

Deep-Learning fundamentals

1.) Is our method of Deep-Learning supervised learning?

- Yes, our method is **supervised learning** because:
 - The model is trained on **input–output pairs**, where the output (sensitive or resistant) is derived from the $\ln(\text{IC}_{50})$ value.
 - The labels are **known in advance**, making it a labeled classification problem.
 - The model **learns a mapping** from gene expression + chemical features → binary label.
 - Supervised learning is preferred because we have **historical labeled drug response data** from large-scale screening studies like GDSC.

2.) In essence, is the output of layers in a multi-layer perceptron (a neural network) always the sum of products of activations and the weights and bias?

- Yes — in each layer, the output is computed as:

$$z = \mathbf{w} \cdot \mathbf{a} + b$$

where \mathbf{a} is the activation from the previous layer, \mathbf{w} is the weight vector, and b is the bias.

- This is a **linear transformation**, representing a **weighted sum** of the previous layer's outputs.
- The result is passed through a **non-linear activation function** (e.g. ReLU or sigmoid) to form the neuron's output.
- This operation is repeated in **all hidden layers**.
- In the **final layer**, the same formula applies, but we use a **sigmoid activation** to convert the output into a **probability between 0 and 1** — suitable for binary classification (sensitive vs resistant).

3.) What is an activation function? (e.g sigmoid and ReLu)

- An **activation function** is a **non-linear transformation** applied to the output of each neuron after the weighted sum:

$$z = \mathbf{w} \cdot \mathbf{a} + b \quad \Rightarrow \quad a = \phi(z)$$

- In our model:
 - We use **ReLU** in the hidden layers:

$$\phi(z) = \max(0, z)$$

Efficient and avoids vanishing gradients

- We use **sigmoid** in the final layer:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Outputs a **probability between 0 and 1**, suitable for binary classification

- It allows the network to **learn complex, non-linear relationships** — without it, even deep networks behave like a linear model.

4.) What is a loss function? Which loss function have we used and why?

- A **loss function** measures how far the model's predictions are from the true labels — it quantifies the **error**.
- It returns a **single scalar value**, which is used to guide weight updates via gradient descent.
- In our project, we used **Binary Cross-Entropy (BCE)** because the task is **binary classification** (sensitive vs resistant).
- BCE compares the predicted probability \hat{y} from the sigmoid output with the true label y , penalizing confident wrong predictions more heavily:

$$\mathcal{L}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

- BCE works well when the model outputs a **probability between 0 and 1**, making it a perfect match for our sigmoid-activated output neuron.

5.) How does gradient descent work? What does gradient descent do to the weights?

- **Gradient descent** is an optimization algorithm that updates the model's weights to **minimize the loss function**.
- After each forward pass and loss calculation, we compute the **gradient of the loss** with respect to each weight:

$$\frac{\partial \mathcal{L}}{\partial w}$$

Each weight is updated in the direction that **reduces the loss**:

$$w \leftarrow w - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$

where η is the **learning rate**.

- This update is done **simultaneously for all weights** after processing each batch.
- Repeating this process over many iterations gradually **improves the model's predictions** by finding a set of weights that minimize the error.

6.) How does backpropagation work?

- **Backpropagation** is the algorithm used to compute **gradients** of the loss with respect to all weights and biases in the network.
- After the loss is calculated from the model's output, backpropagation applies the **chain rule of calculus** to trace how each parameter in each layer contributed to the error.
- It **backtracks**, starting from the **output layer**, through the **hidden layers**, down to the **input layer weights**, applying the chain rule at each step.

At each layer, it computes:

$$\frac{\partial \mathcal{L}}{\partial w}, \quad \frac{\partial \mathcal{L}}{\partial b}$$

for every weight and bias, based on the **error from the next layer**.

- These gradients are then used by the optimizer (e.g. Adam) to **update the parameters** in a way that reduces the loss.
- This process is repeated after each batch during training, allowing the model to **learn which weights improve prediction**.

7.) Why and how do we compute gradients?

- We compute gradients to understand **how much each weight and bias influences the loss** — they show the **direction and magnitude** of change needed to reduce error.
- Gradients are calculated **after the forward pass and loss computation**, using **backpropagation**.

- Mathematically, each gradient is a **partial derivative** of the loss with respect to a parameter:

$$\frac{\partial \mathcal{L}}{\partial w}$$

- These gradients are then used by an **optimizer** (like Adam) to **update the weights** and make the model more accurate over time.
- Without gradients, the model would have **no feedback signal** to learn from — so computing them is essential to training.

8.) What do optimizers such as Adam do?

- Optimizers are algorithms that use gradients to **update the model's weights and biases**, helping the model minimize its loss.
- Adam is a widely used optimizer that combines the benefits of:
 - Momentum** (uses moving averages of gradients to smooth updates)
 - Adaptive learning rates** (each parameter gets its own step size)
- Adam tracks:
 - The **first moment** (mean of gradients)
 - The **second moment** (mean of squared gradients)
- It then updates each weight using both:

$$w \leftarrow w - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- This makes Adam efficient, stable, and effective — especially in high-dimensional or noisy settings like gene expression and drug feature data.

9.) How does the activation function differ from an optimization function?

- Activation functions** are used **inside the network**, after each layer's linear transformation:
 - They introduce **non-linearity**, allowing the network to learn complex patterns.
 - Examples: ReLU, sigmoid, tanh
- Optimization functions (optimizers)** operate **outside the network**, after the loss is computed:
 - They use gradients to **update the weights and biases** during training.

- Examples: Gradient Descent, Adam, RMSprop

Feature	Activation Function	Optimization Function
Role	Adds non-linearity to neuron output	Updates weights to minimize loss
Used during	Forward pass	After backpropagation
Examples	ReLU, Sigmoid	Adam, SGD

10.) How is loss different from accuracy?

