Foundation models
for spatial-time series

Week 2

Classic versus Vibe coding

1. Your project to develop
2. Computational experiment setup
3. Data download and preprocessing
4. Your model
5. Alternative models
6. Computational experiment
7. Manual reporting

1. Your project among alternatives
2. Prompt engineering
3. Library datasets
4. Your ***generated*** model
5. Generated superposition of models
6. Generated model comparison
7. Generated report and paper

The Vibe Coding Technique

1. Start with curiosity, not software architecture

Instead of planning a whole system, just pick a question

The vibe is to play, not to ship production code

2. Keep it lightweight

A Jupyter notebook, Google Colab, or a single Python file

Only import the minimum you need

3. Prototype in the smallest way possible

Instead of building a whole dataset loader,
hardcode a tiny dataset

4. Write tiny models first

Instead of aiming for GPT-4, write a baby transformer

5. Iterate by feel

Once something works, ask “what if I add…?”

Add self-attention. Add positional encodings.
Scale up dataset a bit.

Each step should be incremental and driven by curiosity.

6. Embrace imperfection

Code will be messy. That’s okay — vibe coding is about insight first, refactor later.

When the experiment feels solid, *then* clean it up.

Motivation

As we write papers,

we make a lots of code to compare many similar models.

The Vibe coding style

eases the process of model collection, creation, and comparison.

Minimum problem statement

There given as input

1) a spatial time series and its context (physics of measurements)
2) a mathematical, computational or machine learning model (prior)
3) a prompt

