Deep Learning Best Practices (2025 update)

Author List: Jake Gagnon, Hangyu (Cedric) Liu, Lira Pi, Phoebe Jiang,
Lukasz Kniola, Himanshu Pandya, and Jeff Thompson

[1]

[1] Cover image created with DALL·E 3 by OpenAI

Table of Contents

Introduction/Scope

Deep Learning methods have been quite popular in recent years in areas such as Computer Vision, Natural Language Processing, and Generative AI. In this white paper, we hope to guide deep learning practitioners in industry with best practices and introduce the core concepts and architectures of deep learning. The manuscript is organized as follows: we begin by providing recommendations for exploratory data analysis, data pre-processing, and data governance. After these preliminaries, we review data splitting/cross validation techniques to reduce overfitting of the data and describe common neural network architectures. We then discuss common techniques for model evaluation and data augmentation. Since pretrained models may not have acceptable performance for your specific domain of interest, we discuss various techniques to improve model performance such as transfer learning or fine-tuning. Lastly, we discuss other deep learning pipeline considerations such as model interpretation, model reproducibility, and deployment of models into production.

Chapter 1: Exploratory Data Analysis

Assess Data Bias

There are multiple types of biases which can occur during the data gathering process. Selection bias occurs when the sample is not randomly selected from the population but influenced by external factors such as convenience, availability, and preference. A non-response bias occurs when the sample is randomly selected, but some of the participants do not respond or drop out of the study, creating a critical difference between the respondents and the non-respondents. A third type of bias is measurement (technical) bias. This occurs when the data is collected or recorded in a way that introduces errors or inaccuracies, such as using faulty instruments, ambiguous questions, or subjective judgments. Another type of bias that should be avoided is survivorship bias. This type of bias occurs when the sample only includes the survivors or the winners of a process, ignoring those who failed or dropped out. Lastly, we have a confirmation bias which occurs when the analyst interprets or presents the data in a way that confirms their pre-existing beliefs or expectations, rather than being objective and open-minded. If any of the above biases are expected to be present in your data, correction of the biases is highly recommended prior to model training.

Assess Representativeness

Check that your gathered data is representative of your desired target population. This includes checks such as demographic tables to validate that your gathered data represents the demographic composition (gender, race, ethnicity) of your target population.

Assess Data Balance

It's very crucial and of paramount importance to have balanced data during data analysis and machine learning. Balanced data means you have the same number of samples/individuals for each group in a classification dataset. Using imbalanced data when training a classifier can cause a bias in favor of the majority class and can lead to incorrect predictions, so having a balanced dataset is highly recommended. See "Imbalanced data strategies" section below for potential solutions.

Assess FAIR Principles

FAIR data are data which meet principles of findability, accessibility, interoperability, and reusability (Wilkonson et al. 2016). The FAIR principles emphasize machine-actionability (i.e., the capacity of computational systems to find, access, interoperate, and reuse data with no or minimal human intervention) because humans increasingly rely on computational support to process/analyze due to the increase in volume, complexity, and the computational time to analyze data. The FAIR Data Principles are a set of guiding principles proposed by a consortium of scientists and organizations to support the reusability of digital assets. computational time to analyze data. The FAIR Data Principles are a set of guiding principles proposed by a consortium of scientists and organizations to support the reusability of digital assets.

Assess Batch Effects

Another important aspect of data exploration is the assessment of batch effects. "Batch" effects include the effects of sample run, time of day, day of the week, experimenter, reagent batch, location, etc. To assess your data for batch effects, visual displays such as PCA, density plots, boxplots, RLE plots, or heatmaps can be constructed and interpreted. For example, in PCA, a dataset with a batch effect will show a clear separation of batches in PCA space. On the other hand, if the batches are well-mixed in PCA space then the batch effect is minimal. If a significant batch effect is detected, then the analyst should consider batch effect correction methods such as 1) regressing out a batch covariate in a linear regression or linear mixed model, 2) surrogate variables analysis (sva) which is a common method in genomics, 3) ComBat, 4) batch mean centering, or 5) SVD based approaches.
(See https://evayiwenwang.github.io/Managing_batch_effects/index.html#study-description for more details)

References

1. Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3**, 160018 (2016).

Chapter 2: Data preprocessing

Database Normalization

Database normalization is the process of reorganizing data within a database so that users can utilize it for further queries and analysis. It is one of the popular processes for developing clean data, which includes eliminating redundant and unstructured data and making the data appear similar across all records and fields. The main objective of database normalization is to eliminate redundant data, minimize data modification errors, and simplify the query process for better data query/access.

Ultimately, normalization goes beyond simply standardizing data, and can even improve your workflow, increase security, and lessen costs. When data is normalized, the entire database is more logically structured, leading to an ease of understanding and use. This uniformity ensures that teams can collaborate more effectively, as everyone is working with the same consistent information. Also, by eliminating redundant data, the size of files is reduced, which speeds up both analysis and data processing. This efficiency is a boon for cost-saving, as it means quicker access to data and reduced requirements for extensive storage and processing capacities. Moreover, a well-organized database is inherently more secure, as data is precisely located and uniformly maintained, adding an extra layer of protection against security breaches. Overall, data normalization isn't just a cleanup task; it's an essential step towards making databases more functional, secure, and cost-effective.

Statistical Normalization

Data normalization, or rescaling of the data to a specified range, is an essential step to biologically interpret your data. For example, in genomics, the total RNA content can vary from cell to cell. Consequently, to enable fair comparisons across cells, we need to adjust (normalize) each cell’s gene expression with respect to its total RNA content. For a second example, to enable fair sample to sample comparisons in qPCR gene expression measurements, a reference gene (control) is utilized to normalize the gene expressions of each sample. In general statistics, normalization is typically a min-max re-scale of values between 0 and 1.

Data Scaling

Data scaling is the process of transforming the values of the features of a dataset until they are within a specific range, i.e. 0 to 1 or -1 to 1. One common technique of standardization is z-scaling (i.e. centering and scaling) to a standard normal distribution of mean 0 and standard deviation 1. This scaling is to ensure that no single feature dominates and can help improve the performance of an ML algorithm. As another example, in lasso regression, scaling allows us to fairly interpret the effects of each feature and aids in feature selection. However, some ML methods are not sensitive to the magnitude of data (e.g. tree-based algorithms) so standardization is not needed for them.

Data Cleaning

Data cleaning is important because it ensures data quality. This can prevent errors, improve decision-making, and increase productivity. Data cleaning techniques are essential to getting accurate results when you analyze data for various purposes.

There are many methods that can be used to improve data quality. The following are some common and recommended approaches: removing duplicates, removing irrelevant data, converting data types, and handling outliers. It is important to remove duplicates to maintain data quality. Duplicates not only represent a data storage problem but can also affect the accuracy and processing time of the model. Removing duplicates sometimes involves merging several partial records together, while other times it involves deleting the duplicate records.

Another important method is to remove irrelevant data. Similarly, duplicate records (irrelevant data will increase the storage demands for your data) will increase processing costs, and potentially decrease output accuracy.

Data type conversion can be needed when the source of the data differs from how it is stored. When this method is used properly, it ensures all data will be in an easily storable format, can be easily processed, and maintains the data with minimal or no loss of information.

Lastly, handling outliers is an important method to maintain data quality. Depending on the robustness of your model, outliers can significantly bias the model results. Outliers often indicate data source issues such as data errors, a lack of a representative sample, or an exceptional data point. They can also indicate an issue with the current usage of the data. For example, it is common to apply a logarithmic scale to financial data; however, if no transformation was applied, the data quality issue could show up as an outlier.

Missingness/Imputation

The data missingness proportion should be assessed for each feature, for each subject, as well as the response variable. Additionally, ML practitioners should determine the missing data mechanism. If data missingness is observed, it can be addressed using a variety of different techniques. The following techniques are some recommendations for data imputation of missing data:

Mean/median

Imputation using the mean or median value is a straightforward approach that replaces all occurrences of missing values (NA) for a feature within a dataset by using the mean (if the variable has a Gaussian distribution) or the median (if the variable has a skewed distribution).

Multivariate Imputation by Chained Equations (MICE)

MICE performs imputation by creating several sets of data, each with imputed values based on specified prediction models for that feature. The imputation sets are chained, meaning that each subsequent dataset uses the prediction model from the previous one to obtain new values for each feature that contains missingness. The missing values are then updated across the full dataset based on the new fit in each iteration. There is a small amount of randomness introduced in the iterated predictions to attempt to avoid bias during the chaining procedure. The MICE procedure ends when the variable predictions have converged, and the set of MICE-generated data can then be pooled, including the imputed values. The R package mice can be used to perform MICE imputation and has documentation available on its usage. For more details, the package publication and example R codes can be found from https://www.jstatsoft.org/article/view/v045i03. There are also several Python implementations of MICE, such as scikit-learn IterativeImputer (see https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html for vignettes and example codes). The MICE approach can be used together with other techniques to optimize imputed values (for example, see Random Forest imputation).