- **Loss** is a continuous value that measures how far off the prediction is from the true label — it's what the model actually learns from.
- **Accuracy** is a discrete metric that tells you the **percentage of correct predictions** (e.g. predicted label matches the true label).
- Loss is used **during training** to compute gradients and update weights
- Accuracy is used **after training steps** to evaluate model performance.
- Loss takes into account **confidence** in predictions (e.g., predicting 0.49 vs 0.9 for a true label of 1 results in very different losses).
- Accuracy only checks whether the prediction is **above or below 0.5** in binary classification.
- In imbalanced datasets (like yours), **accuracy can be misleading**, while **loss (and metrics like AUPRC)** give a better reflection of model behavior.

Full sequence of Events (Summary)

♦ 1. Forward Pass

1. Input features are fed into the model.
2. Each layer computes:

$$z = \mathbf{w} \cdot \mathbf{a} + b$$

3. Then, the **activation function** is applied:

$$a = \phi(z) \quad (\text{e.g., ReLU or sigmoid})$$

4. This continues layer by layer until the **final output** is produced (e.g., a probability from a sigmoid).

♦ 2. Loss Computation

- The predicted output \hat{y} is compared to the true label y , using the **loss function** (e.g., binary cross-entropy).
- This results in a **single scalar loss value** for the batch.

♦ 3. Backpropagation

- Now that the loss is known, we **compute gradients** of the loss with respect to:
 - All weights and biases
 - All intermediate activations (via the chain rule)
- Backpropagation starts at the **output layer** and works backward through the network, layer by layer.

♦ 4. Weight Update (via Optimizer)

- The optimizer (e.g., Adam) uses the gradients to update each parameter:

$$w \leftarrow w - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$

- This completes one training step.

Graph Neural Networks (GNNs)

1.) What are the inputs to a GNN in biological applications?

1.) Graph Structure:

- The biological system is represented as a graph:
 - **Nodes:** drugs, genes, proteins, pathways, or diseases
 - **Edges:** known relationships such as “inhibits,” “targets,” “interacts with,” etc.
- In **multi-relational knowledge graphs**, each edge has a **type** that defines the nature of the interaction.

2.) Node Features:

- Each node is assigned a **feature vector** that describes its properties. Examples:
 - **Drug node:** chemical fingerprints (e.g. 2048-bit Morgan fingerprints)
 - **Gene node:** gene expression levels from cell line datasets
 - **Protein node:** structural or sequence-based features

These vectors serve as the **initial embeddings** that are updated through message passing.

3.) Edge types of edge features

- In **heterogeneous graphs**, edge types may also be encoded (e.g., “binds”, “activates”, etc.).
- **Relational GCNs (R-GCNs)** use **relation-specific transformations** — each edge type has its own weight matrix.
- **Graph Attention Networks (GATs)** can incorporate **attention-based edge weights** — dynamically learned per edge.

2.) What is a Graph Neural Network, and how does message passing work?

A GNN is a neural network designed to operate on graph-structured data. It updates each node’s representation by aggregating information from its neighbors.

How does message passing work?

1. Message

- Each node sends its feature vector (embedding) to its neighbors.

2. Aggregation

- A node collects the embeddings from its neighbors (e.g., using sum, mean, or attention).

3. Update

- The aggregated message is combined with the node's current embedding to compute its updated embedding (e.g., via linear transformation and activation function).

Layer-wise propagation

- After **1 layer**, a node embeds information from its **1-hop neighbors**
- After **2 layers**, it captures **2-hop neighborhood** info
- After **k layers**, the node embedding reflects its **k-hop neighborhood**

3.) What is a node embedding, and what does it mean for a node to move in embedding space?

Node Embedding

- A node embedding is a learned vector that summarizes the **features** and **structural context** of a node in the graph.
- It's what the GNN produces as an output for each node after message passing.

Embedding Space

- It's a high-dimensional space where each node is represented by its embedding vector.
- Nodes with similar roles or relationships tend to **cluster together** in this space.

What does it mean for a node to move in embedding space?

- As training progresses, **each node's embedding is adjusted using gradients** to minimize the loss.
- This causes nodes to **shift positions in the embedding space**, clustering closer to nodes they are **meant to be similar or connected to**.

4.) What is the role of the embedding space in capturing similarity or structure in graphs?

What does the embedding space represent?

- The embedding space is where nodes are represented as vectors that reflect their **structural roles, features, and relationships** within the graph.

How does it capture similarity?

- Nodes with **similar features** or **similar neighborhood structures** are mapped to **closer points** in this space.
- For example:
 - Drugs with similar targets
 - Genes with similar expression profiles
 - Proteins in related pathways
- These will have **closely aligned embeddings** (i.e., high cosine similarity or small Euclidean distance).

Why is this useful?

- It allows downstream tasks like:
 - **Node classification** (e.g., predicting if a cell line is drug-sensitive)
 - **Link prediction** (e.g., predicting drug–gene interactions)
 - **Clustering** (e.g., grouping similar drugs)
 - **Regression** (e.g., predicting IC50 values)

5.) How does the model compare a node's embedding to others when predicting new links?

- After training, each node (e.g., a drug or gene) has a **learned embedding vector**.
- To predict links, the model **selects one node** (e.g., a drug) and **compares its embedding** with the embeddings of **all possible target nodes** (e.g., all genes in the graph).
- The model calculates a **similarity score** between the selected node's embedding and each candidate's embedding.
 - Methods include:
 - **Dot product:** $\mathbf{z}_i \cdot \mathbf{z}_j$
 - **Cosine similarity**
 - **MLP decoder** (learns a function on the pair of embeddings)
- **What does the score represent?**
 - The score reflects the **likelihood** of a link (e.g., drug–gene interaction).
 - It's often passed through a **sigmoid** to get a probability.
 - Nodes with **higher scores** are ranked as **more likely to interact**.

6.) How do GNNs differ from traditional neural networks like Multi-layer perceptrons? Why can GNNs capture relationships that MLPs can't?

- **MLPs (Multi-Layer Perceptrons)**
 - Work on **fixed-size, independent input vectors** (e.g., tabular or image pixels).
 - Each input sample is treated **in isolation**, with **no notion of connections** between samples.
 - They cannot model **structural relationships** between entities (e.g., drugs and genes).
- **GNNs (Graph Neural Networks)**
 - Designed to learn from **graph-structured data**, where **nodes are interconnected**.
 - Each node's embedding is influenced by its **neighbors and graph topology**.
 - Can model **complex interactions** between biological entities (e.g., drug-gene-pathway relationships).
- **Why GNNs are better for relational tasks**
 - GNNs can capture patterns like:
 - Functional similarity between genes
 - Shared drug mechanisms
 - Multi-hop biological associations (e.g., gene-pathway-disease)

7.) To what extent are GNNs similar to multi-layer perceptrons in traditional deep learning?

