LLMs in Bioinformatics

A student's step-by-step workshop companion

Start with no machine-learning background. Read the explanation, try the exercise, check the output, then state what it means.

Use this guide beside the seven workshop notebooks and the accompanying *Hugging Face 101* slides. It explains their exercises without modifying their code. File names below omit the extra `(1)` added during upload; both names identify the same supplied copy.

The workshop's question is: **How can we turn biological data into useful predictions without confusing a model's output with biological evidence?**

You will retrieve records, train small RNA classifiers, explore a protein language model, extract publication data, and examine predictions. These are related activities, not one matched experiment. Publication data connect to Day 2; the final small-protein example is a separate interpretation test.

How to read this guide. Read the background immediately before each exercise. In a step, **Do** tells you what to run or complete; **Check** identifies what to inspect; **Explain** asks you to interpret it. You do not need to memorize definitions or reproduce whole programs. Notebook answers and hints are available when needed.

Finding a step. "Cells 15–20" counts all text and code cells from 1, not execution counters. Locate the named notebook part first, then use the cell range. [N1–N7]

Solutions. Several files already contain instructor solutions. Predict what the completed code will do before running it. Replace only actual `___` or unfinished `...` placeholders using the hints; a "TODO" comment may already accompany completed code.

Cautions. A **Notebook caveat** identifies a source issue without changing the code. Ask the instructor before modifying the affected analysis.

Your route through the workshop

Chapter	Open this file	The question you will answer
1	`01_APIs.ipynb`	Did I retrieve the biological information I intended?
2	`02_basic_ML.ipynb`	Can sequence measurements predict an existing RNA label?
3	`04_Transformers.ipynb`	How does a pretrained model represent and score a sequence?
4	`05_Data_Retrieval.ipynb`	Can AI-written code extract correct, traceable publication data?
5	`06_AlphaGenome.ipynb`	What does the model predict for a genomic change?
6	`07_Folding.ipynb`	What changes in a predicted protein structure, and what does not?
7	`08_result_sense.ipynb`	Does a confidence score answer my biological question?

Use `04_slides.pdf`, the Hugging Face introduction, with Chapter 3. [H1]

Before you begin: the minimum vocabulary

In the biological examples, **DNA** (deoxyribonucleic acid) and **RNA** (ribonucleic acid) are sequences of units called **nucleotides**. A **protein** is a chain of **amino acids**; an amino acid in the chain is a **residue**. The sequence letters specify these units, not their three-dimensional positions.

A **model** is a mathematical rule that turns an input into an output. Its adjustable numbers are **parameters**. **Machine learning (ML)** means using examples to determine some of those parameters. **Training**, also called **fitting**, is that adjustment process. **Prediction**, also called **inference**, applies an already fitted model. Downloading data or loading a model is not itself training.

A **dataset** is a collection of observations. An **observation**, or **sample**, is one item being analyzed. A **feature** is an input measurement describing it. A **target** is the answer a model is meant to predict; a categorical target is often called a **label**. For example, one RNA is a sample, its length is a feature, and its existing

coding/noncoding annotation is a label. The annotation is not automatically an independently verified experimental fact.

In **supervised learning**, training examples include targets. **Classification** predicts categories; **regression** predicts numerical values. **Unsupervised learning** fits patterns without those targets. **Clustering** is an unsupervised task that groups similar inputs. These ideas are worked through in Chapter 2; no model knowledge is assumed beforehand.

Work safely in Colab

A **notebook** is a document containing text cells, code cells, and possibly saved outputs. A **runtime** is the running computer process that executes the code and remembers variables. **Google Colab** provides a notebook interface connected to such a runtime. A **CPU** is a general-purpose processor; a **GPU** is a processor suited to many parallel numerical operations. Early exercises use a CPU. Folding may require a GPU or a stated saved-results route.

Save your own notebook copy. Run cells in order with the play button or **Shift+Enter**, reading each instruction first. Save your observations too.

Installing a package makes software available to the runtime; **importing** makes its names available to your Python session. A package is a collection of reusable code. The notebooks' setup cells do both. If you restart, rerun imports and the earlier cells that create the required variables. Old displayed tables may remain visible even though their variables no longer exist.

Data files created in a temporary runtime are not automatically preserved just because the notebook is saved. Download the required output files or follow the instructor's storage instructions before leaving a session. Never put an access key or confidential research data in a shared notebook.

Read the small amount of Python you need

Notation	Meaning
`name = value`	Store a value under a variable name. `==` instead tests equality.
`"text"`, `[a, b]`, `(a, b)`	A text string; a list; a tuple (a fixed ordered group of values).
`{"name": value}`	A dictionary: values stored under named keys.
`items[0]`, `seq[1:4]`	The first item; a slice starting at index 1 and stopping before index 4. Python indices start at 0.
`len(x)`, `str(x)`, `float(x)`	Count items; convert to text; convert to a decimal number.
`def f(x):` / `f(value)`	Define a named function; call it with an input. Its returned value is the result.
`for item in items:`	Repeat the indented instructions for each item.
`assert condition`	Stop with an error if a specified check is false.

A **DataFrame** is a pandas table: rows with named columns. **pandas** is the package used to work with these tables. **NumPy** works with numerical arrays; an **array** is an organized collection of values. Its **shape** lists the sizes of its dimensions, such as `(100, 3)` for 100 rows and three columns. `np` and `pd` are short names created by importing NumPy and pandas.

Record which kind of result you obtained

A **live result** was obtained by your current database or model request. A **cache** is a previously saved result reused instead of running the request again. A **recorded demonstration** is a result supplied in the notebook. **Synthetic data** are deliberately constructed examples, not downloaded biological observations. Record the mode: these routes are useful, but they establish different things.

For each session, write three short lines: **What I ran. What I observed. What I cannot conclude.**

1. Retrieve biological data

Notebook: `01_APIs.ipynb` · Follow Setup and Parts 1–4, 6. Parts 5 and 7 are optional. [N1]

Understand the request before running it

A **database** stores records: individual entries with **identifiers**, often called **accessions**. A **sequence** is an ordered nucleotide or amino-acid string. An **annotation** describes a record, such as its organism. A gene summary and a fetched sequence are different objects.

An **application programming interface (API)** lets programs communicate. An **endpoint** is the address for an operation. Your code sends a **request** with **parameters** describing the desired records and receives a **response**. These are request settings, not learned model parameters. **HTTP** is the communication protocol: status 200 usually indicates success, 429 a request-rate limit, and 5xx a server error. None establishes biological correctness.

GET requests information; **POST** supplies a request body for processing. Either can support a search. **JSON**, JavaScript Object Notation, is structured text with named values and lists. `response.json()` converts it to Python objects; this is inappropriate for non-JSON responses.

The recurring pattern is **request → inspect → parse → validate → save**. To **parse** is to separate structured text into usable components. **Provenance** is the record of where the data came from and what was done to them.

Step 1.1 — Open the notebook and read the supplied sequences

Do — Setup; Part 1; cells 4–12. Run setup. Run the cell writing `sample_mirna.fasta`. **FASTA** is a text sequence format: `>` starts a header, followed by sequence lines. A new `>` begins the next record. The supplied examples are precursor RNA strings, not fresh database downloads.

Fill the parser's filename blank with `"sample_mirna.fasta"`. Biopython's `SeqIO.parse` reads records; `rec.id`, `rec.description`, and `rec.seq` refer to their identifier, full header, and sequence. In the table-building cell, use `str(rec.seq)` for the sequence and `len(rec.seq)` for its length. A list of dictionaries becomes a DataFrame: one dictionary per row.

Check. You should obtain three records with lengths 80, 72, and 65, and a table containing identifiers, descriptions, sequences, and lengths. These numbers check the supplied strings; they do not verify their current database annotations.

Explain. Which part was stored in the file, which part was parsed, and did any database request occur?

Step 1.2 — Search NCBI, then retrieve a gene summary

Do — Part 2.1; cells 15–23. NCBI is the US National Center for Biotechnology Information. The notebook uses its Gene database. `esearch` finds identifiers matching `MIR21[gene] AND human[orgn]`; `esummary` returns details for an identifier.