K-nearest neighbor (KNN)

KNN imputation employs the k-nearest neighbor algorithm in multi-dimensional space to impute missing values for a feature. Using the dataset present, the KNN approach will select the closest neighboring points using an arbitrary distance metric, which is typically the Euclidian distance. For a data point with missingness in a feature, KNN imputation will then assign an imputed value calculated from its neighbors for that feature. The assigned imputed value for that datapoint can be calculated using any arbitrary metric such as the mean, distance-weighted mean, or the mode. The optimization of the number of neighbors (k value) is important to avoid overfitting (low k) or noise (high k) and often requires tuning the value of k using cross-validation or other k selection strategies. KNN imputation is non-parametric and can be sensitive to outliers in the data and will tend not to provide reliable imputation values in generally sparse datasets or in datasets with regional sparsity. KNN imputation using the average value of the nearest k neighbors to impute missing features can be performed using several different R packages, such as caret, which has documentation on how to perform this method, utilizing caret::preProcess(..., method = “knnImpute”,...) function. For more details, see the documentation of CRAN webpage, https://cran.r-project.org/web/packages/caret/index.html. For the Python implementation, there are again several options such as scikit-learn function KNNImputer (find more details from https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html#sklearn.impute.KNNImputer), which can be used to perform KNN imputation.

Random Forest (RF)

Random Forest imputation uses a random sampling approach to create a set of decision trees to create the RF from existing data. Imputation is performed for the missing data within this RF by averaging the values for that feature. RF imputation does not require a specific parametric model, nor does it require an assumption of a normal distribution; however, this approach can still be sensitive to highly skewed distributions. There are R packages that can perform random forest imputation, such as missForest and CALIBERrfimpute, which have documentation available for their usage. The missForest package is a popular choice for performing RF imputation using RF alone (for more details, see https://cran.r-project.org/web/packages/missForest/missForest.pdf). On the other hand, CALIBERrfimpute uses MICE-imputed datasets in concert with RF to optimize imputed values (for example codes, visit https://cran.r-project.org/web/packages/CALIBERrfimpute/CALIBERrfimpute.pdf). In Python, there are several implementations that can be used for RF imputation, such as the missingpy MissForest function (find example codes from https://pypi.org/project/missingpy/), and miceforest represents a Python package that can use MICE in concert with RF for imputation. See https://pypi.org/project/miceforest/ for more details on the package.

Choosing an imputation strategy

Deciding on which imputation strategy to employ will be highly dependent on a user's needs. Factors to consider include the data size format and sparsity, the computational resources available, and the missing data mechanism. A table is displayed below which may help in the decision-making process.

Factor	Random Forests	KNN	Median	MICE
Data Features				
- Size	Large (millions of rows)	Large (millions of rows)	Small to Medium (10-100k rows)	Large (millions of rows)
- Type	Mixed (categorical and numerical)	Mixed (categorical and numerical)	Numerical	Mixed (categorical and numerical)
- Sparsity	Less sensitive, can learn from similar patterns but accuracy may decrease	Sensitive, performance worsens significantly with higher sparsity, may overfit	Not recommended (assumes underlying distribution)	Moderately sensitive, requires careful parameter tuning and model selection
- Outliers	Moderately robust, internal trees can downweight outliers in decision making	Sensitive, distance-based approach can be misled by outliers, consider outlier detection beforehand.	Not recommended (can amplify outliers)	Moderately robust, multiple imputations can capture some outlying values, consider robust model selection within MICE.
- Linear relationships	Moderately preserves	Generally, preserves for nearby neighbors	Not recommended	Moderately preserves through multiple imputations
- Non-linear relationships	Less effective, limited ability to capture complex interactions	Not effective, distance-based approach struggles with non-linearity	Not recommended	Can be more effective with appropriate model choices within MICE
Missing Value Type				
- Missing Completely At Random (MCAR)	Performs well in general	Performs well in general	Not recommended	Performs well in general
- Missing At Random (MAR)	Can perform well, may exhibit slight bias	Can perform well, consider advanced distance metrics	Not recommended	Moderately robust, careful model selection needed
- Missing Not At Random (MNAR)	Not recommended, may amplify biases	Not recommended, can be misled by missingness patterns	Not recommended	Limited effectiveness; requires additional techniques to

				address underlying mechanism
Model requirement	No (internal trees)	No (distance-based)	No	Yes
Ease of use	Medium	Medium	Easy	Complex
Implementation language	R (MissForest is robust and well-regarded), Python (miceforest is efficient)	Python or R (both have efficient implementations)	Python or R (both have simple implementations)	R (more mature packages and expertise)
Computational resources	High	Medium	Low	High
Scalability	Highly scalable due to parallelization options	Moderately scalable, efficient implementations available	Not recommended for large datasets	Highly scalable with parallelization and efficient mode choices

Table 1: Choosing an imputation strategy

References

1. https://medium.com/@Cambridge_Spark/tutorial-introduction-to-missing-data-imputation-4912b51c34eb.

2. Li, J., Guo, S., Ma, R. et al. Comparison of the effects of imputation methods for missing data in predictive modelling of cohort study datasets. BMC Med Res Methodol 24, 41 (2024). https://doi.org/10.1186/s12874-024-02173-x

3. Hong, S., Lynn, H.S. Accuracy of random-forest-based imputation of missing data in the presence of non-normality, non-linearity, and interaction. BMC Med Res Methodol 20, 199 (2020). https://doi.org/10.1186/s12874-020-01080-1

4. https://www.analyticsvidhya.com/blog/2022/05/handling-missing-values-with-random-forest/

5. https://towardsdatascience.com/missforest-the-best-missing-data-imputation-algorithm-4d01182aed3

6. https://cran.r-project.org/web/packages/finalfit/vignettes/missing.html

Considerations for Image Preprocessing

Prior to training a deep learning model on images, pre-processing of images is required. The exact pre-processing steps needed will depend on your problem domain. As a first step, inspect the input size of your neural network. If your image size doesn’t match your input size, then your training data needs to be cropped or resized to match your input requirements. You should also inspect the training data for any illumination problems. For example, if the illumination is not constant across the image, then illumination correction will be needed. Additionally, you may want to consider conversion of your images to grayscale. In some computer vision tasks, color information may not be needed so a conversion to grayscale will make your deep learning model more computationally efficient. Other pre-processing steps such as whitening, gamma correction, histogram equalization, or removal of redundant images may be needed depending on your problem domain.

Another consideration in image processing is the stability and robustness of training the neural network. To improve stability, it is common to normalize your images with techniques such as mean image subtraction, per-channel normalization, or per-channel mean subtraction. As for robust model training, data augmentation is a common method to increase training data size and model robustness [See Data Augmentation in CV subsection below]. This includes operations such as flipping, rotating, adding noise, changing contrast, or varying exposure/brightness.

Additionally, model performance will be improved with high quality training images. The images should be in focus and in a variety of real-life situations (images under different lighting conditions, different angles, and different distances). You also need to collect sufficient images for each image category.

References

1. https://medium.com/@saba99/preprocessing-techniques-182c8ac186f6
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8562352/
3. https://encord.com/blog/data-refinement-guide-computer-vision/
4. https://www.quora.com/What-kind-of-image-pre-processing-should-be-done-before-feeding-it-to-a-Convolutional-Neural-Network
5. https://medium.com/@davidfriml/how-to-prepare-images-for-a-training-dataset-f6889433249b

Considerations for Text Preprocessing

In the field of deep learning, preparing text data starts with collecting it from various sources, such as clinical operations, external regulatory agencies, publications, etc. This involves handling different file formats like CSV and JSON. The crucial step of cleaning the data involves removing unnecessary elements like HTML tags and standardizing the text, for instance, by converting it to lowercase. This ensures uniformity and prepares the data for further processing.

The next phase is text normalization. Here, the text is broken down into smaller units, known as 'tokens', and common but less meaningful words (such as 'and', 'the') are removed as they are known as ‘stopwords’. Techniques like stemming and lemmatization are applied to simplify words to their base forms. For large datasets, breaking the data into smaller chunks or using parallel processing can enhance efficiency. The dataset can also be augmented through methods like synonym replacement, which adds variability and depth to the text.

The text transformation is central to preparing data for deep learning models. Here, methods like Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) convert text into numerical features, while sophisticated word embeddings and/or sentence embeddings such as Word2Vec, GloVe, BERT, and Universal Sentence Encoder capture the semantic intricacies of language. Special elements like numbers, dates, and emojis are handled with specific strategies to retain their contextual significance.

Finally, it's essential to ensure the quality of the pre-processed text. This involves thorough checks for consistency and verifying the accuracy of each pre-processing step. Such diligence is key to maintaining the integrity of the data, which in turn, significantly impacts the performance of the deep learning models.

References:

1. https://medium.com/@devangchavan0204/complete-guide-to-text-preprocessing-in-nlp-b4092c104d3e
2. https://neptune.ai/blog/data-augmentation-nlp
3. https://www.datacamp.com/tutorial/python-bag-of-words-model
4. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
5. https://www.geeksforgeeks.org/python/python-word-embedding-using-word2vec/
6. https://www.geeksforgeeks.org/nlp/pre-trained-word-embedding-using-glove-in-nlp-models/
7. https://www.tensorflow.org/text/tutorials/classify_text_with_bert
8. https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder

Chapter 3: Data Governance

Data governance refers to a collection of policies and standards overseeing how data is gathered, stored, processed, shared or reused, and disposed of. It governs who can access what kinds of data and what the expectations and limitations are. Good data governance improves decision-making, innovation, regulatory compliance, and efficient data usage. Proper knowledge of the data available – both its location and contents, consistent policies aligned with business goals, and privacy by design are all signs of data governance being done well.