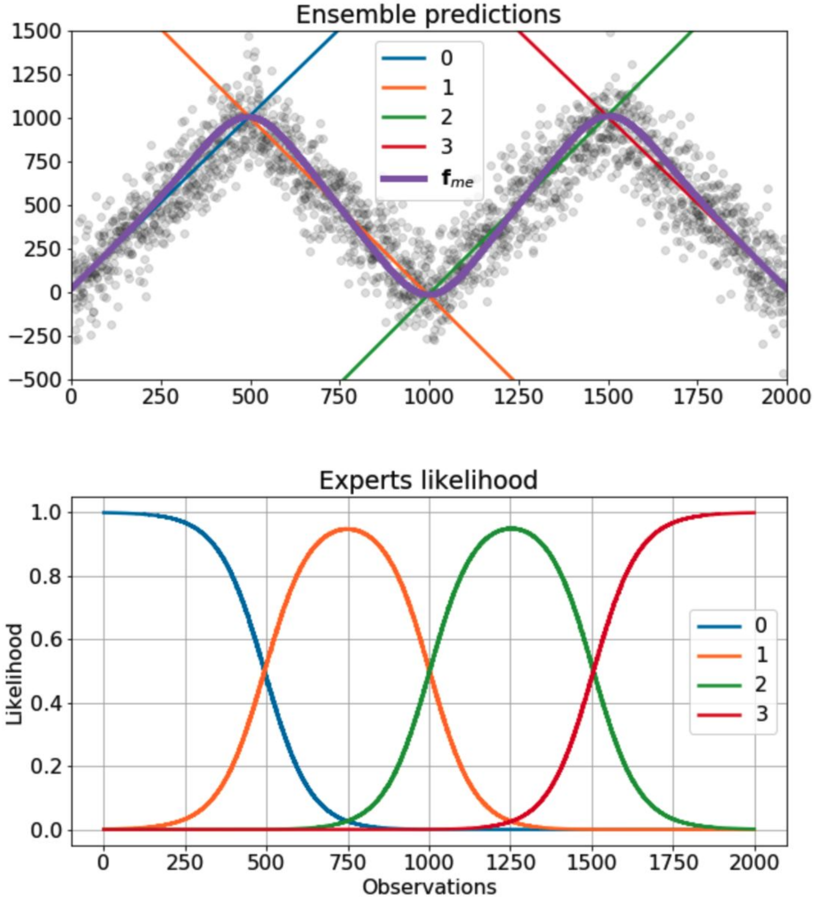
Expected as output

1) a selected model that fits data
2) an analytical report to the prompt

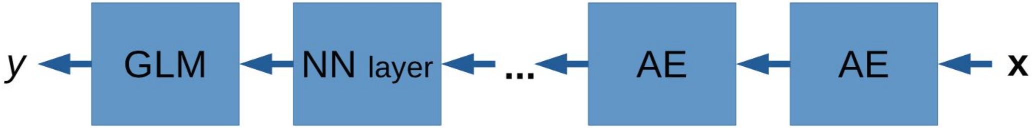
Prompt examples

1. Find change points in the [time series]
2. Select a model from [class] for [time series] given [mathematical model]

PINN is the Hinton's distillation of models



Creating composite loss function



$$f = \underset{1\times 1_k}{\boldsymbol{w}^\top} \boldsymbol{z}_{k-1} \circ \boldsymbol{W}_{k-1}^\top \boldsymbol{z}_{k-2} \circ \cdots \circ \underset{n_2\times 1}{\boldsymbol{W}_2^\top \boldsymbol{z}_1} \circ \underset{n_1\times n}{\boldsymbol{W}_1^\top} \underset{n\times 1}{\boldsymbol{x}}$$

Neural network error

$$E_y = \left(y_i - f(\boldsymbol{x})\right)^2$$

Autoencoder reconstruction error

$$E_{\boldsymbol{x}} = \|\boldsymbol{x} - \boldsymbol{r}(\boldsymbol{z})\|^2$$

Types of autoencoders

PCA	skip block	metric	multi-linear
$\boldsymbol{W}^\top \boldsymbol{W} = \boldsymbol{I}_n$	$\boldsymbol{W} = \boldsymbol{I}_n$	$\boldsymbol{x}^\top \boldsymbol{W} \boldsymbol{x} \geqslant 0$	$\underline{\boldsymbol{W}}\boldsymbol{X}$

Autoencoder transform: $\boldsymbol{z} = \left(1 + \exp(-\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})\right)^{-1}$

Reconstruction decoder: $\hat{\boldsymbol{x}} = \boldsymbol{r}\left(\boldsymbol{z}(\boldsymbol{x})\right)$

loss $\rightarrow$ model $\rightarrow$ parameters $\rightarrow$ optimizer

To construct a tool, one has to generate the composite loss function

```
model = TinyGPT(vocab_size)
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3)

for step in range(200):  # try 2000+ for better results
    xb, yb = get_batch()
    logits, loss = model(xb, yb)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if step % 50 == 0:
        print(step, loss.item())
```

Knowledge distillation between physics model and NN

For some objects $\mathbf{x}$ there given privileged information $\mathbf{x}^*$. Introduce a student model $\mathbf{f}_\mathrm{s} \in \mathfrak{F}_\mathrm{s}$ and a teacher model $\mathbf{f}_\mathrm{t} \in \mathfrak{F}_\mathrm{t}$:

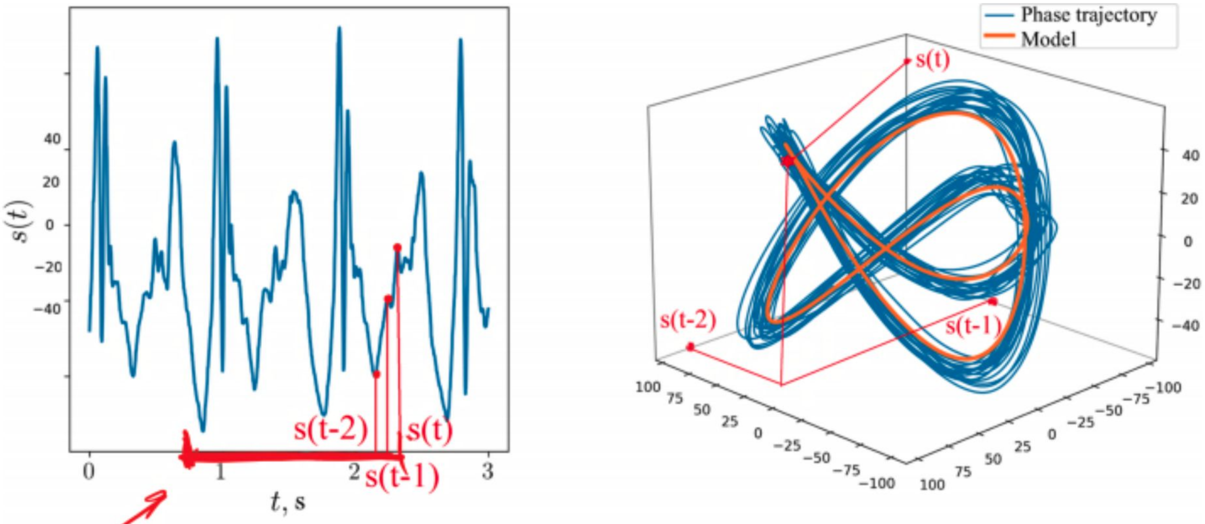
$$\mathbf{f}_\mathrm{s} : \mathbf{x} \longrightarrow \mathbf{y}, \quad \mathbf{f}_\mathrm{t} : \mathbf{x}^* \longrightarrow \mathbf{y}.$$

$$\mathbf{f}_\mathrm{s} = \arg\min_{\mathbf{f} \in \mathfrak{F}_s} \frac{1}{n} \sum_{i=1}^{n} \Big[(1-\lambda) S\big(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i)\big) + \lambda S\big(\mathbf{s}_i, \mathbf{f}(\mathbf{x}_i)\big)\Big],$$

T is the temperature, parameter of smoothing, for classification:

$$\mathbf{s}_i = \boldsymbol{\sigma}(\mathbf{f}_\mathrm{t}(\mathbf{x}_i)/T), \; S\big(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i)\big) = -\sum_{k=1}^{c} \mathbf{y}_k \log \boldsymbol{\sigma}\big(\mathbf{f}(\mathbf{x}_i)\big), \; \boldsymbol{\sigma} - \text{softmax}$$

Time series and the state space embedding



$t - 1000$

$\dim(s) \approx 1000 \longrightarrow \dim(x) = 4$

Reduce dimensionality with the principal component analysis, autoencoder $\boldsymbol{z} = \boldsymbol{W}^\mathsf{T}\boldsymbol{x}$ where $\boldsymbol{W}$ is an orthogonal (rotation) matrix. The first principal components are given by Singular Values Decomposition

$$\sqrt{\lambda_k}\boldsymbol{V_k} = \boldsymbol{X}^\mathsf{T}\boldsymbol{U}_k \quad \text{the SVD is} \quad \boldsymbol{X} = \boldsymbol{U}\Lambda\boldsymbol{V}^\mathsf{T}$$

Forecasting problem and model selection

trajectory

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\underset{n\times d}{\mathbf{X}} \arrow[d, bend right, "\varphi_{\mathrm{enc}}"'] \arrow[r, "\mathrm{BCI}"'] & \underset{n\times s}{\mathbf{Y}} \arrow[d, bend left, "\psi_{\mathrm{enc}}"] \\
\underset{n\times K}{\mathbf{T}} \arrow[u, "\varphi_{\mathrm{dec}}"'] \arrow[r, leftrightarrow, "\mathcal{L}_{\mathrm{cons}}"] & \underset{n\times K}{\mathbf{U}} \arrow[u, "\psi_{\mathrm{dec}}"]
\end{tikzcd}
$$