Complete the first request using `search_params`. A **timeout** limits how long the request waits. `raise_for_status()` raises an error for an unsuccessful HTTP status; it does not check the gene name. Inspect `gene_ids` before using `gene_ids[0]`, the first returned item. If the list is empty, stop rather than continuing with `[0]`.

Complete the summary step using `summary_resp.json()`. Read the nested keys from the outside in: `summary_json["result"][gene_ids[0]]` selects the returned result and then the chosen gene record.

Check. Verify human MIR21, its chromosome and description. Do not select an unrelated first hit merely to obtain a nonempty result.

Explain. You have a gene summary, not the mature RNA sequence and not a protein sequence.

Step 1.3 — Retrieve a UniProt protein record

Do — Part 2.2; cells 25–27. UniProt is a protein sequence and annotation resource. Use `uniprot_url` in the request. The example accession is `P05067`, for human amyloid-beta precursor protein, APP. The code prints nested fields for the protein name, gene names, and sequence length.

Check. Verify APP before checking the expected 770-residue length. Investigate differences rather than editing the result. This cell prints metadata, not the full sequence.

Explain. APP is an independent example. It is not the protein product of the MIR21 gene queried earlier. For the optional accession change, predict which identifying fields should change before rerunning.

Step 1.4 — Turn an API response into a table

Do — Part 2.3; cells 29–36. RNAcentral brings together noncoding RNA records. The response's `results` is a list; each `r` is one record dictionary. Complete the two table fields with `r.get("length")` and `r.get("rna_type")`. `.get()` returns `None` when the key is absent, rather than raising the missing-key error produced by `r["key"]`.

Check. Examine descriptions, RNA types, identifiers, and any available organism fields. Are they consistent with the requested human let-7 query? A row count alone is not an adequate check.

> **Notebook caveat — query validation.** This request supplies `query` to a record-list endpoint; do not assume it applied the filter. RNAcentral documents text searching separately. If unrelated RNAs appear, report an unvalidated retrieval and ask for the approved route. [N1, cells 29–36; B2]

Save the file only with an accompanying note about what was actually retrieved. **CSV** means comma-separated values: a plain-text table. `index=False` avoids exporting pandas' row counter as an additional column.

Step 1.5 — Decode and send a structured search

Do — Part 3; cells 37–45. RCSB PDB is a portal to the Protein Data Bank, a structure resource. Follow the browser demonstration if directed. Browser developer tools show requests a page makes; the Console executes JavaScript. Only run inspected commands that you understand and that the instructor approves.

Back in Python, the long page address contains a search in its `request=` parameter. **URL encoding** represents special characters with codes such as `%7B` for `{`. Complete `parse_qs(urlparse(page_url).query)`, then `json.loads(query_params["request"][0])`. The first extracts and decodes parameters; the second parses the JSON search specification.

POST `decoded_request` to the search endpoint. **Check** the status and returned identifiers. The example requests entry `1B23`. A returned error dictionary is not a successful result merely because `.json()` can parse it. If the endpoint rejects a field, ask the instructor rather than randomly deleting fields.

Step 1.6 — Read a web table and export a reusable result

Do — Part 4; cells 47–53. HTML is the tagged text used to structure a web page. `<table>` marks a table, `<tr>` a row, `<th>` a header cell, and `<td>` a data cell. **Web scraping** extracts content from the page itself. BeautifulSoup is the parser used to find these tags.

Complete the table-tag blank with `"table"` and the DataFrame input with `rows`. A **codon** is a three-nucleotide unit used in translation. Inspect the headings before accepting the selected codon table. A changed page can alter the number and order of tables. The supplied cleanup removes reference markers and writes `scraped_table.csv`; inspect whether a meaningful value was accidentally changed.

Do — Part 6; cells 61–65. A **pipeline** is a sequence of processing steps. Complete the same two dictionary lookups inside `build_rna_dataset`, then call the function and inspect its result before export. Defining a function makes the recipe available; calling it performs the work.

> **Notebook caveat — this is a metadata table.** `session1_rna_dataset.csv` contains identifiers, descriptions, lengths, and RNA types, not a sequence column or verified classifier targets. It is not loaded as the training dataset by `02_basic_ML.ipynb`, which performs its own retrieval. The RNAcentral query caution also applies here.

Optional routes and exit check

Optional Part 5 uses **Selenium**, a browser-automation package; **headless** means no visible browser window. Part 7 uses **Docling** to extract PDF text and tables. A **PDF** is a document format. Compare extraction with the original: layout, units, or table contents can be lost.

Before leaving: identify a saved file, explain its rows, and state how you checked the request. Preserve it beyond the runtime.

2. Learn from RNA sequence features

Notebook: `02_basic_ML.ipynb` · Setup; Parts 1–6; optional clustering. [N2]

Understand the model's inputs and answers

Here a row represents one RNA sequence, and the intended label is coding or noncoding. Coding RNA carries instructions used to make a protein; a **long noncoding RNA (lncRNA)** is an RNA annotated as noncoding and longer than short noncoding-RNA classes. The word "lncRNA" in a query does not prove every returned record belongs to that class.

Feature engineering means choosing and calculating numerical descriptions of the inputs. This notebook uses nucleotide length, GC percentage, and a simple open-reading-frame length. **GC percentage** is 100 times the fraction of G or C letters. An **open reading frame (ORF)** is a candidate run of codons read without an intervening in-frame stop. A **reading frame** specifies where those three-letter codons begin. The supplied helper looks for AUG-started runs in three frames and reports a length in nucleotides.

`X` is the input table. `y` is the corresponding target list. With 200 observations and three features, `X.shape` is `(200, 3)` and `y.shape` is `(200,)`. The first entry in `y` belongs to the first row of `X`. Keep that pairing when splitting or reordering.

Illustrative input only: a row `[600, 48.0, 300]` means length 600 nucleotides, GC 48%, and the helper's ORF-length value 300. Those are not three observations; they describe one observation. The label is stored separately and must not be included as an input feature. [N2, Part 2; B1]

Step 2.1 — Decide which question each toy dataset asks

Do — Part 1; cells 7–13. Run the three small, invented datasets. A supplies category labels: classification. B supplies numerical targets: regression. C withholds targets: clustering. These plots illustrate task types, not biological findings.

For optional **k-means clustering**, choose the requested number of clusters, `n_clusters=2`. A **vector** is an ordered list of numbers. A **cluster centre** is the average feature vector of the points assigned to a group. K-means alternates between assigning points to nearby centres and updating those centres. For an illustrative one-feature group containing 1 and 3, the centre is 2. The group numbers are arbitrary names, not coding/noncoding labels.

`random_state` fixes a random seed, the initial state used to reproduce randomized choices. `n_init=10` tries multiple starting arrangements. Neither setting guarantees biologically meaningful groups.

Check. Which plots used known answers? Could a clustering algorithm produce a grouping unrelated to the biological distinction you care about?

Step 2.2 — Inspect the real-data or synthetic-data route

Do — Part 2.1–2.3; cells 15–20. Run retrieval and read its message. It requests noncoding sequences from RNAcentral and coding sequences from NCBI. If requests fail or too few records are obtained, it generates synthetic sequences. Record which route you used before interpreting scores.

Complete `raw_df["label"].value_counts()` to count each class. `.shape` reports rows and columns; `.isna().sum()` counts missing entries. A **missing value** is an unavailable value, not a measured zero. Inspect both classes before continuing.

Remove duplicates using the `"sequence"` column and inspect the row counts before and after. Normalization changes letters to uppercase and T to U for this RNA exercise. It does not establish that the sequence is biologically valid.

> **Notebook caveat — labels and duplicates.** The RNAcentral fetch assigns "non-coding" without retaining enough metadata to validate the requested subtype. The same search issue as Chapter 1 remains. Also, duplicates are removed before case and T/U normalization, so equivalent normalized strings can survive. Ask the instructor to resolve questionable records. Synthetic examples must be reported as synthetic; their scores are not evidence about RNA biology.