Data access

Refers to the ability to retrieve and utilize data stored in a database or other storage system. It encompasses a range of activities, including data retrieval, management, analysis, and protection (i.e retrieving data from the Biostats drive, RWE NaSuni, etc). Depending on the infrastructure, access can be granted "in-place" – where the data is not moved or copied but rather access to a specific asset is granted to a specific user for a specific period. Alternatively, a copy of the data asset can be provided using secure file transfers, S3 bucket transfers, etc. Data Use Agreements may be required to govern the use of such data assets outside of the original storage system.

Data storage

Data storage is the retention of information via technology that is explicitly designed to store that data and make it as accessible as possible (i.e, storing data under the Biostats drive, RWE NaSuni, HPC etc).

Data Privacy/Anonymization/De-identification

Data Privacy encompasses processes and standards aimed at protecting the privacy and identity of individuals represented in data records in line with relevant regulations.

Data de-identification is the process of removing or transforming personally identifiable information (PII) in a data set to facilitate the reuse of data in scenarios for which the data was not originally collected.

Data anonymization combines data de-identification methods with statistical assessment (ideally, quantitative in nature) which ensures the right level of data transformation is used – enough to reduce residual reidentification risk to an acceptable level, but no more than necessary, so that maximum data utility can be retained.

It needs to be stressed that reducing the risk to 0 is neither possible nor desired as it would inevitably lead to total loss of data utility. Rather, the goal is to reduce the risk of reidentification below a threshold deemed acceptable. There are precedents for risk thresholds that should be observed in the context of clinical data which guard against targeted and inadvertent reidentification.

There are several types of identifiers for which different de-identification strategies can be utilized. Direct identifiers are details which refer to individuals directly. A tested approach to deal with them is pseudonymization, which is a process of substituting identifiable information with a placeholder. A common application of this method is hashing. Each ID is hashed consistently across records. Hashes should not be reversable to the original value. Another approach is data masking, which is the process of de-identifying where the records are not only changed to no longer point to an individual, but the overall structure of the data is also maintained. A common example of this method is using XXX-XX-XXXX to denote an SSN. Pseudonymization does not affect the data's utility if links within the data set are maintained. Moreover, direct identifiers other than subject ID rarely have scientific value and can be dropped.

Quasi-identifiers are a collection of data that in themselves are not identifying but when combined or used alongside other information can make records identifiable. Examples of such information include birth date, race, sex, and geographical location. An effective strategy to deal with quasi-identifiers is to generalize them. The birth date can be replaced with age or age group, while geographical details can be elevated to country, region, continent, etc. It should be noted that generalization affects data utility. The broader the criteria to generalize quasi-identifiers the less granular those datapoints become. Residual risk of re-identification can be assessed to ensure that the right combination of rules has been applied to reduce the risk to an acceptable level (data anonymization). One approach is the K-anonymity test, which focuses on ensuring that no record would correspond to a single individual but to at least K individuals.

Another type of identifier is dates. Where possible, these can be replaced with relative study days, or offset by a random number, consistent within a record, to retain relationships between events.

Free entry and verbatim text values can contain identifiable details and rule-based de-identification, even in combination with regular expressions, may be impractical. Where coded values are available, those should be retained, while the verbatim values should be dropped.

De-identifying the data, and anonymization in particular, leads to some key benefits. First, it limits your risk exposure and protects individuals' identity and privacy. Second, since the data is no longer considered to be identifiable or personal, it makes sharing and reusing data possible without requiring additional consent from data subjects. It may also address privacy limitations and regulations, e.g., it may not be required to report breaches or data leaks, and it certainly minimizes the impact of any data leaks. Lastly, anonymization facilitates data reuse and makes it easier to share internally and with third parties through data use agreements or secure data licensing.

Chapter 4: Data Splitting Strategies and Imbalanced Data

Train/Test Split

It is recommended to split your dataset into training data and test data for deep learning projects. A common splitting strategy is 80% of the data for training and 20% of the data for testing. Data splitting helps minimize overfitting of your data and helps estimate model performance on unseen data in an unbiased way. The training set is used for training the model and determining NN weights, whereas the test set is useful for determining model performance.

Train/Validation/Test Splits

Typically, neural networks have various hyperparameters that require tuning. For this scenario, it is recommended to divide your dataset into three partitions: training, validation, and test. The training and test set have a similar purpose as above, while the validation set can be used to tune NN hyperparameters and perform feature selection.

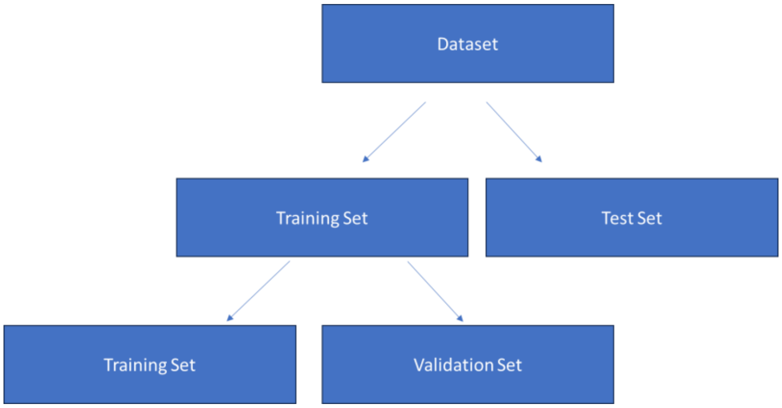


Figure 1: Data splitting
This figure shows a data set split into training, validation, and test sets.

Data Splits and Imputation

Note that data splitting should occur prior to imputation. The imputation model should be trained on training data to avoid data leakage. Then, the fitted imputation model is applied to the validation and test sets via re-imputation. See https://mlr.mlr-org.com/articles/tutorial/impute.html for more information.

Imbalanced data strategies

For imbalanced data (i.e. one class has many more examples than other class, say 10:1), it is recommended to rebalance the data (1:1) for the training set to create an unbiased training sample. Rebalancing can be accomplished through subsampling, ROSE, SMOTE, propensity score matching, or by improving the data gathering strategy. For the test and validation sets, the imbalance ratio should match the target population of interest. If the observed imbalance ratio in the validation and test sets do not match the target population ratios, then resampling can be performed.

Cross-validation

Another approach for model evaluation is cross-validation. While this approach is common for traditional machine learning, it is not used often for neural networks due to its computational cost. We begin by splitting our dataset into a training part and testing part. In 5-fold CV applied to the training set, we split the training set into 5 equal parts.

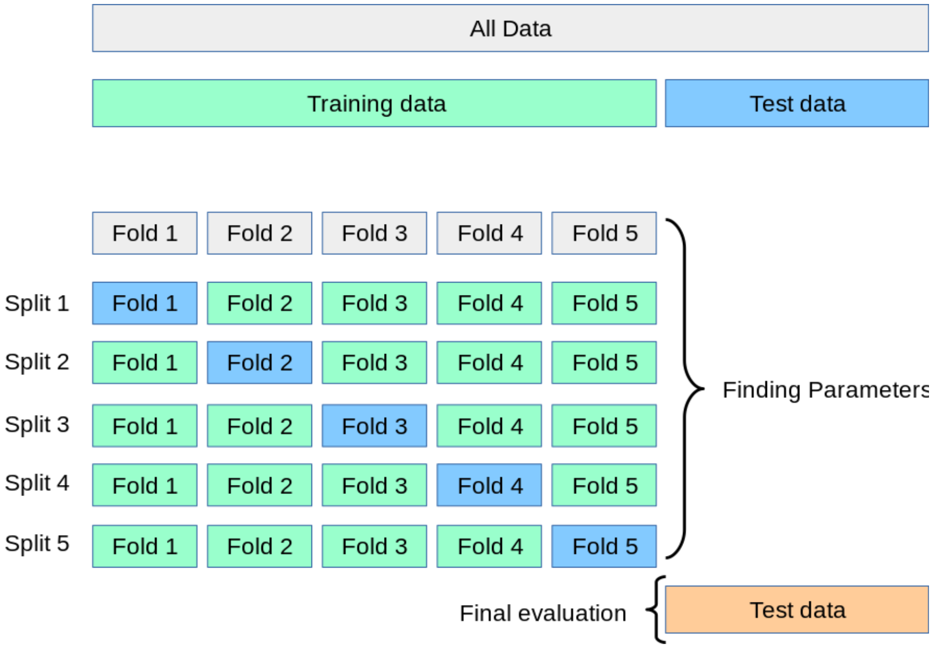


Figure 2: Example of 5-fold cross validation from https://scikit-learn.org/stable/modules/cross_validation.html

In the first iteration the deep NN is trained on folds 2-5 and assessed on fold 1 (validation set). In the second iteration, the deep NN is trained on folds 1,3,4,5 and assessed on fold 2. The process continues until the 5th iteration. Averaging the performance across the blue folds gives our final performance estimate. By varying the hyperparameters of the NN (ie learning rate, drop-out rate, batch size, etc), we can determine the optimal hyperparameter set that maximizes our average performance estimate. With the final set of hyperparameters, we fit the model to the training set and assess its performance on the test data.