dim reduction

alignment

$x \to x$ **transformations**

1. Linear model
2. Stack of autoencoders
3. Neural ODE
4. Neural PDE
5. Graph diffusion

$y \to y$ **transformations**

1. Linear model
2. Spherical harmonics
3. Topological alignment
4. Lagrangian, Hamiltonian neural networks

Initial models

1. **Models**

    Direct models: AR, ARIMA, GRU, LSTM

    Metric models: LLE, DM, GH, RBF

    Non-parametric: GPR

2. **Ways to construct state spaces**

    SSM models: S4, S5, Hippo

    Kalman

    SSA, SSM

3. **Ways to transform state spaces**

    FT, OL, ODE

    CCA, CCM

The time series has two domains: the time domain and the frequency domain. The spatial time series also has a metric space or metric tensor that changes in time.

Or any model from Prophet: https://github.com/facebook/prophet

For the Week 3

1. Join a group
2. Discuss the goals of the project and a solution (see the problem statement above)
3. Make a review of various ways to solve the problem
4. Select an LLM, GPT
5. Run the code to check if it works
    1. Store the code in the group repository
    2. Store the talk slides/report, too
6. Make a 10-minute talk about
    1. Functionality and architecture/principles of the model
    2. Why did you select this model
    3. The alternative models to select from

The challenge

How to generate the composite loss function?

How to manage hyperparameters (components of the loss function)?

Do we need to treat the tokens (what LLM forecasts) as hyperparameters?

How to select a class of models according to request?

How to select a model from a class?

    __init__()

Do we need to use tool-embedding class of LLM or there are the other solutions?