Step 2.3 — Calculate features and encode labels

Do — Part 2.4–2.5; cells 23–28. Fill `.apply(longest_orf_length)`: `.apply` calls the function once for each sequence. Examine the new columns. On the illustrative string `ACGUAC`, length is 6 and GC percentage is 50, because three of its six letters are G or C.

A **histogram** groups numerical values into intervals and shows how many observations fall in each interval. Compare the two class histograms, but do not assume the printed statement about overlap matches your run.

`LabelEncoder` maps names to integers. Here the intended mapping is coding = 0 and noncoding = 1. Inspect the printed mapping. These numbers are category codes, not a claim that one class is "more" than the other.

Check. `X` will contain the three feature columns, not sequence identifiers or the label. The ORF helper is deliberately simplified and can count a run to the sequence end without finding a stop; its number is not a validated gene prediction.

Step 2.4 — Understand scaling and data splitting

Scaling changes numerical representation. Standardization subtracts a feature's mean and divides by its standard deviation, a measure of spread. For an illustrative training mean of 500 and standard deviation of 100, a length of 600 becomes `(600 − 500)/100 = 1`. This is not a probability.

A **training set** supplies examples used to fit model parameters. A **test set** is held aside for a final performance check. A **validation set**, or validation folds, helps choose settings before that final check. **Generalization** means performing usefully on relevant new observations, not merely reproducing training answers.

Do — Part 2.6–2.7; cells 31–34. Read the scaling code and the split. The required `test_size` is `0.2`, meaning 20% is assigned to testing. `stratify=y` aims to preserve class proportions. The resulting variables pair training inputs with training labels and test inputs with test labels.

> **Notebook caveat — preprocessing leakage. Data leakage** is using information during development that should be unavailable for the evaluation claim. Cell 31 fits the scaler on all records before the split. For a valid held-out workflow, split first, fit preprocessing only on training data, and apply that same transformation to test data. The supplied exercise can demonstrate the commands, but its evaluation needs this correction before a research claim. Do not describe its test set as untouched by preprocessing. Ask the instructor for the approved correction rather than silently changing the exercise. [B3]

Step 2.5 — Fit three kinds of classifier

A **classifier** predicts a category. A **decision tree** learns a sequence of questions about features, such as whether an ORF-length feature exceeds a threshold. A **random forest** combines many trees trained with random variation in examples and candidate features. The combined prediction can be more stable than that of a single tree, but this must be checked.

Logistic regression is also a classifier here, despite its name. It learns a weighted score from the features and converts it into estimated class probabilities: numbers between 0 and 1 assigned to the possible categories. With three features, the score has the form `w1 × feature1 + w2 × feature2 + w3 × feature3 + b`. The fitted weights `w1`, `w2`, `w3` and offset `b` are parameters. You need not derive the probability formula to use it.

Do — Part 3; cells 36–40. Fit the tree with `X_train, y_train`. Then run logistic regression and the forest with the requested 200 trees. The pattern is:

```
# Variables and model type are defined in the notebook.
model.fit(X_train, y_train)   # learn parameters using known answers
prediction = model.predict(X_test)  # apply the fitted rule; do not fit again
```

`.predict()` does not need the correct test labels. Those are used afterward for evaluation. **Accuracy** is the fraction of predictions matching the labels. A **baseline** is a simple reference method, such as always predicting the commonest training class, against which a model can be compared.

Check. Compare training and test accuracy. A large gap can indicate **overfitting**: the fitted rule captures details that do not transfer well to new data. It is a warning, not a complete diagnosis. No particular model is guaranteed to win.

Step 2.6 — Compare settings without reusing the final answers

Cross-validation repeatedly fits a model on subsets and evaluates on a withheld subset. Five-fold cross-validation uses five such partitions. Each observation serves as a validation example once. A **hyperparameter** is a setting chosen outside fitting, such as tree depth or number of trees. **Grid search** tries combinations of these settings and compares their validation results.

Do — Part 4; cells 42–46. Fill `cv=5`. Read all five scores and their spread, not just the mean. Optional `GridSearchCV` tries the notebook's settings on `X_train, y_train` and selects a fitted model. If you skip it, the following cell uses the default forest.

> **Notebook caveat — what was evaluated?** Cell 42 runs cross-validation on all of `X, y`, including rows used as the earlier test set. This is a different evaluation design, not five new independent tests. A research workflow would keep final test data outside model selection and fit preprocessing inside each training fold. A **software pipeline** can bundle preprocessing and fitting to enforce that order. [N2, cells 31, 42, 45; B3]

Step 2.7 — Read the errors, not only one score

Before running evaluation, define **positive** as the class coded 1: **noncoding** in this notebook. A **true positive** is a noncoding RNA predicted noncoding. A **false positive** is a coding RNA predicted noncoding. A **false negative** is a noncoding RNA predicted coding. A **true negative** is a coding RNA predicted coding.

Precision asks: among positive predictions, how many are correct? **Recall** asks: among truly positive examples, how many were found? **F1** combines precision and recall as `2 × precision × recall / (precision + recall)` when that denominator is nonzero.

Do — Part 5; cells 48–55. Complete the F1 function name with `f1_score`. A **confusion matrix** tabulates actual versus predicted labels; in this notebook rows are actual labels and columns predictions. Read the axis labels before interpreting a cell.

Illustrative arithmetic, not a workshop result: 18 correct coding predictions, 2 coding-to-noncoding mistakes, 4 noncoding-to-coding mistakes, and 6 correct noncoding predictions give accuracy `24/30 = 0.80`, noncoding precision `6/8 = 0.75`, recall `6/10 = 0.60`, and F1 about 0.67.

`predict_proba` returns estimated class probabilities. Its second column corresponds to class 1 here. A **threshold** converts a probability to a category. The **true-positive rate** is recall. The **false-positive rate** is the fraction of truly negative examples incorrectly predicted positive. The **receiver operating characteristic (ROC) curve** plots these two rates while changing the threshold. **Area under the curve (AUC)** summarizes ranking: 1 means perfect positive-over-negative ranking; 0.5 is chance-level ranking. It is not a measure of whether a probability such as 0.8 is well calibrated. **Calibration** concerns agreement between stated probabilities and observed frequencies.

Check. Use `y_proba` in the AUC calculation. The later discussion text reverses the positive-class interpretation: follow the printed 0/1 mapping and the definitions above, not that sentence. If the output is undefined because a class is absent, investigate the class counts.

Step 2.8 — Interpret, save, and optionally cluster

Do — Part 6 and Save; cells 58–62. Attach `feature_cols` to forest importances. **Feature importance** here summarizes how features contributed to the trees' training splits. It is not proof of a causal biological mechanism. Save `session2_rna_dataset_clean.csv` and `session2_model_results.csv`, together with your data-source and evaluation cautions.

Optional — cells 64–69. Cluster `X` without passing `y`. After fitting, compare clusters with labels. The **adjusted Rand index (ARI)** compares group assignments while adjusting for chance: 1 indicates identical grouping, 0 chance-level agreement, and negative values can occur. Cluster 0 is not automatically class 0.

Exit check. Explain what `X_train` and `y_train` contain, why `.predict()` needs only new inputs, and one reason a high score would not yet demonstrate reliable biological annotation. The next notebook uses ubiquitin, not this dataset; it is not a direct benchmark of these classifiers against embeddings.

3. Transformers, representations, and Hugging Face

Notebook: `04_Transformers.ipynb` · Parts 0–4; Part 5 is optional. Companion: `04_slides.pdf`. [N3; H1]

What happens between a sequence and a prediction?

A **language model** learns patterns in sequences of symbols. For this exercise the symbols describe amino acids, not prose. A **pretrained model** has already had its parameters learned from a training collection. You will use those parameters for inference, not retrain them.

A **token** is one unit processed by the model. A **tokenizer** splits an input into tokens and assigns integer **token IDs** from a **vocabulary**, the set of recognized tokens. An ID is a lookup index, not a meaningful numerical measurement: ID 20 is not "twice" ID 10 biologically.