Chapter 5: Neural Network Selection

Feedforward network

A feedforward network is the simplest deep learning architecture. It consists of a series of input neurons (which are fed a single data point or image), a series of hidden layers, and output neurons. These layers are connected with a dense set of connections each with an associated weight. A simple FF network looks like:

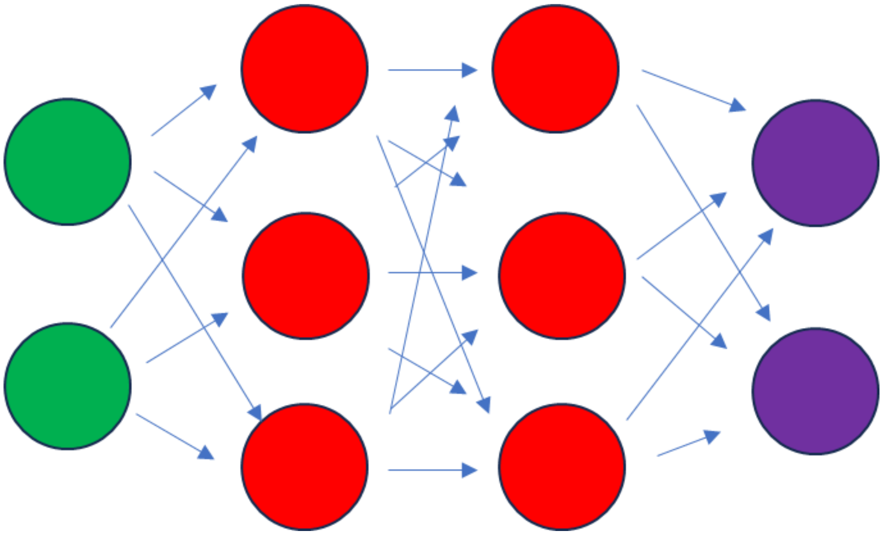


Figure 3: A feedforward network
Green nodes denote input nodes, red nodes are hidden layers, and purple nodes are output nodes.

The output states help make a prediction for a regression problem or output a category for a classification problem. Each neuron contains a series of incoming connections, a bias term, and one output. The final output of an individual neuron is a weighted sum of inputs plus bias then activated with an activation function:

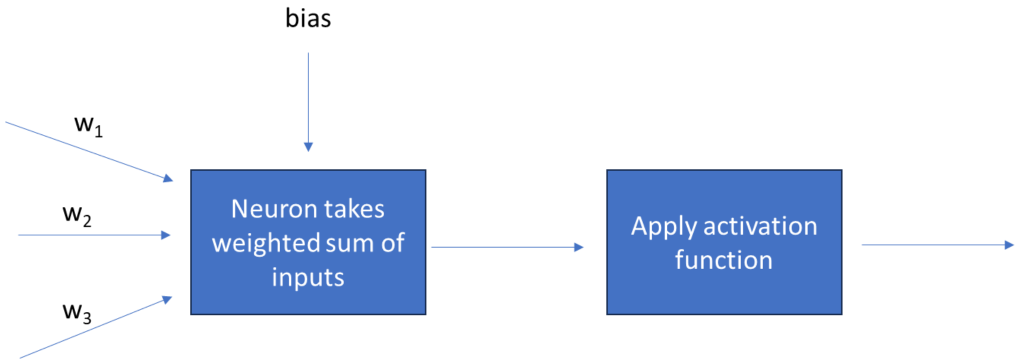

bias
w_1
w_2
w_3
Neuron takes weighted sum of inputs
Apply activation function

Figure 4: a schematic of a neuron

References

1. https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d
2. https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-feed-forward-network-in-deep-learning/

Convolution Neural Network (CNN)

A convolutional neural network is a common architecture for image analysis which usually consists of a series of convolution and pooling operations. One famous example is the LeNet architecture:

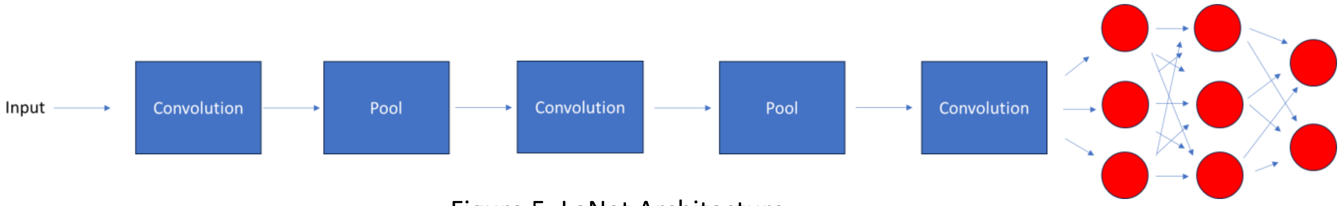


Figure 5: LeNet Architecture

The convolutional layers help extract key features from the image by applying a series of image filters, whereas the pooling layers help to down sample the feature maps and investigate the image at different levels of resolution. See example code at: https://medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b.

The applications of CNNs are widespread in image analysis (object detection, face detection, emotion detection, medical imaging, and image captioning) and text analysis (machine translation, next word prediction for smartphones, and document classification).

One challenge of CNN networks is interpretability of the neural network layers. To address this challenge, various methods such saliency maps, grad-cam, and LIME can highlight regions of the image that the network focused on during its classification of image. For more information about deep learning interpretability methods, see the chapter on **Model Interpretability**.

References:

1. https://medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b
2. https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network
3. https://www.codemag.com/Article/2205081/Implementing-Face-Recognition-Using-Deep-Learning-and-Support-Vector-Machines
4. https://www.geeksforgeeks.org/deep-learning/emotion-detection-using-convolutional-neural-networks-cnns/
5. https://pmc.ncbi.nlm.nih.gov/articles/PMC7778711/
6. https://arxiv.org/html/2404.18062v1

7. https://cs224d.stanford.edu/reports/LambAndrew.pdf
8. https://github.com/nimitsolanki/next-word-predictor
9. https://github.com/Arghyadeep/Document-Classification-with-CNN

Recurrent neural networks (RNNs)

A RNN is a deep learning architecture that is often used to model sequence data. Common applications include speech recognition, natural language processing, time series analysis, video tagging, OCR, and others. Its architecture is shown below:

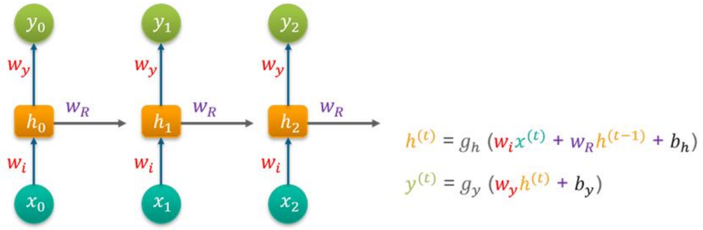


Figure 6: RNN architecture
Image reference: https://www.edureka.co/blog/recurrent-neural-networks/#z2

where h is a hidden state, x are inputs, y are outputs, w are weights, b is bias, and g is a nonlinear activation function. RNNs are named recurrent neural networks due to their recurrence relation in h. Hidden state, $h^{(t)}$, relates to a linear combination of inputs, previous hidden state, and bias which is then activated by the activation function, g. The output state y is the activation function applied to a linear combination of the hidden state and a bias term.

In recent years, RNNs are not a popular architecture due to two major limitations. First, RNNs can suffer from the vanishing gradient problem which can lead to slow training, and secondly, RNNs can have exploding gradients leading to unstable training. Alternative architectures such as LSTMs and GRUs overcome these limitations. For a code example of an RNN, please see: https://github.com/microsoft/AI-For-Beginners/blob/main/lessons/5-NLP/16-RNN/RNNPyTorch.ipynb

References:

1. https://keras.io/examples/vision/video_classification/
2. https://github.com/qjadud1994/CRNN-Keras
3. https://www.edureka.co/blog/recurrent-neural-networks/#z2

Long short-term memory network (LSTM)

A LSTM network is an improvement compared to RNNs since it has less problems with vanishing gradients and can learn long range dependencies. However, LSTMs are slow to train. Another disadvantage is that LSTM networks are not parallelizable so they can't take advantage of GPU acceleration. Despite these disadvantages, LSTMs have been applied in text mining (sentiment analysis, language modelling, text generation), audio analysis (speech recognition, wake word detection), and video classification.

The architecture of a LSTM consists of a forget gate, a learn gate, a remember gate, and a use gate. The forget gate allows the network to decide which information to forget, whereas the learn gate helps the network decide which information from the inputs and short-term memory to learn. The remember gate decides how to update long term memory and lastly the use gate updates short-term memory based on long term memory, short term memory, and the current input.

For a code example of a LSTM applied to time series forecasting, please visit: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/.