Similarities:

- Both consist of **layers of neurons** with **weights, biases**, and **activation functions**.
- Both are trained using **backpropagation** and **gradient descent**.
- Both aim to **minimize a loss function** over labeled data.

8.) What is inductive bias, and how is it embedded into GNNs through graph construction and architecture?

Inductive bias refers to the **assumptions a model makes about the structure of the data** to generalize well — especially when data is limited or noisy.

How GNNs embed inductive bias:

- Graph structure encodes prior biological knowledge
 - Nodes and edges reflect real-world relationships (e.g., drug-gene, protein-pathway).

- The GNN assumes this structure is **both meaningful and correct** — that the connections reflect **true relationships**, and that **connected nodes influence each other**.
- **Locality bias (a form of “guilt by association”)**
 - The model assumes that **nodes close together (e.g., 1-hop or 2-hop neighbors)** share similar properties.
 - For example, if a drug is linked to a gene that is associated with cancer, the drug may be inferred to have anticancer effects.
- **Weight sharing across the graph**
 - In most GNN architectures, a **shared transformation matrix** is used to update all node embeddings in a layer.
 - This reflects the assumption that **the same update rule can apply consistently across nodes**.
 - It simplifies the model, helping to **reduce overfitting** and encouraging **generalization**.
- **Message passing follows the graph topology**
 - Information is only passed between **connected nodes**, so the model assumes **structure defines information flow**.
 - If there's no edge between two nodes, the model assumes they shouldn't directly influence each other.
- **Attention mechanisms refine this bias**
 - In **GATs (Graph Attention Networks)**, the model learns which neighbors are **more important** than others.
 - This adds flexibility to the inductive bias, allowing the model to **focus on informative neighbors** and **downplay noisy or irrelevant ones**.
 - In **GCNs**, all neighbors contribute equally — strong, fixed inductive bias

9.) What is “guilt by association”, and how does it influence learning in biological graphs?

“Guilt by association” is the assumption that **nodes connected in a graph tend to be similar or share properties**.

In biology, this means:

- Genes that are connected (e.g., in a pathway) are likely to be **functionally related**
- A drug near a known toxic compound might also be **toxic**
- A protein interacting with a cancer gene might also be **cancer-related**

How does it affect GNN learning?

- GNNs rely on **message passing**, so each node **absorbs features from its neighbors**.
 - This means the model **implicitly assumes** that connected nodes contain **useful and relevant signals**.
 - If a node is surrounded by cancer-related genes, the model may learn that this node is also cancer-relevant.

Strengths	Risks
<ul style="list-style-type: none">● Helps the model generalize from sparse labels● Enables link prediction, classification, and clustering using local structure	<ul style="list-style-type: none">● If the graph is noisy or biased, the model may spread incorrect assumptions● Over-relying on neighborhood similarity can lead to false associations

10.) What are Graph Convolutional Networks (GCNs) and how do they work?

A Graph Convolutional Network (GCN) is a type of GNN that updates each node's embedding by **averaging its own features with those of its neighbors**, followed by a linear transformation and non-linear activation.

Each GCN layer perform the following steps:

1. Aggregate
 - Each node collects features from its **immediate neighbors** (1-hop).
 - It takes a **mean** (or normalized sum) of its neighbors' embeddings.
2. Combine
 - The aggregated vector is **linearly transformed** using a shared weight matrix W
 - A non-linear activation function (e.g., ReLU) is applied.
3. Update
 - The result becomes the **node's new embedding** for the next layer.

Key property: Uniform Neighbor weighting

- GCNs assume **all neighbors contribute equally** to the message.
- There is **no attention mechanism** or learned neighbor importance.

11.) What are Graph Attention Networks (GATs), and how do they improve on GCNs?

- GATs are a type of GNN that use **attention mechanisms** to assign **different importance to each neighbor** when updating a node's embedding.
- Unlike GCNs, which treat all neighbors equally, GATs **learn which neighbors matter more** based on the node features.
- This makes GATs better at:
 - Handling **noisy graphs**
 - Focusing on **biologically relevant connections**
 - **Making the model more flexible and expressive**

12.) How does attention differ in Multilayer Perceptrons vs GNNs?

- **In MLPs (Multi-Layer Perceptrons):**
 - Attention is usually applied to **features within a single input vector**.
 - For example, in a multi-modal model, attention might help the network **focus more on gene expression than on drug features**, if those are more informative.
 - The goal is to **weight features**, not neighbors.
- **In GNNs:**
 - Attention is used to **weight neighboring nodes** — deciding which connected nodes are more important when updating a node's embedding.
 - This allows the model to **prioritize relevant neighbors** and **downplay irrelevant or noisy connections** in the graph.
- **Summary:**
 - In MLPs, attention weights **input features**.
 - In GNNs, attention weights **neighbor nodes**.
 - Both serve to highlight the most useful information, but in different structures.

13.) What does it mean for message passing to be relation-type specific in multi-relational biological graphs?

What is a multi-relational graph?

- It's a graph where **edges can represent different types of biological relationships** — such as:
 - Drug **inhibits** a protein

- Gene **activates** another gene
 - Protein **interacts with** a pathway
- Each edge type encodes a **different kind of semantic meaning**.

What does relation-type specific message passing mean?

- The GNN treats each edge type **differently** during message aggregation, instead of mixing all neighbors the same way.
 - It may use **different weight matrices** or **scoring functions** per edge type.
 - This ensures the model **learns unique patterns for each relation type**.
 - Seen in:
 - **Relational GCNs (R-GCNs)**
 - **Relational GATs**
 - Some knowledge graph GNNs

Why is this important in biology?

- Relationships like “binds to” and “inhibits” are **not interchangeable**.
- Preserving edge semantics helps the model make **more accurate and biologically meaningful predictions**.