An **embedding** is a numerical vector representing a token or input; recall that a vector is an ordered list of numbers. A learned lookup converts token IDs to initial embeddings. A **contextual representation** changes according to the surrounding input. A **neural network** is a model that processes inputs through layers of learned numerical transformations. A **transformer** is a neural-network architecture that builds contextual representations using a mechanism called **attention**.

Attention computes weighted combinations of representations; these input-dependent weights differ from the fixed model parameters learned in training. Its learned **query**, **key**, and **value** vectors are numerical projections: queries are compared with keys to calculate weights, which combine values. These are not database queries. One **attention head** performs one such calculation; several heads operate within a layer.

Tiny illustrative example: weights `0.2, 0.3, 0.5` on vectors `[1,0]`, `[0,1]`, `[1,1]` yield `[0.7,0.8]`. The weights determine the mixture. They do not, by themselves, establish which biological interaction caused a prediction.

A **model head** is the output component for a task. A masked-language-model head outputs a **logit**, an unnormalized numerical score, for each possible token at a hidden position. **Softmax** transforms those logits into nonnegative probabilities that sum to 1 over the vocabulary. The probability is conditional on the input and model; it is not a measurement of protein function.

Step 3.1 — Load the matched tokenizer and model

Do — Part 0; cells 4–7. Hugging Face hosts model repositories and provides software to use them. A repository is a versioned collection of files. Its **model card** describes intended use, training data, limitations, and license. The `transformers` Python package loads models; PyTorch is the numerical framework executing the network. Read the model name rather than choosing by popularity. Public availability does not remove license conditions. [H1, pp. 4–5, 11]

Load `facebook/esm2_t6_8M_UR50D`. **ESM-2** is the protein language model used here. This small configuration has six layers and twenty attention heads per layer. `.eval()` selects inference behavior; `torch.no_grad()` avoids recording information needed to calculate training **gradients**, which describe how a training-error measure changes as parameters change. Neither call fits parameters to ubiquitin.

The supplied `ubq` string is the 76-residue ubiquitin sequence. Confirm its length before continuing. The notebook requests an attention implementation that returns attention weights; if it cannot return them, ask the instructor before the attention exercise.

Check. Record the model name and device. Unexpected missing model weights deserve investigation; do not treat every loading warning as harmless.

Step 3.2 — Compare tokenization

Do — Part 1; cells 9–16. Compare the English text tokenizer, protein tokenizer, and DNA chunking helper. A **k-mer** is a consecutive sequence of k letters. Overlapping 6-mers advance one base; nonoverlapping ones advance six.

For 30 bases, overlapping 6-mers yield `30 − 6 + 1 = 25` chunks, and nonoverlapping 6-mers yield 5. For ubiquitin, count `len(enc["input_ids"])`: 76 residues plus start and end tokens gives 78. **Special tokens** are markers such as `<cls>`, `<eos>`, and `<mask>` rather than amino acids. Obtain the mask ID through `tok.mask_token_id`.

A **context window** limits how many tokens the model can process at once. Token count and biological sequence length are therefore different quantities.

Do — Part 1b; cells 18–24. Byte-pair encoding (BPE) learns a vocabulary of variable-length pieces. Inspect DNABERT-2's DNA tokenizer and GPT-2's text tokenizer on the supplied biological names. Only tokenizers are being used for those comparisons, not the corresponding prediction models. Count tokens per character and identify the most fragmented string.

Check. An `rs` identifier names a variant record; a transcript accession identifies a particular sequence record; `p.Cys61Gly` is a protein-change description. The tokenizer need not preserve each as one token. Fragmentation does not prove that a model cannot represent the name's meaning, and it does not perform a database lookup.

> **Notebook caveat — discarded remainder.** The nonoverlapping helper keeps only complete chunks. The 64-base DNA example produces ten complete 6-mers and leaves four bases out. Dividing 64 by 10 prints 6.4, but each retained chunk still contains six bases; the extra bases were not encoded. Inspect **coverage**, the fraction of the original input actually retained, as well as token count.

Step 3.3 — Hide one residue and inspect the answer distribution

Self-supervised learning creates a training target from the data itself, such as hiding an existing residue and predicting it. That is how the exercise motivates pretraining. Today, however, masking a residue is an inference experiment using fixed weights. **Fine-tuning** would update pretrained weights using additional training data; this notebook does not do that.

Do — Part 2; cells 26–30. Read `masked_predictions`. Python sequence positions are 0-based: residue number 46 is `my_pos=45`. The model token position is `pos + 1` because `<cls>` comes first. Select a position, write your guess, and inspect the top predictions and probability assigned to the actual residue.

A **tensor** is a multidimensional numerical array. Here `logits` has dimensions for batch, token position, and vocabulary. A **batch** is a group of inputs processed together. For one ubiquitin input, the first two dimensions are 1 and 78; the third has one entry per vocabulary token. `logits[0, pos+1]` selects a vector, and softmax converts it to probabilities.

Use the actual residue's token ID to read `probs[true_id]`. Top predictions are alternatives, not experimentally observed mutations. The vocabulary includes special tokens, so the code's full probability vector is not restricted to exactly twenty entries.

Check. A valid probability lies between 0 and 1. Do not expect the true residue to rank first at every position.

Step 3.4 — Repeat masking over the protein

Do — Exercise 2b; cells 32–35. Hide one residue at a time, leaving the others visible, and record the probability of the correct residue. `p_true` should contain 76 numbers. The plotted horizontal mean averages those probabilities.

The color grouping marks residues selected as hydrophobic, meaning they tend to favor non-water environments. A difference between colors does not establish that hydrophobicity caused the probabilities.

Explain. A low probability means the observed residue was not strongly favored by this model under this masking experiment. It does not by itself mean the protein is dysfunctional or the database sequence is wrong.

Step 3.5 — Inspect attention without treating it as an explanation

Do — Part 3; cells 37–46. Inspect `A.shape`: after removing special-token rows and columns, the supplied model produces `(6, 20, 76, 76)`. An **axis** is one dimension of an array. These axes are layer, head, receiving residue, and contributing residue. A **heatmap** uses color to show the entries of a matrix, a two-dimensional array.

The notebook renormalizes rows after removing special tokens. A bright diagonal means nearby sequence positions receive much weight. A distant bright spot means a large separation in sequence index—not necessarily a physical contact in three-dimensional space.

For average attention distance, `dist[i,j] = |i-j|` is the separation along the sequence. Multiplying by weights, summing over `j`, and averaging over `i` gives one number per layer/head, hence shape `(6,20)`. Illustratively, weights 0.8 and 0.2 at distances 1 and 6 give weighted distance 2.

Check. The largest and smallest values identify different attention patterns. Code indices start at 0; printed layer/head labels add 1. You are not measuring physical distance here. An **angstrom** (Å), used later for atomic coordinates, is one ten-billionth of a metre. The optional interactive viewer can be skipped.

Step 3.6 — Use a scrambled-sequence control

A **control** is a comparison designed to help interpret a result. Here a **scrambled sequence** preserves the amino-acid counts but changes their order. That tests sensitivity to order beyond composition, not foldability directly.

Do — Part 4; cells 50–55. Read the batched helper: it creates one masked copy for each position, rather than hiding all residues in one input. Check that its probabilities agree with the slower loop within numerical tolerance. Then shuffle the letters using the supplied seed and compare native and scrambled scores.

Check. Length and letter counts must match; order must differ. Record your actual mean probabilities. A lower scrambled score is consistent with order-sensitive sequence learning in this example. One scramble of one familiar protein does not demonstrate universal function prediction or an independent benchmark against unseen proteins.

Exit check. Explain the difference between a token ID, an embedding, an attention weight, and a masked-residue probability. No RNA-classifier comparison or separate embedding-based classifier is implemented in this notebook; the Hugging Face slides provide additional background, not evidence that those exercises were run.