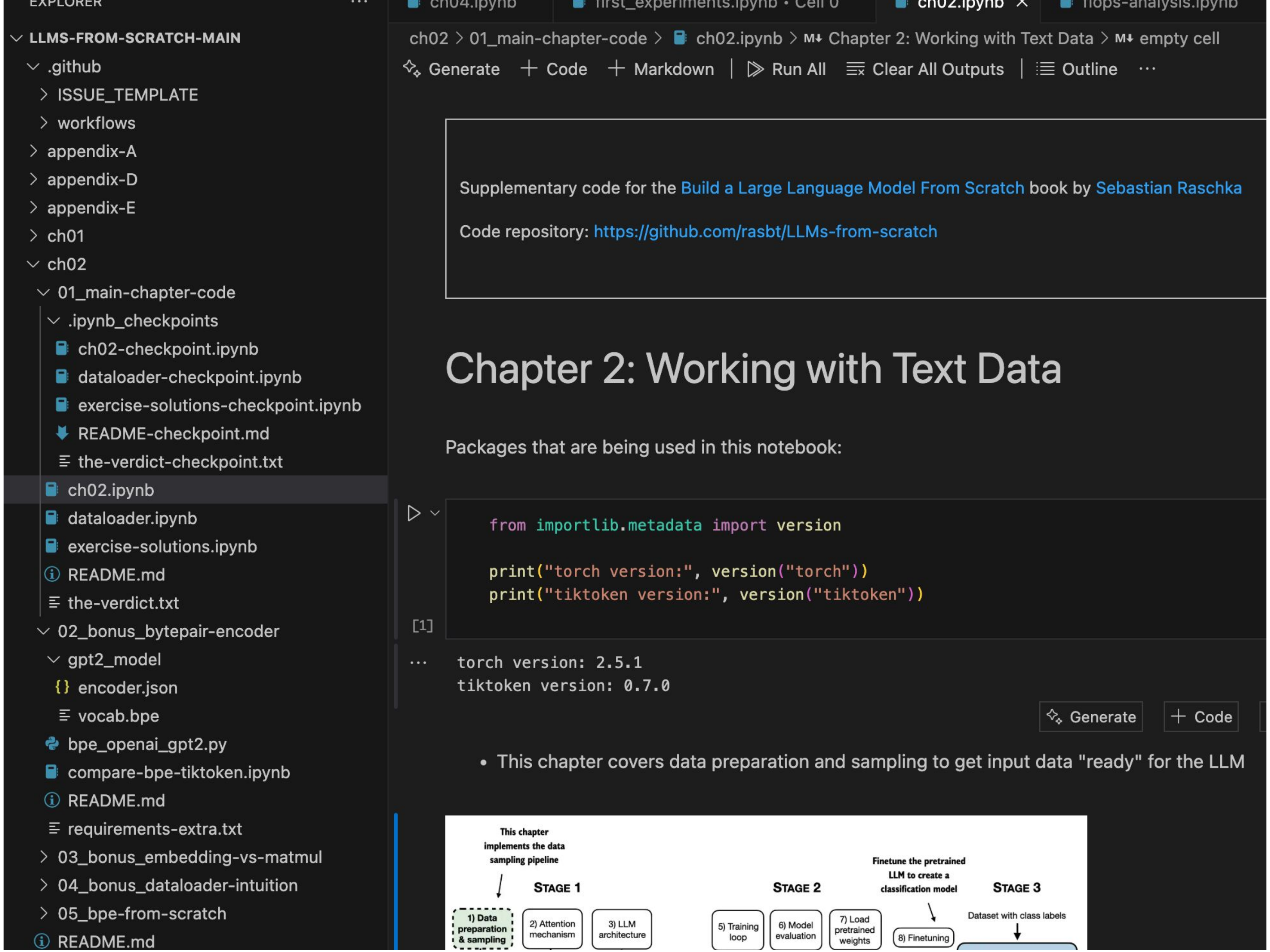
To read

Papers

1. *Konstantin Yakovlev* et al. Toolken+: Improving LLM Tool Usage with Reranking and a Reject Option, 2025
2. *Shibo Hao* et al. ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings, 2023
3. *Hugo Touvron* et al. LLaMA: Open and Efficient Foundation Language Models, 2023

Sandbox examples

1. Token_Example.ipynb
2. TinyGPT_from_Scratch.ipynb
3. https://github.com/karpathy/nanoGPT
4. https://github.com/facebook/prophet


EXPLORER
ch04.ipynb
first_experiments.ipynb • Cell 0
ch02.ipynb
flops-analysis.ipynb
LLMS-FROM-SCRATCH-MAIN
.github
ISSUE_TEMPLATE
workflows
appendix-A
appendix-D
appendix-E
ch01
ch02
01_main-chapter-code
.ipynb_checkpoints
ch02-checkpoint.ipynb
dataloader-checkpoint.ipynb
exercise-solutions-checkpoint.ipynb
README-checkpoint.md
the-verdict-checkpoint.txt
ch02.ipynb
dataloader.ipynb
exercise-solutions.ipynb
README.md
the-verdict.txt
02_bonus_bytepair-encoder
gpt2_model
encoder.json
vocab.bpe
bpe_openai_gpt2.py
compare-bpe-tiktoken.ipynb
README.md
requirements-extra.txt
03_bonus_embedding-vs-matmul
04_bonus_dataloader-intuition
05_bpe-from-scratch
README.md
ch02 > 01_main-chapter-code > ch02.ipynb > Chapter 2: Working with Text Data > empty cell
Generate
Code
Markdown
Run All
Clear All Outputs
Outline
Supplementary code for the Build a Large Language Model From Scratch book by Sebastian Raschka
Code repository: https://github.com/rasbt/LLMs-from-scratch
Chapter 2: Working with Text Data
Packages that are being used in this notebook:
from importlib.metadata import version
print("torch version:", version("torch"))
print("tiktoken version:", version("tiktoken"))
[1]
torch version: 2.5.1
tiktoken version: 0.7.0
Generate
Code
• This chapter covers data preparation and sampling to get input data "ready" for the LLM
This chapter implements the data sampling pipeline
STAGE 1
1) Data preparation & sampling
2) Attention mechanism
3) LLM architecture
STAGE 2
5) Training loop
6) Model evaluation
7) Load pretrained weights
Finetune the pretrained LLM to create a classification model
STAGE 3
8) Finetuning
Dataset with class labels