14.) What are multimodal GNNs, and how are separate modalities like gene expression and drug fingerprints handled?

What are multimodal GNNs?

- Multimodal GNNs are Graph Neural Networks that process and combine information from **different types of data (modalities)** — such as:
 - **Gene expression** (omics data)
 - **Chemical fingerprints** (molecular structure)
 - **Pathway graphs, protein-protein interactions**, etc.
- They allow the model to learn from **diverse biological signals** within a single framework.

How are separate modalities handled?

- Typically in three stages:
 1. **Separate encoders per modality**
 - Each input type (e.g., gene expression, drug fingerprints) is processed using its own neural branch to extract hidden representations.
 2. **Graph-aware integration**

- These embeddings are **projected into a shared graph structure**, where message passing happens.
- Nodes (e.g., drugs or genes) may have **feature vectors from multiple modalities**.

3. Fusion layer

- The outputs are merged using **attention-based fusion, concatenation, or shared embedding space** techniques.

Why is this useful in Biology?

- Biological systems are inherently **multimodal** — genes, chemicals, phenotypes, and interactions all provide unique insights.
- Multimodal GNNs help capture **complementary patterns** that single-modality models might miss.

15.) What are the three key stages in a GNN pipeline for drug discovery?

1. Graph construction (biological network creation)

- Build a graph where **nodes** represent biological entities (e.g., drugs, genes, proteins), and **edges** represent relationships (e.g., binds to, inhibits, interacts with).
- Can be built from public databases or literature.
- May be **multi-relational** (multiple edges) or **heterogeneous** (multiple nodes and edges).

2. Embedding learning (representation learning)

- Use GNN layers to learn a **vector embedding** for each node.
- Through **message passing**, each node updates its representation based on its neighbors.
- Embeddings encode structural and functional information relevant to prediction tasks.

3. Prediction (decoding stage)

- Use learned embeddings for **downstream tasks** like:
 - **Link prediction** (e.g., drug–target interaction)
 - **Node classification** (e.g., cancer vs non-cancer gene)
 - **Regression** (e.g., binding affinity score)

A **decoder** (e.g., dot product, MLP) compares embeddings to make final predictions.

16.) What is the decoder in a GNN, and how is it used for link prediction?

What is a decoder in a GNN?

- A **decoder** is the component that takes the **learned node embeddings** and **produces a final prediction** — such as whether two nodes are connected.
- It's the **final step** in the GNN pipeline, used for tasks like:
 - Link prediction
 - Node classification
 - Regression

How does it work in link prediction?

- After message passing, each node has a **vector embedding**.
- To predict a link between two nodes (e.g., drug–gene), the decoder:
 - Takes their embeddings as input
 - Measures their **compatibility** or **similarity**
 - Outputs a score or probability for the link

Types of decoders for link prediction:

- **Dot product:** Measures similarity between two node embeddings; fast and commonly used.
- **MLP (multi-layer perceptron):** Takes concatenated embeddings and learns interaction patterns.
- **Neural tensor network:** Models complex relationships using a trainable tensor; used in knowledge graphs.

17.) What are some key applications of GNNs in drug discovery and design?

- **Molecular property prediction:** Predict chemical traits like toxicity, solubility, or activity from a drug's molecular graph.
- **Drug–target interaction prediction:** Determine if a drug is likely to bind to a specific gene or protein.
- **Binding affinity estimation:** Predict how strongly a drug binds to its target (e.g., IC50 values).
- **Polypharmacy side-effect prediction:** Predict adverse effects from combining multiple drugs.

- **Protein interface prediction:** Identify regions where proteins or drugs are likely to interact.
- **Drug similarity analysis:** Cluster drugs based on structure or biological effect.
- **Knowledge graph completion:** Infer missing links in biological graphs (e.g., unknown interactions).

18.) How is drug sensitivity prediction different from drug-target binding affinity prediction?

- **Drug–target binding affinity** measures how strongly a drug binds to a **specific biological target**, such as a protein.
 - It is a **biochemical property** of a single drug–target pair, measured in isolated conditions (e.g., in vitro).
 - Common metrics: **K_d**, **K_i**, or **IC₅₀** (from target binding assays).
 - It tells us **how well the drug fits the target**, but **not how the cell or organism responds**.
- **Drug sensitivity** measures how effective a drug is at **killing or inhibiting a cell line** (e.g., cancer cell).
 - It reflects the **cell's overall response**, influenced by **gene expression, mutations, pathways**, and more.
 - IC₅₀ is also used here — but it represents the **drug concentration required to inhibit 50% of cell viability**, not target binding.

19.) What are the limitations of GNNs in biological systems?

1. **Data quality and noise**
 - GNNs rely on accurate biological graphs — but biological data can be **incomplete, noisy, or biased**.
 - Wrong or missing edges can lead to misleading predictions.
2. **Over-smoothing**
 - With too many GNN layers (i.e., large k -hop neighborhoods), node embeddings may become **too similar**, losing their unique identity.
3. **Scalability**
 - Large biological networks (e.g., full protein interaction graphs) can make GNNs **computationally expensive** and **slow to train**.
4. **Limited interpretability**

- Despite progress in explainable AI, GNNs still often act as **black boxes**, which is problematic in **clinical or regulatory** settings.

5. Static graphs

- Most GNNs assume the graph structure is **fixed**, but biological systems are **dynamic** — interactions can change across tissues, conditions, or time.

20.) Why are GNNs often preferred over CNNs and point-based methods for 3D biological modeling?

- **CNNs** require converting structures into 3D grids (**voxelization**), which can lose fine molecular details and is **computationally expensive**.
- **Point-based methods** (e.g., PointNet) use atom coordinates directly but **ignore molecular connectivity** — they lack awareness of bonds or interactions.
- **GNNs** preserve both the **3D structure** and the **relational graph** of atoms, making them ideal for capturing **biochemical interactions**.