4. Retrieve data from publications with a text model

Notebook: `05_Data_Retrieval.ipynb` · Parts 0–5. Skip the instructor-preparation section at the end. [N4]

Know the sources and the output

A **text large language model (text LLM)** generates text or code from input context. A **prompt** supplies instructions and context. Here **vibe coding** means requesting code, then inspecting and testing it. Generated code is a proposal, not a verified analysis.

A **cell line** is a population of cells maintained for research. **CCLE**, the Cancer Cell Line Encyclopedia, supplies molecular data for cancer cell lines. **Multi-omics** combines molecular layers: DNA mutations and **copy number** (quantity of a genomic region), RNA abundance, and **proteomics** (protein measurements).

Keep the two papers distinct: Jarnuczak et al. 2021 supplies text through Europe PMC; Nusinow et al. 2020 supplies protein tables through the authors' site. A **PubMed Central identifier (PMCID)** names an article; here the text example is `PMC8065022`.

XML, Extensible Markup Language, is tagged structured text used for the article. A **supplementary table** accompanies a paper. A **schema** specifies required columns or fields. The target table has one row per `(cell_line, gene)`, not per patient or mutation.

Step 4.1 — Set up and state a precise code contract

Do — Part 0; cells 3–6. Check that `ccle_panel.csv` downloaded. It is instructor-prepared RNA, copy-number and mutation data, not extracted article text. Parts 1–3 can proceed without it; the final handoff cannot.

Before asking for code, specify the source, function name, inputs, returned fields, error behavior, and independent checks. An **edge case** is an unusual input, such as an invalid identifier or missing file, that the function must handle intentionally. The notebook's prompt checker searches for words or patterns. A high checker score does not prove the prompt is clear or its generated code is correct.

The **safety net** contains reference functions used when a participant's function fails. Record when it was used. Read generated code before running it; do not approve arbitrary deletion, installation, shell commands, or transmission of private data.

Step 4.2 — Retrieve the article without losing its contents

Do — Part 1; cells 8–16. Write your prompt for `get_paper(pmcid)`. The output must include `title`, `sections`, `text`, and the retrieval route. Use the notebook's named routes: structured full text, then PDF-derived text, then an explicitly supplied local text file if needed.

The file already contains a reference solution; replace it only in your saved copy or record that you used it. `verify_part1` checks text length, an invalid ID, and strings `191`, `0.58`, `7,171`. These checks are article-specific, not universal rules.

Check. Does the title identify the right paper? Are the expected sections present? A PDF fallback may supply its opening text as a provisional title, which needs manual review. Passing a substring check does not establish complete extraction.

The deliberately flawed function keeps only 5,000 characters with `[:5000]`. Explain what it might lose. Check invalid-ID behavior too: an accidental exception is not necessarily the informative error required by your contract.

Step 4.3 — Download and inspect the supplementary tables

Do — Part 2; cells 18–29. Table S1 contains sample information. Table S2 contains protein measurements. An Excel file can have multiple sheets; `.csv.gz` is a compressed text table. `header=None` initially treats every row as data, letting you inspect where the actual column headings begin.

Inspect `get_supplements` and `clean_table`. Tables may contain notes, extra headings, mixed values, and missing entries. pandas often represents missing numbers as **NaN**, "not a number." Do not replace missing with zero or assume every missing protein value means nondetection.

Inspect the catalogue before accepting `S1_KEY` and `S2_KEY`. The code guesses the protein matrix from its width. **Wide format** puts samples in separate columns. Verify gene/sample headings; size alone does not identify a table.

Check. Use the dimensions, remaining missing values, and ERBB2 entry as sanity checks. The material gives approximate counts from a published dataset; live versions may differ. A biological expectation, such as high ERBB2 in some lines, is not an identity check on every row.

Run `clean_table_flawed` and inspect which rows disappeared. Its unrestricted `dropna()` removes a protein if any sample entry is missing. That can change the population being analyzed, even though no programming error occurs.

Step 4.4 — Extract facts and verify their supporting text

Do — Part 3; cells 31–37. Ask the assistant for the four specified facts as JSON. Each item must contain the fact name, value, and an exact supporting sentence. JSON `null` means no value is supplied; it is not a zero.

> **Notebook caveat — supply the whole intended passage.** Cell 31 prints only a 1,500-character preview even though its directions tell you to copy the abstract and results. In your own copy, add `print(RESULTS_TEXT)` to view that full variable before supplying the text. Check its headings and content. Do not expect the assistant to extract facts from passages it was never given.

Also, cell 34 constructs `AI_JSON` from reference facts. To test an actual assistant answer, replace that assignment with your own JSON string. Leaving it unchanged tests the reference answer, not a new model response. Keep that distinction in your record.

Check. `verify_facts` tests whether the quoted words occur in the paper, whether the value occurs in the quote, and whether the value matches the notebook's reference. Read the surrounding paragraph too. A real sentence may concern a different cohort or endpoint. A failed exact-string match may reflect paraphrasing or extraction formatting; it is not automatically evidence of fabrication.

A **hallucination**, in the text-LLM sense used here, is unsupported or fabricated generated content. A requested `null` when the passage lacks the answer can be the correct response, even if the notebook's fixed-answer checker flags it for review. Do not force a value to make a check pass.

Step 4.5 — Build the shared table and check the join

Do — Part 4; cells 39–46. A **join** matches rows from two tables using **keys**, the fields identifying the same observation. Here both `cell_line` and `gene` are required. **Long format** gives one row per cell-line/gene pair; `melt` reshapes sample columns into rows.

Sample names such as `AU565_BREAST_TenPx05` contain a cell-line identifier and a measurement-batch suffix. A **batch** here means a group of laboratory measurements, not the model-input batch from Chapter 3. Removing the specified suffix makes the key compatible with the prepared file. Repeated measurements are averaged by the reference code; inspect that grouping.

`rna_log2tpm` is the prepared RNA-expression column: TPM means transcripts per million, and "log2" indicates a base-2 logarithmic scale. Check the source's precise transformation and any added constant before making quantitative comparisons. It is not in the same units as `protein`.

> **Notebook caveat — requested versus implemented table.** The prompt requests a left join, which keeps every left-hand key, and asks to retain missing measurements. The reference implementation drops missing protein observations and uses an inner join, retaining only keys present on both sides. Report this retained-observation scope. Do not describe the resulting table as containing all missing gene/cell-line combinations.

The validation checks duplicate keys, leftover suffixes, implausible inserted zeros, and agreement between RNA and protein. **Correlation** measures association. The code uses **Spearman correlation**, which compares ranks, ranging from −1 to 1 when defined; constant values can make it undefined. A positive value indicates that higher values tend to occur together. It does not prove correct row matching or causation. The quoted reference correlations also summarize different studies or populations; they are not exact targets for this small panel.

Save only after inspecting the checks. Preserve `ccle_multiomics_panel.csv` with its source, filtering, dimensions and code route. The save cell does not enforce all previous checks; the file may still need correction. **Exit check:** Why are two join keys needed, and why must you read a quote in context?

5. AlphaGenome: predict effects of genomic changes

Notebook: `06_AlphaGenome.ipynb` · Parts 0–3; Part 4 contains optional extension and troubleshooting. [N5]

Day 2 files. AlphaGenome needs the instructor's approved `ccle_mutations.csv` and `ccle_panel.csv`: variant coordinates and RNA measurements. Folding uses your `ccle_multiomics_panel.csv`. The publication exercise does not create every required file. Download helpers may reuse existing local files; confirm their versions.

What is predicted, and from which input?

AlphaGenome predicts molecular signals from DNA. It is not a conversational assistant. Python runs in Colab; the model runs through a hosted API. You request inference, not training on these data.

A **genome assembly**, or **genome build**, is a particular reference sequence and coordinate system. This notebook uses human GRCh38, also called hg38. A **locus** is a genomic region. A **variant** describes a difference from a reference sequence; its **reference allele** is the reference sequence at the site and its **alternate allele** is the proposed replacement. An **SNV** is a single-nucleotide variant, one base replaced by another.

Genes have a **strand**, the direction in which their sequence is read. The supplied mutation table uses alleles on the forward genomic strand even when the gene lies on the reverse strand. Genomic positions and protein positions are different coordinate systems.