References:

1. https://ravjot03.medium.com/how-lstms-solve-the-vanishing-gradient-problem-in-sequential-data-b786eec3966f
2. https://medium.com/@prudhviraju.srivatsavaya/lstm-implementation-advantages-and-diadvantages-914a96fa0acb
3. https://arxiv.org/html/2408.10006v1
4. https://wandb.ai/madhana/Language-Models/reports/Language-Modeling-A-Beginner-s-Guide---VmlldzozMzk3NjI3#:~:text=LSTMs%20can%20be%20used%20as,next%20word%20in%20the%20sequence.
5. https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/
6. https://www.geeksforgeeks.org/machine-learning/text-generation-using-recurrent-long-short-term-memory-network/
7. https://github.com/giyu51/wake-word-detection
8. https://www.apriorit.com/dev-blog/609-ai-long-short-term-memory-video-classification
9. https://builtin.com/data-science/recurrent-neural-networks-and-lstm
10. https://medium.com/@ck2886/lstm-explained-633d89384004

Transformers

A recent popular neural network architecture is the transformer architecture. It has some advantages compared to other networks, namely that the input tokens can be parallelized so that gpu acceleration can be applied. A disadvantage of this network architecture is that it is much more complex architecture compared to previous networks. Despite its complexity, transformers have been applied to many areas of text mining including sentiment analysis, question and answering, machine translation, text summarization, and text classification.

Specifically, the transformer architecture consists of an encoder subnetwork on the left and a decoder subnetwork on the right. The encoder subnetwork finds an encoded vector representation of input words, while the decoder subnetwork can learn representations of the relationship between the input and outputs. Some major architectural innovations compared to the previous architecture include attention (the network pays attention to certain words and their context), masked attention (which allows for masked learning), and positional encoding (which encodes the word position in a sentence).

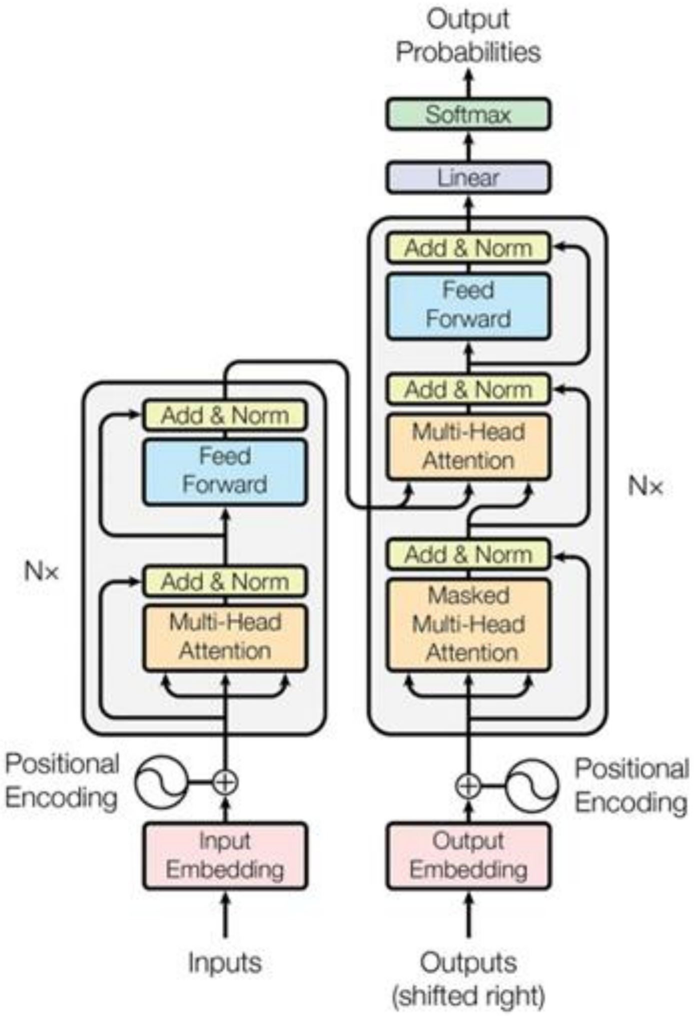


Figure 7: Transformer architecture
Image reference: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

The Text Mining Center of Excellence at Biogen has implemented transformer models for various internal applications in regulatory, quality, and other R&D domains. By leveraging state-of-the-art language models and machine learning techniques, the team has developed solutions to enhance text analysis, risk assessment, pattern recognition, and document processing. These initiatives aim to improve efficiency, streamline operations, and provide valuable insights across different business areas. While showing promising results, the projects need further validation for potential wider deployment. For code examples of the transformer architecture, please see: https://huggingface.co/docs/transformers/v4.41.3/en/model_doc/roberta#transformers.RobertaForQuestionAnswering

References:

1. https://medium.com/@hassaanidrees7/transformers-in-action-real-world-applications-of-transformer-models-1092b4df8927
2. https://builtin.com/artificial-intelligence/transformer-neural-network
3. https://huggingface.co/docs/transformers/en/tasks/sequence_classification
4. https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

The Autoencoder

The autoencoder architecture is commonly used in applications that require compression or dimension reduction. Other applications include image processing (image generation, image denoising, and image compression), time series data generation, and anomaly detection. Here we will focus on the simplest autoencoder architecture, but other variants such as Deep CNN autoencoder and Denoising autoencoder exist.

The simplest model of an autoencoder architecture is displayed below. The inputs are compressed into a lower dimensional representation through the bottleneck layer. This process is called encoding. From the hidden layer to the output layer performs decoding (or transforming the latent representation back to the original dimensionality). The objective in training this model is for the reconstruction image (or dataset) at the output layer to be as close as possible to the input image (or dataset).   For code examples, please visit: https://www.tensorflow.org/tutorials/generative/autoencoder

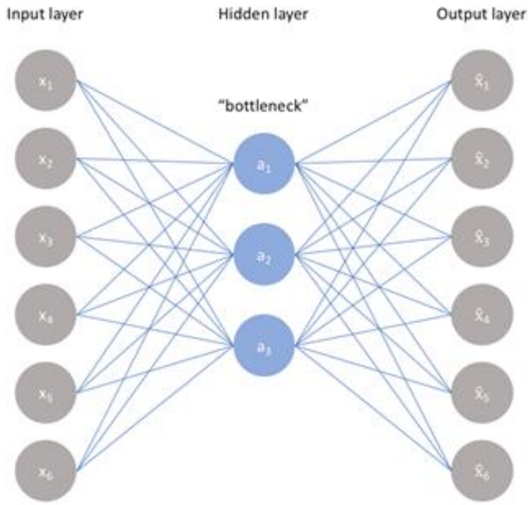


Figure 8: autoencoder architecture
Image reference: https://www.v7labs.com/blog/autoencoders-guide

Reference

1. https://www.v7labs.com/blog/autoencoders-guide

Generative Adversarial Network architecture (GANs)

GANs have been recently quite popular due to their ability to generate new images from a text prompt (ie deepfakes) and their ability to age faces, perform photo up-sampling, perform photo inpainting, and perform style conversion (such as transforming a photo to a sketch or transforming a photo to a painting). GANs are based on the idea of an adversarial game, where the discriminator tries to determine which images are real versus fake and the generator tries to improve its ability of generating fake images to fool the discriminator. After much training, the generator becomes so good at generating fakes that the discriminator only has a 50% chance of detecting the fakes.

Early literature used the DCGAN model (Deep Convolutional Generative Adversarial Networks) with CNNs for both the generator and the discriminator. The generator has as input a random input and will generate a fake image. The discriminator is a binary classification model to distinguish real versus fake images. The detailed architecture of GANs is shown below:

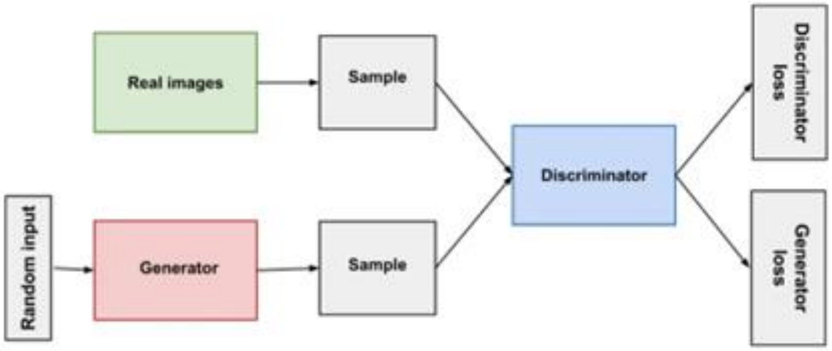


Figure 9: GAN architecture from https://developers.google.com/machine-learning/gan/images/gan_diagram.svg

The generator and discriminator are updated as follows:

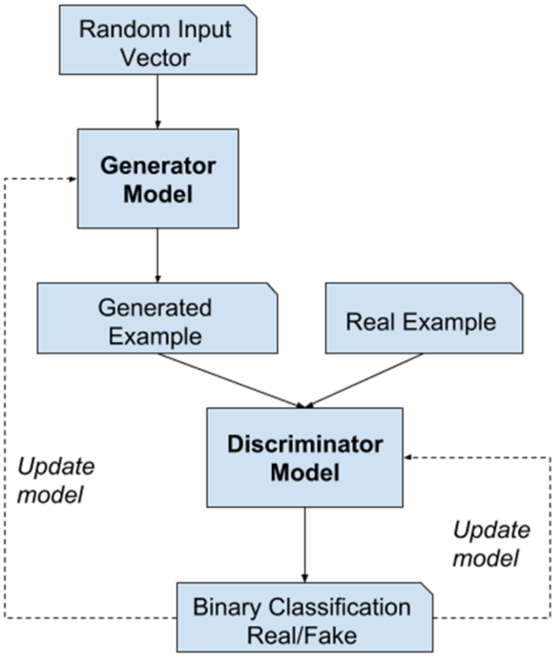


Figure 10: GAN update process flow, image from: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