IC50 Thresholding and Class Imbalance

IC50 and $\ln(\text{IC}_{50})$: Meaning and Modeling Use

- **IC50** (half-maximal inhibitory concentration) measures the drug concentration (in μM) required to inhibit **50% of cancer cell viability**.
 - **μM (micromolar)** = 10^{-6} moles per liter (mol/L)
- **Lower IC50 = higher potency** → the drug is effective at lower doses.
- IC50 values span wide ranges (e.g., 0.01 to 10,000 μM), so they are **converted to $\ln(\text{IC}_{50})$** (natural log) before modeling in order to:
 - **Compress the scale** — maps large numerical ranges into smaller, manageable ones
 - **Reduce skewness** — transforms the right-skewed distribution into a more symmetric one
 - **Stabilize model training** — avoids issues like exploding or vanishing gradients during learning
- **$\ln(\text{IC}_{50})$** is better suited for regression and classification models due to its normalized and interpretable range

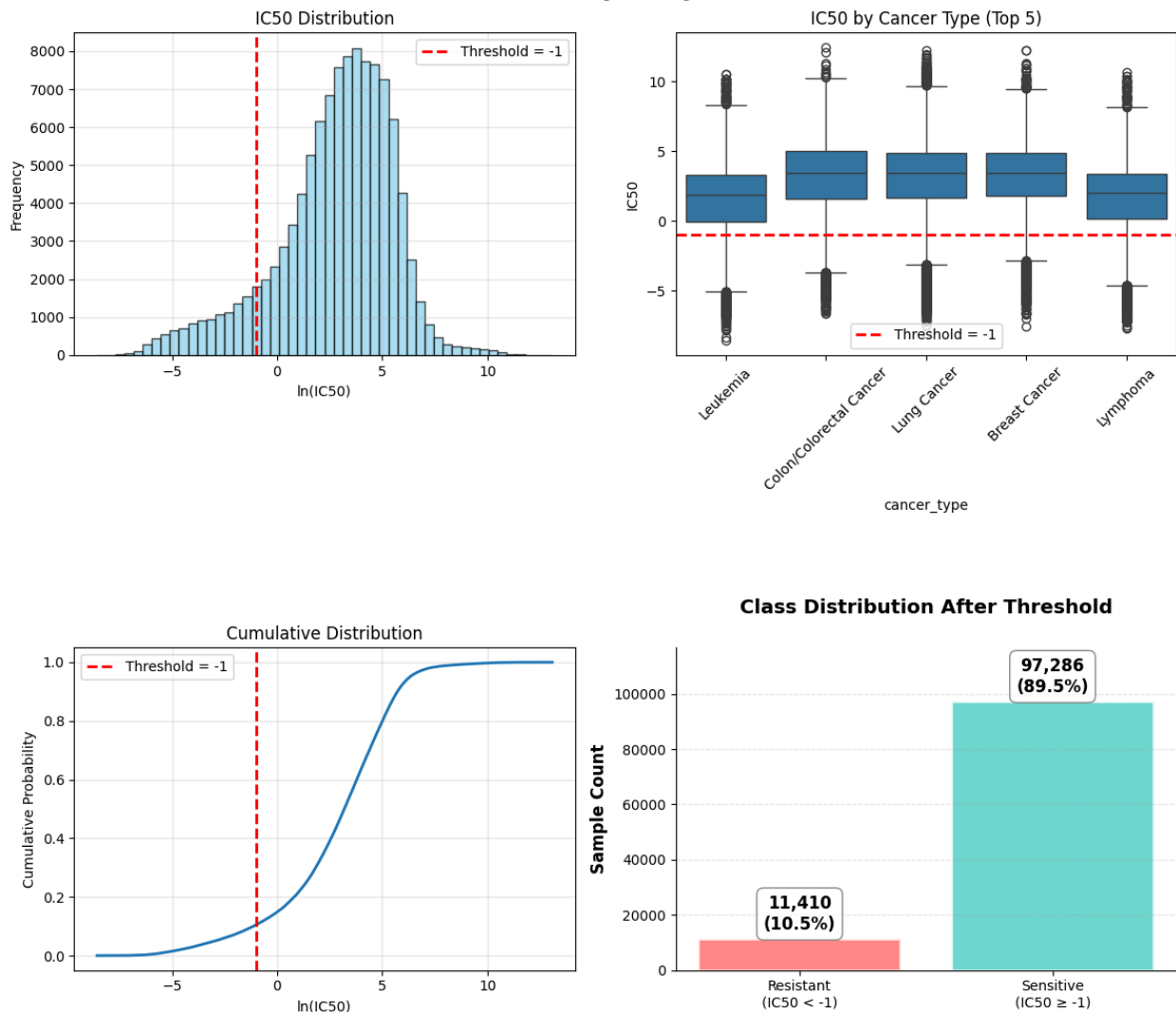
Classification Threshold: $\ln(\text{IC}_{50}) = -1$

- **Threshold chosen: $\ln(\text{IC}_{50}) = -1$** → corresponds to **$\text{IC}_{50} = 0.368 \mu\text{M}$**
- **Rationale:**
 - $\ln(0.368) \approx -1$
 - In pharmacology, **$\text{IC}_{50} < 1 \mu\text{M}$** is a widely accepted benchmark for **high-potency drugs**
- This threshold isolates **truly potent** drug–cell line responses
- Backed by literature and common practices in:
 - GDSC, ChEMBL, and large-scale drug screening studies
- A strict threshold prevents mislabeling **moderate or borderline activity** as "sensitive"

Class Distribution After Thresholding

- After thresholding:
 - **Sensitive:** $\ln(\text{IC}_{50}) \leq -1 \rightarrow 97,286$ samples (~**89.5%**)
 - **Resistant:** $\ln(\text{IC}_{50}) > -1 \rightarrow 11,410$ samples (~**10.5%**)

IC50 Threshold Analysis & Justification



Biological and Statistical Interpretation

- $\ln(\text{IC}_{50}) = -1$ lies at the **10.5th percentile** of the overall IC50 distribution
 - Only the most potent 10.5% of drug–cell line interactions are labeled "sensitive"
- This reflects **natural biological imbalance**:
 - Most drug–cell line combinations are **not highly sensitive**
 - This is expected in drug discovery — most compounds fail to strongly inhibit most cell types
- While balanced data is preferable for modeling, this class imbalance is **biologically realistic**
- **Boxplots confirm**:
 - $\ln(\text{IC}_{50}) = -1$ falls **below the interquartile range** across the top-5 cancer types
 - Threshold is **consistently valid** across different malignancies

- **Leukemia** cell lines show **greater drug sensitivity** than solid tumors
 - Matches real-world trends — hematologic cancers often respond better to drugs

Summary of CCLE_expression.ipynb: Preprocessing CCLE Gene Expression Data

This notebook processes the raw gene expression matrix from the Cancer Cell Line Encyclopedia (CCLE) to prepare a clean and biologically relevant dataset for use in downstream drug response modeling (e.g., in `01_data_prep.ipynb`).