A transcript is an RNA copy from a gene. **Exons** are the portions retained in a spliced transcript; **introns** are removed. **Splicing** joins exons, using signals near donor and acceptor boundaries. The notebook groups variants as splice-related, nonsense (creating a premature stop), missense (changing an amino acid), synonymous (not changing the amino acid), or synonymous near a splice region. These annotations describe types of change, not guaranteed effects on RNA abundance.

Step 5.1 — Connect without exposing the key

Do — Part 0; cells 3–4. An **API key** is a credential authorizing access. Obtain one through the instructor-approved service process and store it under `ALPHA_GENOME_API_KEY` in Colab Secrets, with notebook access enabled. Never paste it into a shared code cell or saved output. Read the service's current terms before sending data.

Install the notebook's packages and use the supplied v1 client. A newer tutorial may use another interface; do not mix interfaces by changing individual imports. The test sequence in `connect` is artificial and checks communication, not biology.

Check. The function can print "connected" after a non-authentication test failure. Inspect preceding errors and the first prediction; the banner and historical outputs are not proof of your successful run.

Step 5.2 — Inspect the TP53 region and a predicted track

Do — Part 1; cells 6–9. The annotation file describes gene locations and transcripts. GENCODE is the annotation source; MANE Select identifies a representative transcript choice used by the helper. An **interval** is a chromosome segment with start and end coordinates. A **base pair (bp)** is the length unit here; kb and Mb are approximate thousand- and million-base units.

Read TP53's interval and strand. The code resizes the requested interval to a supported input length; this changes the surrounding context, not the gene itself. It requests **RNA-seq**, RNA-sequencing-related signal, for the named right-liver-lobe context. An **ontology term** is a standardized identifier for a biological concept, here a tissue.

A **track** is a sequence of signal values along genomic coordinates, with metadata describing what was predicted. Inspect the exons and strand-specific tracks before describing the pattern. This is one model context, not a measurement of every CCLE cell line.

Step 5.3 — Construct and validate the variant inputs

Do — Part 2; cells 11–20. Check that the two prepared files exist and contain the required fields. Filter TP53 and inspect the class counts. **TODO 1** formats chromosomes once as `chr17`, converts the position to an integer, and uppercases the alleles.

A **1-based** system calls the first base position 1. A **0-based, half-open interval** includes its start and excludes its end. For example, a single base at 1-based position 100 corresponds to interval `[99,100)`. AlphaGenome's variant position is 1-based; some interval and reference-service operations use other conventions. The supplied helper performs the conversions. [B4]

The reference lookup compares each stated reference allele with GRCh38. Review every excluded mismatch. The 90% aggregate assertion is a warning threshold, not permission to score mismatches: the code removes individual mismatches afterward.

Check. A matching base is necessary but not sufficient to establish correct coordinates, gene, or strand. An accidental match is possible. Do not complement alleles merely because TP53 is on the reverse strand: the supplied variant table is defined on the forward genomic strand.

Record how many distinct variants and carriers survive. A **carrier** here is a cell line with that recorded variant. The notebook scores each distinct variant once, even if several lines carry it.

Step 5.4 — Turn predicted tracks into scores

Do — Part 3; cells 24–27. A **variant scorer** summarizes the difference between reference and alternate predictions. The two selected output types concern splice-site signals and RNA-seq. A **raw score** is a scorer-specific numerical result, not automatically a probability or clinical label.

Read `tidy`, the returned table. It can contain many rows for each variant because genes and biological contexts differ. The notebook filters TP53 and summarizes with the maximum absolute splice score and the mean RNA score across tracks. "Absolute" discards the sign, retaining magnitude. The RNA mean retains direction under this scorer's definition.

Check. Confirm the selected output and gene before grouping. One summarized value per variant is not the same as a directly predicted RNA measurement in a particular carrier cell line. Averaging different contexts can conceal opposing effects. Use the notebook's v1 definitions when interpreting score direction. [B4]

Step 5.5 — Compare predictions with observations carefully

Do — cells 29–31. A **z-score** expresses a measurement relative to a reference mean and standard deviation: `(value − mean)/standard deviation`. The notebook averages the carriers' RNA, then standardizes it against lines without a recorded TP53 mutation in the supplied table. Negative values indicate lower RNA relative to that reference group.

This reference is "no mutation recorded in this table," not proof of an experimentally verified mutation-free genome. Before plotting, inspect how many predictions and observations matched. Identifier-format mismatches can silently remove rows. If the summary unexpectedly becomes empty or much smaller, stop and inspect the join.

Check. Read axes and class counts before the rank correlation. A printed **p-value** summarizes how extreme the calculated test quantity is under a specified no-association model and its assumptions; it is not the probability that a biological explanation is true. Small groups, shared carrier lines, tissue differences, other mutations, and copy number limit this comparison.

> **Interpretation caution.** The notebook's proposed class ranking is a hypothesis. Premature stops do not always reduce RNA; synonymous and missense changes can affect RNA processing. **Nonsense-mediated decay** is a cellular pathway that can reduce some premature-stop-containing transcripts. This exercise does not prove that a DNA-input model can never learn associations with downstream processes. Nor does a between-cell-line association isolate a variant's causal effect.

Optional — Part 4. Liftover maps coordinates between genome assemblies. Read whether the mapping succeeds and recheck the reference; a changed build cannot be handled by relabeling the old coordinate.

Exit check. State the build, allele convention, selected model output, and one reason the predicted effect and observed RNA comparison are not a controlled experiment.

6. ESMFold: from a protein sequence to predicted coordinates

Notebook: `07_Folding.ipynb` · Parts 0–4; Part 5 is optional. [N6]

Structure is a different prediction target

Folding stability concerns a protein's tendency to retain a folded state under specified conditions. **Protein function** concerns what it does, such as binding another molecule or catalyzing a reaction. Neither is measured merely by drawing predicted coordinates.

ESMFold combines a protein language model with a **folding head**, the component that predicts three-dimensional structure. It takes an amino-acid sequence and outputs coordinates and confidence estimates. It does not simulate a molecule physically folding over time. A **wild-type** sequence is the reference sequence for the comparison; a **mutant** differs from it.

A protein **domain** is a region with a structural or functional role. Selected fragments reduce computing cost. An **isoform** is an alternative protein form, potentially with different numbering. Check reference identity and coordinates before mutation.

The exercise compares structural summaries with cell-line measurements. A predicted structure is not an assay of activity, stability, or degradation.

Step 6.1 — Check the data and choose the execution route

Do — Part 0; cells 3–4. Load the approved `ccle_multiomics_panel.csv`. Verify that it has the RNA, protein, and mutation columns and that it is your validated handoff or an explicitly supplied reference copy.

`FOLD_MODE="api"` sends sequences to the hosted service; `"local"` runs inside Colab on a suitable GPU. Local here is still cloud computation, not your computer. Model downloads and memory demands can be substantial; timing estimates are not guarantees.

The helper may reuse a structure or fall back to GPU computation. A **hash** is a fingerprint of input bytes, used here to name saved structures by sequence. Record the route: a hash alone does not document the model version, settings, or a fresh run.

Step 6.2 — Parse substitutions and check reference residues

Do — Part 1; cells 6–9. `p.R175H` means reference residue R at protein position 175 is replaced by H. A **missense substitution** changes an amino acid. A stop change and a **frameshift**, which changes the translation frame, are different kinds of variation.

A **regular expression** is a pattern for recognizing text. The supplied parser recognizes `p.` followed by one letter, digits, and one letter. Its returned tuple contains reference residue, integer position, and new residue. Several changes separated by `;` can be returned separately.

Check. Run the parser checks and verify the reported reference residues against UniProt. Matching position 175 requires `sequence[174]` because Python starts at 0. Unexpected isoforms or naming conventions need review, not forced substitutions.

> **Notebook caveat — scope.** The parser deliberately ignores other change types. This does not mean that only missense-derived sequences can ever be submitted to structure predictors; it means other consequences require constructing and validating a different input sequence. A missing record is not proof of no biological variation.