In detail, if the generator fools the discriminator, the generator network is rewarded and the discriminator network is penalized. In contrast, if the discriminator successfully detects the fake, the discriminator network is rewarded and the generator network is penalized. For code examples, please visit: https://www.tensorflow.org/tutorials/generative/dcgan

References:

1. https://developers.google.com/machine-learning/gan
2. https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
3. https://arxiv.org/abs/2012.03459
4. https://medium.com/@kdk199604/srgan-the-power-of-gans-in-super-resolution-94f39a530a61
5. https://github.com/researchmm/AOT-GAN-for-Inpainting
6. https://arxiv.org/html/2409.00345v1
7. https://ucladatares.medium.com/make-a-monet-image-style-transfer-with-cycle-gans-5475dcb525b8
8. https://arxiv.org/abs/1511.06434

Mixture of Experts (MoE)

A mixture of experts architecture (MoE) has been a popular architecture in recent months (such as X AI's grok-1, Databricks's DBRX, Mixtral's 8x22B, DeepSeek-AI's deepseek, and Tencent's Hunyuan-Large) due to its efficiency in pretraining and its fast inference speed. Given a fixed compute budget, MoE models can have a larger model size or larger dataset size compared to a dense model. Although MoE models have some important advantages compared to dense models, we also need to be mindful of their limitations, namely high GPU memory requirements and overfitting challenges during fine tuning.

Technically, the MoE architecture consists of a series of experts along with a router/gating network. The router will determine which tokens are received by each expert. Specifically for the transformer architecture, the dense feed forward layers are replaced with MoE layers, which consists of a router and a collection of experts. These experts are modelled as feed forward networks and the router can be modelled as a softmax gating function or by Noisy Top-k Gating. Besides the transformer architecture, MoEs have been applied to other architectures such as CNNs, where a router is combined with CNN experts (see https://github.com/shibuiwilliam/mixture_of_experts_keras).

During inference time, the MoE architecture is quite efficient since only a subset of parameters is active leading to its fast inference speed. The architecture is also quite scalable allowing 100s or 1000s experts under the same inference budget. However, the tradeoff is that all experts are loaded into VRAM resulting in high GPU memory requirements.

Some additional challenges with MoEs include load balancing of tokens, training stability, overfitting during fine tuning, and computational inefficiency due to branching. Each of these challenges has potential solutions. For example, auxiliary loss functions have been proposed to help with load balancing of tokens. Training stability can be improved with selective precision or router Z-loss, and fine-tuning strategies have been explored by Shen et al (https://arxiv.org/pdf/2305.14705). Additional research is being conducted to improve branching inefficiencies.

References:

1. https://www.marktechpost.com/2024/11/16/list-of-large-mixture-of-experts-moe-models-architecture-performance-and-innovations-in-scalable-ai-solutions/
2. https://huggingface.co/blog/moe

GNNs

GNNs are a type of deep neural network architecture designed to process and analyze data that is structured as a graph, which can take any shape or size and can contain different modalities like image and text. In a graph, data is represented as a collection of interconnected nodes and edges. Nodes can represent entities (e.g., people, items, molecules), and edges represent the relationships between those entities (e.g., friendship, product categories, chemical bonds). GNNs excel at capturing relationships and dependencies between nodes in a graph, making them suitable for a variety of tasks involving interconnected data.

GNNs leverage a “message-passing” mechanism, where each node aggregates information from its neighboring nodes through the edges connecting them. This aggregation process allows nodes to learn contextualized representations based on the relationships and information they receive from their neighbors. The aggregated information is then used to update the node’s own internal representation, and this process is repeated iteratively across multiple layers.

GNNs can be broadly categorized into several types based on their architecture and how they process graph-structured data in non-Euclidean space. These include Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), Graph Recurrent Networks (GRN), and various other specialized GNNs like Message Passing Neural Networks (MPNN). GCNs and GATs are particularly prominent due to their ability to leverage the structure of graphs to learn node and edge representations.

Unlike other deep learning networks such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), GNNs offer distinct advantages from their specific design to handle graph-structured data. This includes the ability to process diverse graph structures, capture relational reasoning, and perform tasks at different levels (node, edge, and graph). In contrast, CNNs are primarily used for grid-like data (e.g., images) and RNNs for sequential data (e.g., text), making them less suitable for graph-based tasks.

Nevertheless, there are several downsides to GNNs. The “message-passing” algorithm across nodes can require a massive computational cost for large graphs. If the number of layers goes up, distinctiveness among node representations will perish, thereby leading to lower performance. In general, complex data preprocessing and feature engineering to create high quality graph data are necessary. As of 2025, GNNs are not generally leveraged for image classification, where CNNs are more popular to use.

GNNs are particularly well-suited for analyzing data that is inherently relational, such as time-series forecasting (only if nodes in the time series are associated, otherwise LSTM or transformer models will be applicable), social networks, knowledge graphs, and molecular structures. By iteratively aggregating information from neighbors, GNNs can capture intricate patterns and dependencies within a graph. GNNs have been successfully applied in various fields, including drug discovery to predict the biological activity of new molecules based on their molecular graph structure, systems to recommend products or users based on the relationships between them in a graph, social network analysis to analyze social connections and predict use preferences or social interactions, and fraud detection to identify fraudulent transactions or activities based on the relationships between actors in a network. For those applications, GNNs can handle various graph structures, including directed, undirected, and weighted graphs, and can incorporate node and edge features.

Particularly, GNNs accelerate drug discovery by enabling the modeling of complex molecular structures and biological networks. GNNs can be used to predict drug-target interactions, molecular properties, and even assist in designing new drugs. GNNs can directly process molecular graphs, capturing local and global dependencies within a molecule, represented as graphs, where atoms are nodes and chemical bonds are edges. This allows GNNs to model complex interactions between molecular components that traditional vector-based models might miss. GNNs can predict which drugs will interact with specific protein targets, which is crucial for drug discovery and drug repurposing. GNNs can be used to predict whether combining two drugs will have a synergistic effect, which can lead to more effective treatments. Also, GNNs can help predict the toxicity of potential drug candidates, helping to identify and avoid toxic compounds early in the development process. GNNs can capture both local and global interactions within molecules, which is crucial for understanding drug behavior, and have demonstrated high predictive power in various drug discovery tasks. Such GNN architectures, like Graph Attention Networks (GATs), offer interpretability, allowing researchers to identify key atoms or substructures influencing predictions.

The "Awesome-GNN-based-drug-discovery" github open-source repository (https://github.com/gozsari/Awesome-GNN-based-drug-discovery) provides a curated list of resources and tools for drug discovery, leveraging GNNs. The repository helps researchers investigate the comprehensive overview of the most recent developments in GNN-based drug discovery such as scientific papers, public domain datasets, programming tutorials, and open-source software. For a wide range of GNN applications with graph-structured data, PyG (PyTorch Geometric, https://github.com/pyg-team/pytorch_geometric) is a library built upon PyTorch. This repository has lower barriers of entry to run GNNs even for beginners compared to machine-learning toolkits because: 1) it takes only 10-20 lines to train a GNN model utilizing a tensor-centric API, 2) a comprehensive interface to build cutting-edge GNN models is provided, 3) PyG models are flexible to be extended for customizing user's own research with GNNs, and 4) PyG supports learning on diverse types of graphs in challenging real-world scenarios such as scalable GNNs/dynamic GNNs/heterogeneous GNNs.

WholeGraph (https://github.com/rapidsai/wholegraph) enables faster training of GNNs, particularly on large-scale graphs by overcoming limitations in memory capacity and communication speed. It also simplifies the development process by providing a unified storage and retrieval mechanism for multi-GPU GNN training (https://developer.nvidia.com/blog/optimizing-memory-and-retrieval-for-graph-neural-networks-with-wholegraph-part-1/). WholeGraph leverages NVLink, NVIDIA's high-speed inter-GPU communication technology, to facilitate fast and efficient data transfer between GPUs. WholeGraph works in conjunction with cuGraph, NVIDIA's open-source library for GPU-accelerated graph analytics. This enables easy integration with other GNN frameworks like Deep Graph Library (DGL) and PyG.

References:

1. https://blogs.nvidia.com/blog/what-are-graph-neural-networks/
2. https://www.ibm.com/think/topics/graph-neural-network
3. Khemani, B., Patil, S., Kotecha, K. *et al.* A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *J Big Data* **11**, 18 (2024). https://doi.org/10.1186/s40537-023-00876-4
4. https://blogs.mathworks.com/deep-learning/2025/02/04/graph-neural-networks-in-matlab/

Mamba Architecture

The Transformer architecture has played a pivotal role in the success of Large Language Models (LLMs), forming the backbone of nearly all current LLMs, from open-source models like Mistral to closed-source models like ChatGPT. As researchers continue to push the boundaries of LLM capabilities, new architectures are emerging that may surpass the performance of Transformers. One such innovative approach is Mamba, a State Space Model, which shows promise in advancing the effectiveness of LLMs.