1. Load Full Gene Expression Matrix

- File loaded: `CCLE_expression.csv`
- Initial shape: **1406 rows × 19221 columns**
 - Rows = cell lines (`DepMap_ID`)
 - Columns = gene expression features, labeled as "`GENE_NAME (Entrez ID)`"

2. Clean Gene Names

- Removes the Entrez ID part from column names (e.g., "`TP53 (7157)`" → "`TP53`")
- Leaves only clean gene symbols in the column headers

3. Filter Genes Using COSMIC Census

- Loads `CancerGeneCensus_GRCh38_COSMIC_v101.csv`, which lists genes known to be involved in cancer
- Filters the gene expression matrix to keep **only genes present in the COSMIC census**
- Reduces the number of features from **19221 genes** → **753 cancer-related genes**
- This step focuses your model inputs on **biologically meaningful genes** relevant to cancer and drug response.

4. Final Output

- `CCLE_expression_cleaned.csv`
- 1406 cell lines × 753 cancer genes (expression values)

What Are Cell Lines vs Gene Expression Features?

- **Cell lines = rows**
 - Each row represents a unique biological sample (cancer cell line) identified by `DepMap_ID`.
- **Gene expression features = columns**

- Each column represents the expression level of a specific gene in each cell line.

Why Keep All Cell Lines but Reduce Gene Features?

- **Rows = samples** (your training data) → keep all to preserve biological diversity
- **Columns = features** (input to model) → reduce to keep only cancer-relevant genes based on COSMIC

Summary of 01_data_prep.ipynb: Preparing Drug Sensitivity and Gene Expression Data

This notebook builds a clean and fully aligned dataset linking drug sensitivity measurements from the **Genomics of Drug Sensitivity in Cancer (GDSC)** with gene expression profiles from the **Cancer Cell Line Encyclopedia (CCLE)**. The output consists of three high-quality CSV files used later for machine learning.

1. Load and Clean GDSC Drug Sensitivity Data

- Load GDSC1 and GDSC2 datasets.
- **Filter for valid IC50 values** using `np.isfinite(...)` — removing rows with NaN or infinite values.
- Drop evaluation-related columns (`AUC`, `RMSE`, `Z_SCORE`, etc.) that are not needed.
- When duplicate entries exist (same `CELL_NAME`, `DRUG_NAME`, `COSMIC_ID`), retain the entry from **GDSC2** for consistency.
 - `DRUG_NAME`: drug tested
 - `COSMIC_ID`: internal identifier for the cell line
 - `IC50`: drug sensitivity measurement

2. Map `COSMIC_ID` to `DepMap_ID` to Align with CCLE

- Load `sample_info.csv` from CCLE, which maps each `COSMIC_ID` to its corresponding `DepMap_ID`.
- Replace all GDSC cell line identifiers (`COSMIC_ID`) with standardized `DepMap_ID`.
- Rename this column as `cell_line` for clarity and consistency.

3. Filter Valid Drugs with SMILES Strings

- Load a curated drug information table (`processed_drug_df`) that includes:
 - `DRUG_NAME`: cleaned, standardized drug names
 - `SMILES`: molecular structure in text format

- Cleaned SMILES: with salts removed
- Keep only drugs that have valid SMILES representations.

4. Filter and Align Gene Expression Data

- Load gene expression data from `CCL_E_expression.csv`, indexed by `DepMap_ID`.
- Keep only those cell lines that are present in the filtered GDSC dataset.
- Ensure that the gene expression matrix (`GeneExp`) and the drug sensitivity entries now refer to the **same set of cell lines**.

5. Match Only Shared Cell Lines

- Determine the **intersection** of cell lines that appear in both:
 - The gene expression matrix
 - The drug sensitivity table
- Filter both datasets to keep only these shared `DepMap_IDs` (`MultiOmics_CellLine`).

6. Pivot Drug Sensitivity Data (Matrix Form)

- Use `.pivot(index='gdsc_name', columns='cell_line', values='IC50')` to create a matrix:
 - Rows = drugs
 - Columns = cell lines
 - Values = IC50 values
- Each entry represents the sensitivity of a cell line to a drug.

7. Flatten Matrix and Annotate with Metadata

- Use `.stack()` and `.reset_index()` to convert the matrix back to long format:
- One row per (drug, cell line) pair
- Add:
 - `SMILES`: via mapping from drug names
 - `cancer_type`: via mapping from cell line metadata

Final Outputs and Their Purpose

File	Description	Used For
<code>binary_IC50_table.csv</code>	Final long-format table: (<code>cell_line</code> , <code>drug</code> , <code>IC50</code> , <code>SMILES</code> , <code>cancer_type</code>)	Target variable + drug/cell info
<code>binary_gene_expression.csv</code>	Expression matrix: <code>DepMap_ID</code> × <code>735</code> <code>genes</code>	Input features (cell lines)
<code>binary_drug_info.csv</code>	Metadata about drugs with SMILES strings	Input features (drugs)

Summary of Data Integration & Preparation.ipynb: **Data Integration and Preparation**

Dataset Integration Overview

The notebook prepares a supervised learning dataset by combining three core data sources:

1. Drug Sensitivity Measurements (GDSC)

- Columns: `cell_line`, `drug_name`, `gdsc_name`, `ln(IC50)`
- Each row represents the experimentally measured drug response for a specific drug–cell line pair.