Step 6.3 — Extract and fold the TP53 fragment

Do — Part 2; cells 12–13. The TP53 fragment is residues 94–312 inclusive, giving `312 − 94 + 1 = 219` residues. Python takes `sequence[93:312]`: start at index 93 and stop before 312. Other selected genes and fragments are listed in `GENES`; do not change them mid-comparison without rerunning dependent steps.

A **PDB file** is a text structure format. In a predicted PDB, atom lines contain coordinates. An **angstrom (Å)** is a length unit equal to one tenth of a nanometre. A **C-alpha atom**, written `CA`, is a backbone atom used here as one representative coordinate per residue.

Check. Confirm length, execution route, and that the response contains a structure rather than an error page. Retrieval is not experimental validation.

Step 6.4 — Read confidence residue by residue

pLDDT, predicted local distance difference test score, estimates local structural accuracy. The workshop displays it on a 0–100 scale. It is not the probability that the protein folds or is functional. A low score may reflect several causes, including flexibility, **disorder** (lack of one stable local conformation), missing context, or model uncertainty; it is not a definitive disorder experiment. [B5]

Do — TODO 2; cells 15–17. Parse only `ATOM` lines whose atom name is `CA`. Read the fixed coordinate and confidence fields. The coordinate array should have shape `(219,3)` for this fragment; the confidence array should contain 219 values. The two-residue artificial test validates the parser's arithmetic only.

The predicted file stores confidence in a field conventionally called the **B-factor**. In experimentally determined structures, that field has a different meaning; do not interpret an experimental B-factor as pLDDT.

Check. The source parser multiplies scores by 100 if the largest value is at most 1. That is a heuristic for different exporter conventions. Confirm the source convention with the instructor rather than assuming any small number needs multiplication. Inspect the profile, not just its mean.

Step 6.5 — Compare wild type, mutants, and a scrambled control

Do — Part 3; cells 22–24. The mutation function replaces one position in a fragment. For full-protein R175H in a fragment starting at residue 94, the fragment's 1-based position is `175 − 94 + 1 = 82`, and its Python index is 81. Confirm the reference letter before making the change.

Structural alignment rotates and translates corresponding coordinates to make them comparable without deforming either structure. The supplied Kabsch method finds such a fit. **Root-mean-square deviation (RMSD)** summarizes the remaining coordinate distances after alignment: square each distance, average, then take the square root. Report the length unit and which residues were compared. Two already-aligned example distances of 1 Å and 3 Å give RMSD `sqrt(5) ≈ 2.24 Å`.

The notebook compares R175H, R248Q, and R273H with the reference fragment and folds a composition-preserving scramble. "Small RMSD" means these predicted backbones are similar under the stated alignment. It does not establish unchanged binding, activity, stability, or abundance.

Check. Matching residues matters more than equal array length. Aligning coordinates cannot fix mismatched fragments. Report your run, not a numerical interpretation saved from an earlier run.

Step 6.6 — Explore structural summaries against cell-line measurements

Do — Part 4; cells 27–33. The notebook's **burial proxy** counts other C-alpha atoms less than 10 Å away. A **proxy** is an indirect approximation. It is not a calculated solvent-accessible area or a physical energy. In the artificial five-point test, neighbours are `[2,3,4,3,2]`; subtracting one excludes each point from its own count.

Candidates must meet the requested carrier and reference counts. Keep `MIN_LINES`, `MIN_REF`, and `PER_GENE` unchanged unless directed. If no candidates survive, ask for help rather than lowering thresholds until a pleasing result appears.

For each gene, protein and RNA measurements are separately standardized relative to lines with no recorded mutation. The summary `protein_minus_rna` is the difference between these two z-scores, averaged over selected carriers. A negative value means the standardized protein level is lower than the standardized RNA level in that comparison.

> **Interpretation caution.** A **residual** is an observed value minus a value predicted by a fitted model. This difference is not such a residual: no model predicting protein from RNA is fitted here. It does not establish degradation. Many biological and measurement factors, including tissue and other alterations, can affect the two measurements.

Read the plots by gene as well as across all points. A pooled association can be driven by differences between genes or by one observation. `dpLDDT_site` is wild-type minus mutant local confidence; positive values indicate a drop. None of the three summaries has a guaranteed relation to abundance.

Exit check. Write one result, one limitation, and one independent check. An experimental structure can test coordinates; an appropriate activity or stability measurement tests a different question. Optional reference comparisons require matching residue identities and coverage, not just coordinate counts.

7. Make sense of confidence and disagreement

Notebook: `08_result_sense.ipynb` · Parts 0–6; optional extensions follow. [N7]

Define the question before comparing the models

AlphaFold2 is a protein-structure predictor. **ColabFold** provides a practical workflow for running structure prediction, including AlphaFold2. A **multiple sequence alignment (MSA)** arranges related sequences into corresponding positions, providing evolutionary context. ESMFold instead uses a protein language model at inference without requiring that alignment search. Different inputs and architectures do not imply statistically independent evidence: training sources and biases can overlap.

You already met pLDDT, which concerns local structural accuracy. **pTM**, predicted template modelling score, estimates overall structural agreement on a 0–1 scale. It is not a measured folding probability. A local score and a global score can differ without logically contradicting one another. [B5]

This notebook studies **GB1**, the B1 domain of protein G, a small 56-residue domain. Its **hydrophobic core** contains residues that pack away from water. The experiment substitutes selected positions with aspartate, denoted D, and compares the confidence scores. This is a stress test, not a direct measurement of folding stability.

> **Evidence boundary.** The supplied notebook calls a variant "impossible" and says it "cannot fold." It does not include an experimental stability measurement for that exact variant. For your report, distinguish the notebook's biological hypothesis from the observed model scores. Use "strongly perturbed core variant" rather than turning the premise into measured ground truth. Likewise, a shuffled sequence is a comparator, not a universal "no idea" floor.

Step 7.1 — Establish what will be live and what is recorded

Do — Part 0; cell 5. Read the loading outcome. `MODEL_OK` is a Boolean flag, `True` or `False`, indicating whether ESMFold loaded. It does not guarantee every later fold succeeds. If loading fails, the notebook can use recorded summaries but cannot generate a new arbitrary variant or supply missing coordinates.

In Part 2, the `AF2` dictionary contains pasted results described by the notebook as an earlier ColabFold run. This file does **not** run AlphaFold2 or ColabFold live. Do not write that you performed such a run.

The optional precision discussion explains memory settings: **fp16** and **fp32** store floating-point values using different numbers of bits. Less precision can save memory but cause numerical problems. Leave the supplied configuration to the instructor; do not dismiss unexpected missing-weight or NaN warnings.

Step 7.2 — Inspect the variant series and controls

Do — Part 1; cells 7–10. `WT` is the reference sequence; `CORE` lists the chosen 1-based positions. `core_00D` has no substitutions, while `core_10D` changes all ten listed positions. **Sequence identity** here is the percentage of matching positions in two same-length strings; ten changes out of 56 leave about 82.1% unchanged.

Choose `my_n`, an integer from 0 to 10, and inspect the resulting sequence. The shuffled and reversed controls retain the same residue composition but change order. Core substitution variants change both identity and composition, so the different comparisons control different things.

Check. Count the intended changes and preserve the same input strings for model comparisons. High sequence identity is not a measurement of stability or function.

Step 7.3 — Read the two metrics before thresholding them

Do — Parts 2–3; cells 12–20. Look up a variant by its exact name. For `core_10D`, the supplied recorded values are pLDDT 76.9 and pTM 0.489. These are source values, not a new run or independently verified measurements.

Use 70 and 0.5 as the thresholds requested by the exercise. The script labels one score above its cutoff and the other below. Explain what each cutoff is being applied to; do not describe "passes pLDDT" as "will fold." The plot multiplies pTM by 100 only to share an axis. That multiplication does not make the two scores equivalent quantities.

Check. The labels “confident,” “fold probably wrong,” and “DISAGREE” are display rules in this exercise. They are not calibrated universal decisions about foldability. Report the actual score and its meaning, not only the printed verdict.