Mamba is an advanced neural network architecture designed to enhance sequence modeling by leveraging State Space Models (SSMs). Unlike traditional models, Mamba introduces a selective scan algorithm and a hardware-aware algorithm to address specific limitations of existing architectures like Transformers and Recurrent Neural Networks (RNNs). You can try mamba with the python package: https://github.com/state-spaces/mamba

State Space Models (SSMs)

SSMs are mathematical models used to describe dynamic systems through state representations. They use continuous signals and discrete sequences to predict future states based on input sequences and hidden states. SSMs consist of two main equations: the state equation and the output equation. These equations involve matrices A, B, C, and D, which are learnable parameters that influence how the system evolves over time. One graphic illustration of SSMs is shown below: (https://www.maartengrootendorst.com/blog/mamba/)

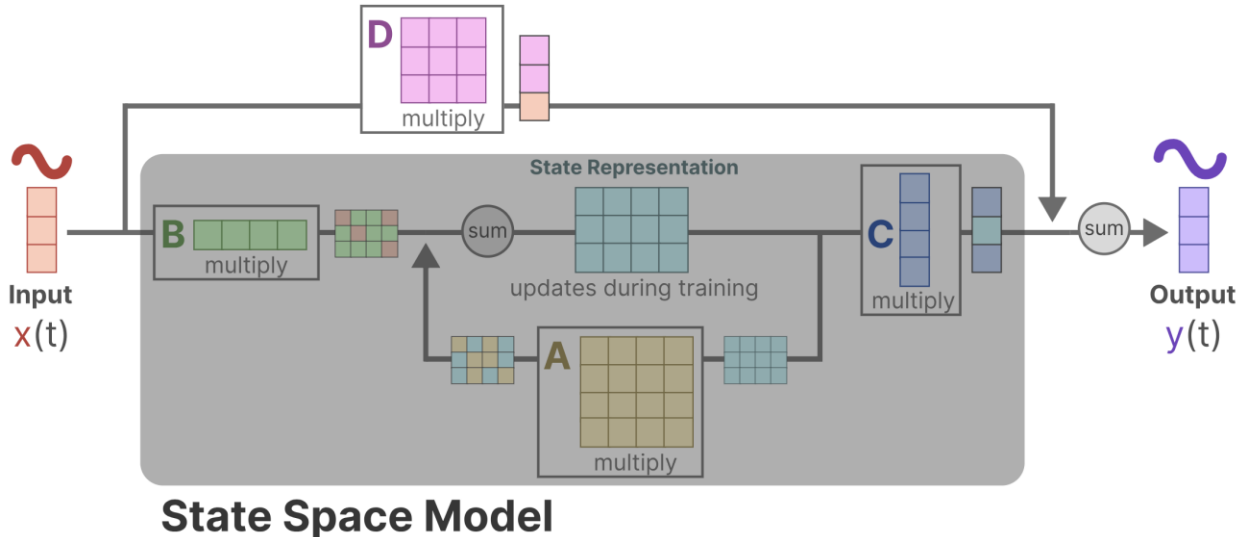


Figure 11: Mamba State Space Model

Key Innovations in Mamba

1. **Selective Scan Algorithm:** Mamba introduces a dynamic approach to selectively compress relevant information into the state, making matrices dependent on the input sequence length and batch size. This allows for content-aware processing.
2. **Hardware-aware Optimization:** By using techniques such as parallel scan (see https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda for details), kernel fusion (https://www.maartengrootendorst.com/blog/mamba/), and recomputation, Mamba efficiently manages memory and computation, particularly on GPUs. This reduces bottlenecks caused by frequent memory transfers between SRAM and DRAM.
3. **Recurrent and Convolutional Representations:** Mamba leverages both recurrent and convolutional representations. The recurrent representation ensures efficient inference, while the convolutional representation enables parallelizable training.

4. **HiPPO Initialization:** The HiPPO (High-order Polynomial Projection Operator) matrix is used to capture long-range dependencies within sequences, providing a more accurate and efficient state representation.

Mamba Block Structure

Mamba's architecture is composed of multiple stacked Mamba blocks, each designed to process input sequences through several stages:

- **Linear Projections**: Each block starts with linear projections to expand the input embeddings, setting the stage for subsequent processing.
- **Convolutions**: The model employs convolutions to prevent the independent calculation of tokens, thereby maintaining the contextual relationship between them.
- **Selective SSM**: The heart of each block is the selective SSM, which includes:
    - **Recurrent SSM**: Constructed through discretization to handle long-range dependencies effectively.
    - **HiPPO Initialization**: A technique that initializes the model to manage long-term dependencies better.
    - **Selective Scan Algorithm**: This dynamically compresses information, ensuring that only relevant data is retained in the state.
    - **Hardware-aware Algorithm**: Optimizes computation by leveraging modern hardware capabilities for efficient training and inference.
- **Mamba block diagram can be found at**: https://arxiv.org/pdf/2312.00752

Advantages Over Traditional Models

Mamba's innovative approach to sequence modeling has significant implications for natural language processing (NLP), time series analysis, and other domains requiring efficient and accurate sequence handling. Its ability to selectively retain information and optimize computation makes it a powerful tool for modern AI applications. Specifically, it has the following advantages:

1. **Efficiency**: Mamba’s hardware-aware algorithm and parallel scan techniques reduce computational costs and enhance scalability compared to traditional models like Transformers and RNNs, which often suffer from high computational complexity and inefficiency in handling long sequences.
2. **Dynamic Information Filtering**: Unlike traditional models that process all information indiscriminately, Mamba’s selective scan algorithm filters out irrelevant data dynamically, improving memory efficiency and model performance.
3. **Improved Long-Range Dependency Handling**: Through the use of HiPPO initialization and state space representation, Mamba can manage long-range dependencies more effectively than conventional models, which often struggle with maintaining long-term context.
4. **Scalability and Flexibility**: Mamba’s ability to switch between convolutional and recurrent representations provides flexibility, allowing it to balance between parallelization during training and efficiency during inference.

References:

1. https://medium.com/@joeajiteshvarun/mamba-revolutionizing-sequence-modeling-with-selective-state-spaces-8a691319b34b
2. Dao, T., Fu, T., R'{e}, C., et al. (2023). *Hungry Hungry Hippos: Towards Language Modeling with State Space Models*. arXiv preprint arXiv:2312.00752. https://arxiv.org/abs/2312.00752

Residual Networks (ResNets)

ResNets are used in many applications such as computer vision, medical imaging, and fault detection. For computer vision, ResNets are a useful deep learning architecture for image data analysis such as object detection and image segmentation. For example, ResNets can assign a class label to each pixel in an image, which is useful for medical imaging and scene understanding. ResNets can also enhance image quality by upscaling low-resolution images, which can improve satellite imagery, old photographs, and video frames. To interpret medical images, ResNets are helpful with clinical diagnosis, staging, therapy planning, and target selection for serious illness. For instance, ResNets can help distinguish COVID-19 cases from other pneumonia cases, diagnose cardiomegaly, and detect and classify brain tumors. For fault detection, ResNets are applicable to detect anomalies and detect faults on seismic structural images.

Moreover, ResNets are a popular choice for transfer learning, which loads a pre-trained ResNet model like ResNet50 and then fine-tune it on specific dataset. This involves freezing the convolutional layers of the pre-trained model and adding or replacing the final classification layers to adapt it to the specific task. This approach can save training time and improve performance as ResNet's architecture allows for deeper networks that are more robust and less prone to vanishing gradients, making it suitable for transfer learning.

ResNet was invented to solve the problem of vanishing or exploding gradients, which causes an increasing error rate as the number of layers in CNN-based architecture increases. A residual (Skip or Shortcut) connection is the solution by fitting residual mapping (e.g., $g(x) \equiv f(x) - x$ with a residual block) instead of underlying (identity) mapping (e.g., $f(x) = g(x) + x$ with a regular block) and hence it becomes easier to train with respect to $g(x) = 0$. The decision of which layers are skipped is made by regularization, for passing over layers harming the performance of the architecture. The residual connection helps deep learning models such as original LSTM network and transformer models train easier and achieve higher accuracy with tens or hundreds of layers.

ResNet is well-known for model robustness, high accuracy, and ease of fine-tuning primarily in image classification despite computational inefficiencies. The cost of higher computational resources (memory consumption) and longer training times induced by more layers in deeper learning architecture, is the tradeoff of superior performance in accuracy. In practice, shallow-layered ResNet (e.g. ResNet18) is appropriate when computational efficiency must be prioritized such as when computational power is limited or automated image classification in resource-limited devices as examples. Deeper ResNets (e.g. ResNet50) is more favorable when high accuracy is required to address such tasks as image recognition in high-resolution images or complex image analysis where fine-grained details are important for object detection and image segmentation.

ResNets offer several advantages including improved training of very deep networks, faster convergence and reduced model complexity. ResNets facilitates the training of deep neural networks without suffering from degradation in performance which is the main issue in plain networks. The skip connections in ResNets allow the network to learn identity mappings making it easier for the model to learn the identity function when needed. The skip connections help in the optimization process which leads to faster convergence and less training time. The skip connections in the residual networks leads to better generalization on the unseen data as the network can skip unnecessary or irrelevant information.

However, one disadvantage of using ResNet is the potential disappearance of gradients in very deep networks, which can make the gradient descent process slow. This can hinder the training of the network, especially as the number of layers increases.