2. Gene Expression Profiles (CCLE)

- Rows: individual **cancer cell lines** (i.e., tumors)
- Columns: **735 genes**
- **Each entry** is a **pre-normalized continuous value** representing the expression level of a gene in a particular cell line.
 - These values are likely log-transformed and scaled by the data provider (e.g., log2 TPM or Z-scores) to correct for sequencing depth and variability across genes and cell lines.
 - Normalization ensures that genes with inherently high expression (e.g., housekeeping genes) do not dominate learning, and makes gene expression values more comparable across samples.

3. Drug Chemical Descriptors (SMILES → Morgan Fingerprints)

- Columns: `drug_name`, `SMILES`
- SMILES strings are converted into **2048-bit Morgan fingerprints** using RDKit, encoding presence (1) or absence (0) of chemical substructures.
- These fingerprints are **sparse** (~97.3% zeros), as each molecule only contains a small subset of possible substructures.

- This sparse binary format enables similarity comparisons between drugs using **Tanimoto coefficients**, which are especially suited for binary vectors.
- An **inner join** is performed between GDSC and CCLE on the **cell_line** field to retain only **676 overlapping cell lines**.
- A mapping with SMILES is then done via **gdsc_name** to align 228 unique drugs. However, **not all combinations were tested experimentally**.
- The final result contains **108,696 valid samples** (rows) — each one being a tested drug–cell line pair with a known IC50 value.
 - This number is **less than $676 \times 228 = 154,128$** because the dataset **does not contain the full Cartesian product** — only actual IC50 measurements are retained.

Label Assignment Using IC50

The **$\ln(\text{IC}_{50})$** values are converted into binary labels for classification:

- **Sensitive (label = 1):** $\ln(\text{IC}_{50}) < -1 \rightarrow \text{IC}_{50} < 0.368 \mu\text{M}$
- **Resistant (label = 0):** $\ln(\text{IC}_{50}) \geq -1 \rightarrow \text{IC}_{50} \geq 0.368 \mu\text{M}$

This biologically informed threshold is supported by GDSC conventions and literature. The resulting label distribution:

- **Sensitive:** ~89.5% (97,286 samples)
- **Resistant:** ~10.5% (11,410 samples)

This class imbalance is preserved in downstream splits, making it crucial to handle properly during training and evaluation.

Feature Construction and Structure

Each sample is encoded using two modalities:

- **Gene Expression (735 continuous features):**
 - Derived from the CCLE matrix — represents the molecular state of the cell line.
- **Chemical Fingerprints (2048 binary features):**
 - Generated from SMILES using Morgan fingerprinting (radius = 2) — captures the structural properties of the drug compound.

These are horizontally concatenated to yield a feature matrix $X \in \mathbb{R}^{108696 \times 2783}$. The binary labels are stored in a separate vector $y \in \{0, 1\}^{108696}$.

Keeping `x` and `y` separate is essential for proper supervised learning workflows (especially in deep learning), where `x` is used to compute forward passes and `y` is used for loss evaluation — but not for gradient updates.

Standardization of Features

- Gene expression features are standardized using **z-score normalization**:

$$z = \frac{x - \mu}{\sigma}$$

This ensures zero mean and unit variance across each gene feature, promoting numerical stability and fair contribution.

- Although fingerprints are binary, they are also standardized using `StandardScaler` to allow unified processing by ML models that assume normally distributed features.

Data Quality Checks

1. **Missing Values:** None detected in either modality.
2. **Feature Distributions:** Means and standard deviations verified using `.describe()`.
3. **Label Imbalance:** Confirmed 89.5% sensitive / 10.5% resistant.
4. **Outlier Detection via Z-score:**
 - Z-score is used to detect outliers in gene expression.
 - Points with $|z| > 3$ are considered outliers, but no significant contamination was found.
5. **Modality Alignment:** Confirmed 735 gene + 2048 fingerprint = 2783 columns.
6. **Sample Representation:** Sufficient sample sizes across major cancer types.

Stratified Data Splitting

To preserve class distribution across splits, **stratified sampling** is used. This ensures that each subset maintains the 89.5% / 10.5% class ratio.

- Final distribution:
 - 60% training
 - 20% validation
 - 20% testing

Once the dataset is split, each subset serves a **distinct and non-interchangeable purpose** in the learning pipeline:

- **Training Set (60%)**
 - Used to **fit the model parameters** — this is where learning happens.
 - During each epoch, the model sees these samples and adjusts its weights to **minimize the loss function**.
 - Backpropagation is applied here:
 - **Forward pass** computes predicted labels.
 - **Loss** (e.g., binary cross-entropy) is computed between predictions and true labels.
 - **Gradients** of the loss w.r.t. model weights are computed via **backpropagation**.
 - **Optimizer** (e.g., Adam) updates weights to reduce the loss.
- **Validation Set (20%)**
 - Not used in backpropagation — weights are **never updated** from validation loss.
 - After each training epoch, the model is evaluated on the validation set to:
 - Monitor generalization performance
 - Detect **overfitting** (if validation loss starts increasing while training loss decreases)
 - Guide **early stopping** and **hyperparameter tuning**
 - Helps answer: “Is the model learning features that generalize?”
- **Test Set (20%)**
 - Used for final evaluation only — provides an unbiased estimate of real-world performance on unseen data.
 - No training or tuning decisions are allowed to touch the test set.
 - Metric scores on this set (e.g., accuracy, AUROC, F1) reflect how well the model might perform on unseen data. Ok this improved the notes for your test set section.

PCA and EDA (Exploratory Data Analysis)

In this project, PCA is used **only for visualization**, not for feature reduction or model training.

- The input feature space (gene expression + fingerprint) is 2,783-dimensional.

- PCA projects this data down to **2D**, using the first two principal components: **PC1** and **PC2**.
- This lets us **visually explore structure** in the data — e.g., whether sensitive and resistant samples occupy distinct regions.

What do PC1 and PC2 actually mean?

- **PC1 (Principal Component 1):**
 - The axis that captures the **maximum variance** in the entire dataset.
 - It's a **linear combination** of the original features:

$$\text{PC1} = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$$

where x_i are the original features, and w_i are learned weights.

- **PC2** is orthogonal (perpendicular) to PC1 and captures the **next highest variance** not already captured by PC1.

Why are PC1 and PC2 Used in Visualization?