Step 7.4 — Run ESMFold and label every fallback

Do — Part 4; cells 25–30. Write your prediction before the comparison. Run the supplied ESMFold function if the model loaded. Then run the backup cell and note precisely which names it fills. A single results table can mix newly computed and recorded rows. Keep a mode note for each input rather than calling the whole table “live.”

Try `esmfold(my_variant, "my_variant")` only when loading succeeded. A new variant has no automatically valid recorded result merely because a neighbouring variant does.

Check. Missing or nonfinite scores are not zero. The notebook’s stored backup contains means, not full per-residue arrays or structure coordinates. The viewer and per-residue comparison may therefore be unavailable. Do not reconstruct those missing data from a mean.

Step 7.5 — Explain agreement and disagreement without overclaiming

Do — Part 5; cells 32–36. Read the table before the structure viewer. `diff` is ESMFold pLDDT minus AlphaFold2 pLDDT: negative means a lower numerical confidence value, not proof of lower true accuracy. Ensure you are comparing the same sequence and note the different settings and modes.

In the viewer, colors represent the chosen score field. Verify its convention before equating a color with the normalized numbers in the table. A convincing image is still a prediction.

A supported conclusion might be: **“For the supplied sequences and recorded settings, the models’ confidence summaries respond differently to the substitutions. This does not determine the variants’ measured stability.”** Add your actual result and state which rows were live. There is no requirement to reproduce the notebook’s preferred biological verdict.

Step 7.6 — Finish with a claim that could be checked

Do — Part 6; cell 37, with the limitations in cell 47. Separate four items: the input manipulation, the observed model output, the interpretation, and the evidence still missing. The supplied limitations describe one protein and limited prediction settings; do not upgrade that to a general comparison of model quality.

Out-of-distribution input means an input unlike the data on which a model was trained or evaluated. It can make familiar confidence conventions less informative. **Miscalibration** means a mismatch between stated probabilities or expected accuracy and observed performance across suitable cases; without measured outcomes, this exercise alone cannot quantify it.

Recall **data leakage** from Chapter 2. A target previously used for training or model selection is not an independent generalization test. A familiar protein may be useful for a sanity check while still being unsuitable as evidence of performance on unseen proteins. We have not audited the training-set overlap of every example here.

Text hallucination, a prediction error, and using a correct score to answer the wrong question are different problems. Your main task is to identify which claim the evidence supports—not to label every surprising output a hallucination.

Optional. Compare per-residue scores only for genuine full outputs. The alanine challenge requires a new live fold. Saving `both_models.pkl` writes Python objects in a serialization format; only open pickle files from trusted sources because loading them can execute code. Keep the model settings and live/backup notes beside the saved results.

Exit check. State what pLDDT estimates, why two scores can differ, and what independent evidence would be needed to choose a biological explanation.

Finish: what to keep and how to use this in your research

Keep your completed notebooks, actual output files, source identifiers, model/settings notes, and a clear record of live versus supplied results. An output screenshot is helpful, but not a substitute for the data and steps that produced it.

For your own project, complete this short plan: **one observation represents ___; the input is ___; the desired output is ___; a simple comparison is ___; the evaluation data are ___; an acceptable first result would be ___; the analysis must stop if ___.**

An **acceptance test** is a check specified before trusting an output. A useful first project is small enough to produce one inspected result and one meaningful test. You do not need to start by training a large model.

Quick troubleshooting

Message or symptom	First check
`NameError`	Was the cell defining the variable or import run in this runtime?
`ModuleNotFoundError`	Did setup finish successfully in this runtime? Ask the TA about the approved environment.
`FileNotFoundError`	Is the exact required file in the current runtime, not just another notebook's session?
`KeyError` or empty table	Print the available fields and check that the returned data match the expected schema.
HTTP 429, timeout, or service error	Record the failed request, wait as directed, and use only an identified approved fallback.
Memory error while folding	Stop repeated runs; ask the TA to clear the runtime or supply genuine saved outputs.
A failed assertion	Read what was tested. Do not remove the assertion to make the notebook continue.
A surprisingly excellent result	Check labels, duplicates, feature leakage, evaluation design, and whether the result is live.

Short self-check answers

1. Retrieval: A successful response establishes communication success, not that the requested organism, molecule, or annotation was returned. The supplied FASTA is local input; the API exercises are separate requests.

2. ML: Rows of `X` contain input features. Matching entries of `y` contain target labels. `.fit()` uses paired examples; `.predict()` applies the learned rule to inputs. Evaluation requires separate target information and a defensible data split.

3. Transformers: An ID names a token; an embedding is a numerical representation; attention weights mix representations; masked-token probabilities are task-specific outputs. None alone is a measured function.

4. Publications: Join on the full biological key. A source sentence must exist, concern the intended population and endpoint, and support the stated value. Exact substring matching checks only part of this.

5. AlphaGenome: Build, position convention, reference allele and strand must agree. Model outputs in reference contexts and observations across different cell lines are not paired experimental interventions.

6. Folding: A small difference between predicted backbones does not prove unchanged activity or stability. The protein-minus-RNA score is a difference of standardized measurements, not a fitted prediction residual.

7. Confidence: Confidence concerns a particular prediction task. Model agreement is not ground truth, and different architectures do not guarantee independent evidence. A claim about stability needs independent stability evidence.

Source map and scope

This companion was written from the exact uploaded copies below. **N** references identify notebook-derived steps; **H** identifies the supplied slides. Added background definitions and explicitly marked methodological cautions are teaching explanations, not claims that the notebooks were changed or their live results reproduced.

- **N1:** `01_APIs(1).ipynb` — 73 cells. Retrieval, file parsing and optional browser/document tools.
- **N2:** `02_basic_ML(1).ipynb` — 70 cells. RNA features, classification, evaluation and optional clustering.
- **N3:** `04_Transformers(1).ipynb` — 58 cells. ESM-2, ubiquitin, tokenization, masking, attention and scrambling.
- **N4:** `05_Data_Retrieval(1).ipynb` — 50 cells. Publication text, supplementary data, evidence checks and CCLE handoff.
- **N5:** `06_AlphaGenome(1).ipynb` — 35 cells. TP53, prepared genomic-variant/RNA files and v1 model scoring.
- **N6:** `07_Folding(1).ipynb` — 36 cells. ESMFold, mutation parsing, fragments and cell-line comparisons.
- **N7:** `08_result_sense(1).ipynb` — 47 cells. GB1 variants, recorded AlphaFold2 scores, ESMFold and interpretation.
- **H1:** `04_slides(1).pdf`, *Hugging Face 101* — especially pages 7–9 on tokenization/model outputs and page 11 on model selection.

The original notebooks and their historical outputs are unchanged. Cell numbers and small arithmetic examples were checked for this guide. Database access, model downloads, external API authentication, and GPU inference were not rerun as part of producing this tutorial. Their numerical outcomes must come from your workshop run or be identified as supplied results.

Background references

These primary documentation sources support the added definitions and cautions. They are not a replacement for the workshop's named models and versions. Follow the supplied v1 AlphaGenome notebook rather than substituting a newer API.

B1. scikit-learn, *Getting Started* — inputs, targets, fitting and prediction. scikit-learn: Getting Started

B2. RNAcentral, *Text search* and linked API documentation — search syntax and documented routes. RNAcentral: Text search

B3. scikit-learn, *Common pitfalls and recommended practices* — preprocessing and data leakage. scikit-learn: Common pitfalls

B4. AlphaGenome, *AlphaGenome 1 quick start* and *Variant* API reference — v1 requests, reference/alternate predictions, positions and scoring. AlphaGenome 1 quick start ; AlphaGenome Variant reference

B5. EMBL-EBI, *pLDDT: Understanding local confidence* and *Confidence scores in AlphaFold-Multimer* — local versus global confidence. EMBL-EBI: Local confidence ; EMBL-EBI: Global confidence

B6. ESM authors' repository and Hugging Face ESM documentation — the model family and implementation. ESM authors' repository ; Hugging Face ESM documentation