorch) support
  - ints: uint8, int8, int16, int32, int64
  - floats: float16, float32, float64
  - dtype (data type)

# A brief history about Big Data



# Data explosion in the GenAI era

- Hugging Face: ML model storage is



**HuggingFace's AI/ML models are growing exponentially!**

# Floating points in Large Language Models

- Modern deep learning (DL) relies on floating point computations.
- Large-scale models (e.g., GPT, LLaMA) require **memory-efficient** representations.
- Lower **precision** formats (FP16, BF16, FP8) improve **speed & efficiency**.
- Trade-off: **Accuracy VS. Speed VS. Memory.**

# Floating points in Large Language Models

- FP32 (32-bit)- standard precision in LLMs
  - 8-bit exponent, 23-bit mantissa. High precision but high memory cost. Used in early ML models.
- FP16 - Faster Computation, Less Memory
  - 5-bit exponent, 10-bit mantissa. Less precise but faster on GPUs. Efficiently used in NVIDIA Tensor Cores.
- BF16 - Balancing Range & Precision
  - 8-bit exponent, 7-bit mantissa. Same exponent as FP32 but a lower mantissa. Used in Google TPUs.
- FP8 - The Future of Efficient Inference
  - Emerging format (E5M2, E4M3). Reduces model size 4x of FP32. Lower precision but good enough for inference.

# Floating points in Large Language Models

Format	Bits	Exponent	Mantissa	Memory	Use Case
<b>FP32</b>	32	8 bits	23 bits	4 bytes	Standard training
<b>FP16</b>	16	5 bits	10 bits	2 bytes	Faster training
<b>BF16</b>	16	8 bits	7 bits	2 bytes	Training stability
<b>FP8</b>	8	E5M2 or E4M3	2-3 bits	1 byte	Efficient inference



# Demos ...