When plotting the dataset in (PC1, PC2) space:



- **Clusters** suggest that the original high-dimensional features encode meaningful patterns.
- **Linear Separability!**
 - **Separation between classes** (e.g., resistant vs sensitive) suggests that a **linear decision boundary might exist**.
- **Heavy overlap** implies weak signal or need for non-linear models (Deep-learning approaches).
- **Outliers** are easier to spot — points lying far from all others.



Pearson Correlation in EDA

- **Pearson correlation coefficient** was computed between individual features (genes or fingerprint bits) and the IC50-derived binary label (sensitive vs. resistant).
- The goal was to **identify features linearly associated with drug sensitivity**.
- Pearson's formula:

$$r = \frac{\text{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y}$$

- Where X is the feature vector and Y is the binary target
- Covariance in the numerator measures how X and Y vary together
- Denominator scales it by their standard deviations
- A high $|r|$ indicates strong linear correlation — useful for ranking features before modeling.
-  While MLPs do not assume linearity, this analysis helps **spot informative patterns** and guides **feature interpretation**, not model training. 

Summary of Model Development & Training.ipynb: Model Development and Training

Model Architecture

The model predicts whether a cancer cell line is **sensitive or resistant** to a given drug using two modalities:

- **Gene expression features** (cell-line biology, 735 dimensions)
- **Chemical fingerprints** (molecular structure, 2048 bits)

These two feature types are handled via **parallel processing branches**, followed by **cross-modal attention** and **feature fusion**.

Dual-Branch Feature Extraction:

1. Gene Expression Branch

- Input: 735 features
- Dense layers: $735 \rightarrow 1024 \rightarrow 512 \rightarrow 256 \rightarrow 128$
- Each layer followed by **Dropout** to reduce overfitting

2. Chemical Fingerprint Branch

- Input: 2048 features
- Dense layers: $2048 \rightarrow 1024 \rightarrow 512 \rightarrow 256 \rightarrow 128$
- Also includes **Dropout layers**, but dropout rate is slightly higher due to data sparsity (binary bits)

Cross-Modal Attention (Core Innovation):

- **Gene-to-Chemical Attention:**
 - Gene features attend to chemical features — models **how cellular states respond to drug properties**
- **Chemical-to-Gene Attention:**
 - Chemical features attend to gene profiles — models **how drug structure might influence expression**
- **The resulting attended vectors:**
 - `gene_attended_flattened` → (None, 128)
 - `chem_attended_flattened` → (None, 128)

Fusion Layer (4-Way Feature Fusion):

The model combines:

1. Gene representation (128)
2. Chemical representation (128)
3. Gene-to-chemical attended vector (128)
4. Chemical-to-gene attended vector (128)

→ Fused feature vector: 512-dimensional

This fusion preserves independent modality learning and cross-modal interactions, ensuring a richer representation.

Classification Head:

- Fully connected layers: $512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 1$
- Output: A **sigmoid-activated scalar** (probability of resistance)

Training Strategy

Design Philosophy:

- Leverage **NVIDIA A100 GPU** with:
 - **Large batch size** (512)
 - **Mixed precision training (FP16)** for faster throughput
 - **GPU memory growth** enabled



Loss Function: Weighted Binary Crossentropy

- Class imbalance: ~89.5% sensitive, 10.5% resistant
- Resistant class is underrepresented, so its loss contribution is **upweighted by 8.53x**

$$\text{Loss} = -[w_1 \cdot y \cdot \log(\hat{y}) + w_0 \cdot (1 - y) \cdot \log(1 - \hat{y})]$$

- This ensures the model is **penalized more for missing resistant cases**

Dropout continues to be applied throughout, with **decreasing rates** as layers get deeper — this reflects a strategic regularization choice (fewer units, so fewer need to be dropped).


Optimizer: AdamW

- Combines Adam with **L2 weight decay** (0.01) to penalize large weights → encourages generalization
- Learning rate: 0.001 (fixed)
- **Gradient clipping** (norm capped at 1.0) prevents gradient explosion from mixed precision

Early Stopping + Model Checkpoint:

- **Monitored Metric:** `val_auroc`
- **Best model saved at epoch 14** (where val_auroc peaked)
- Training halted at epoch 29 after 15 epochs of stagnant validation performance
- Final evaluation uses the **model from epoch 14**, not the last epoch

Evaluation Metrics

Metric	Formula / Meaning	Test Result	Interpretation
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$ — total correct predictions	96.3%	High, but not imbalance-aware
Precision	$\frac{TP}{TP+FP}$ — how often predicted resistant cases were correct	97.3%	Excellent, few false alarms
Recall (Sensitivity)	$\frac{TP}{TP+FN}$ — how many true resistant cases were caught	98.5%	Nearly all resistant cases were identified
Specificity	$\frac{TN}{TN+FP}$ — how many true sensitive cases were correctly predicted	77.0%	Some false positives remain
F1 Score	$2 \cdot \frac{P \cdot R}{P+R}$ — harmonic mean of precision and recall 	97.9%	Strong balance between under- and over-detection
Balanced Accuracy	$\frac{\text{Recall} + \text{Specificity}}{2}$ — fairness across classes	87.7%	Model treats both classes more equally
AUROC	$P(\hat{y}_{\text{resistant}} > \hat{y}_{\text{sensitive}})$ — ranking ability	98.1%	Model ranks resistant samples higher almost every time
AUPRC	Area under PR curve (focuses on resistant class)	99.8%	Excellent performance in retrieving resistant cases

Interpretation Notes:

- **Resistant** class was labeled as **1** (positive class) to reflect clinical relevance.
- AUROC measures **how well the model ranks** resistant samples **above** sensitive ones based on predicted probability — not just hard classification.
- AUPRC is **especially valuable** in imbalanced datasets where AUROC alone may overestimate performance.

Limitations & Interpretability

- While attention layers offer theoretical interpretability (via attention scores), the model remains largely a **black box**.
- The exact biological relationships (e.g., which gene responds to which compound) are not explicitly revealed.
- Cross-modal attention helps **simulate** interaction, but it doesn't guarantee biological explainability.
- This limitation is common in deep learning models in biomedicine — and is driving interest in **GNNs and symbolic models**.