References

1. https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/
2. https://en.wikipedia.org/wiki/Residual_neural_network#:~:text=A%20residual%20neural%20network%20(also,reference%20to%20the%20layer%20inputs.
3. https://d2l.ai/chapter_convolutional-modern/resnet.html
4. https://github.com/zalandoresearch/fashion-mnist
5. https://medium.com/@mitrapremangshu/comparison-of-the-best-pretrained-vision-models-a-deep-dive-into-efficiency-accuracy-and-task-bb1b8da1dbbc
6. https://www.productteacher.com/quick-product-tips/resnet18-and-resnet50
7. https://viso.ai/deep-learning/resnet-residual-neural-network/
8. Xu W, Fu YL, Zhu D. ResNet and its application to medical image processing: Research progress and challenges. Comput Methods Programs Biomed. 2023 Oct;240:107660. doi: 10.1016/j.cmpb.2023.107660. Epub 2023 Jun 8. PMID: 37320940.
9. Amelio A, Bonifazi G, Cauteruccio F, Corradini E, Marchetti M, Ursino D, Virgili L. Representation and compression of Residual Neural Networks through a multilayer network based approach. Expert Systems with Applications. 2023 Volume 215, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2022.119391.
10. https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet#:~:text=ResNets%20are%20a%20common%20neural,up%20to%20thousands%20of%20layers.
11. https://www.linkedin.com/pulse/what-problems-do-resnets-residual-neural-networks-solve-anil-verma-jyjie#:~:text=Semantic%20SegmentationIn%20semantic%20segmentation%2C%20ResNets,tumor%20detection)%20and%20scene%20understanding.&text=ResNets%20are%20a%20popular%20choice,training%20time%20and%20improves%20performance.&text=ResNets%20enhance%20image%20quality%20by,photographs%2C%20and%20enhancing%20video%20frames.
12. https://medium.com/@ibtedaazeem/understanding-resnet-architecture-a-deep-dive-into-residual-neural-network-2c792e6537a9#:~:text=Residual%20Networks%20offer%20several%20advantages,convergence%20and%20reduced%20model%20complexity.&text=ResNets%20facilitates%20the%20training%20of,main%20issue%20in%20plain%20networks.

13. https://www.comet.com/site/blog/an-intuitive-guide-to-convolutional-neural-networks/#:~:text=ResNet's%20main%20advantage%20lies%20in,more%20computationally%20efficient%20than%20DenseNet.
14. https://typeset.io/questions/what-are-some-of-the-advantages-and-disadvantages-of-using-a-jy29ui45sh
15. Zhang, Aston; Lipton, Zachary; Li, Mu; Smola, Alexander J. (2024). "8.6. Residual Networks (ResNet) and ResNeXt". *Dive into deep learning*. Cambridge New York Port Melbourne New Delhi Singapore: Cambridge University Press. ISBN 978-1-009-38943-3

Variational AutoEncoders (VAEs)

VAEs are enhanced autoencoders that use a Bayesian approach to learn the probability distribution of input data and have many applications across a variety of domains, including image generation, anomaly detection, text generation, data augmentation, data denoising and imputation, medical imaging, and natural language processing. Compared with Generative adversarial networks (GANs), VAEs tend to show poor performance in image fidelity, but they are more stable and are better for estimating the probability distribution itself.

A deep generative model, VAE, is an autoencoder whose encodings distribution is regularized during the training to ensure that its latent space has good properties (i.e. is regular enough) enabling it to generate some new data and avoid overfitting.

To achieve both continuity (e.g. two adjacent points in the latent space should not generate two completely different outputs after decoding) and completeness (i.e. a point sampled from the latent distribution should generate “meaningful” output after decoding), regularization is recommended to train the model close to a multi-variate standard normal distribution with the mean vector of 0 and the variance-covariance matrix close to identity. The latent distribution (the posterior) can be intuitively estimated by Bayes theorem, incorporating the prior with the concurrent likelihood.

In practice, VAE is useful for (fictional/new) image generation by capturing the main characteristics of the training data. Although VAE achieved generalization by resolving the overfitting issue in Autoencoders, in general, its resulting image is less accurate than Generative Adversarial Networks (GANs) since GANs leverage a more complicated data-driven loss function in contrast to the VAE’s user-defined loss function.

While VAEs are powerful models, they have multiple problems and tradeoffs. For example, when being used to generate new images, VAEs are likely to be blurrier compared to other generative models due to mainly variance loss. To overcome this blurriness, more novel methods like PixelVAE, 2-Stage VAE, and VQ-VAE have been developed as very effective tools in generating good quality images. In the original VAEs, balancing between reconstruction loss and the regularization term is difficult so a variant like 2-Stage VAE was proposed to balance these two factors.

References

1. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
2. https://medium.com/@judyyes10/generate-images-using-variational-autoencoder-vae-4d429d9bdb5

3. Singh A, Ogunfunmi T. An Overview of Variational Autoencoders for Source Separation, Finance, and Bio-Signal Applications. Entropy (Basel). 2021 Dec 28;24(1):55. doi: 10.3390/e24010055. PMID: 35052081; PMCID: PMC8774760.
4. https://medium.com/@bkaplan18/an-overview-of-variational-autoencoder-vae-kl-divergence-evidence-lower-bound-elbo-6583729f169a

TABLE 2: SUMMARY OF RESNET VS VAES

	PROS	CONS
ResNet	• Model robustness, high accuracy, and ease of fine-tuning in image classification • Improved training features like faster convergence and reduced model complexity via skip connections, in VERY deep networks	• Computational inefficiency can be caused by more layers trained in deeper networks.
VAEs	• More stable and better for estimating the probability distribution • Useful for new image generation • Resolves the overfitting issue	• Poor performance in image credibility (lower accuracy and blurrier), compared to GANs

Diffusion Models

Diffusion models are advanced machine learning algorithms that generate data by adding noise to a dataset and then learning to reverse the process. This process allows the user to create detailed and high-quality realistic outputs, such as images, text, music, and videos. Diffusion models are a type of latent variable generative model, and their operations are leveraged by some mathematical tools such as conventional denoising diffusion probabilistic models (DDPM) or score-based generative models. Diffusion models start with a simple distribution and use a series of invertible operations to change it into a more complex distribution. They introduce (usually Gaussian) noise to the pre-processed data in the forward diffusion process, and then gradually reverse the process to transform the noise into a structured output for creating a new data point similar to the original dataset. By handling various input types, diffusion models have many applications, including image-to-image translation, text-to-image (or video) synthesis, (reverse) image search, layout-to-image generation, inpainting, and super-resolution tasks.

Compared to other generative models like GANs, diffusion models have better image quality, are more robust to overfitting, generate a more diverse range of images, and have an interpretable latent space. However, a significant drawback of DDPM is an intensive computation time for sampling. To overcome this limitation, other decent methods such as denoising diffusion implicit model (DDIM) for sampling with fewer steps and progressive distillation for fast sampling were invented. Some popular diffusion models include DALL-E 2, Imagen, Midjourney, and open-source Stable Diffusion that can generate realistic images based on the user’s text prompts to guide the image generation process.

In medical imaging, the Stable Diffusion model (a text-to-image generation model based on Latent Diffusion Models, LDM) can be applied to the tasks such as visualizing potential disease progression, enhancing low-quality image scans by tailoring generated images to specific patient characteristics, creating synthetic medical images with different variations of pathologies to expansively train AI models, and improving the accuracy of diagnosis by providing clearer and more elaborate 3D images of anatomical structures. Various tasks (e.g. text-to-image, super resolution, and so on) and pipelines can be implemented via **diffusers** Python package (https://github.com/huggingface/diffusers).

When conducting fine-tuning experiments on Stable Diffusion, compared to other fine-tuning methods like DreamBooth/HyperNetwork/Embedding, LoRA is the best efficient fine-tuning method that optimizes the model output by fine-tuning the weights of the U-Net cross-attention layer. While the image outputs of Stable Diffusion without fine-tuning is more abstract, the effects generated from LoRA's fine-tuning are visually closer to the real image. That is, the application of LoRA enables Stable Diffusion to gain a wider range of adaptability and application while maintaining its original performance. The example Python codes to fine-tune Stable Diffusion using LoRA can be found from https://machinelearningmastery.com/fine-tuning-stable-diffusion-with-lora/ and https://huggingface.co/blog/lora.

References

1. https://www.superannotate.com/blog/diffusion-models#:~:text=Diffusion%20models%20are%20advanced%20machine,transform%20it%20into%20something%20new.
2. https://shelf.io/blog/diffusion-models-for-machine-learning/#:~:text=Diffusion%20models%20for%20machine%20learning%20are%20cutting%2Dedge%20technologies%20that,the%20limits%20of%20artificial%20intelligence.
3. https://en.wikipedia.org/wiki/Diffusion_model#:~:text=In%20machine%20learning%2C%20diffusion%20models,space%20of%20all%20possible%20data.
4. https://academic.oup.com/bjrai/article/1/1/ubae013/7745314
5. https://www.sciopen.com/article/10.26599/TST.2024.9010047#:~:text=Diffusion%20models%20are%20a%20type,several%20medical%20image%20computing%20tasks.
6. https://netsolcloudservices.com/blog/stable-diffusion-based-solution-for-medical-imaging/#:~:text=Stable%20diffusion%20is%20a%20valuable,with%20high%2Dquality%20medical%20imaging.

Deep Learning Use Cases and their Associated Architectures

To aid users in selecting a proper neural network architecture, we have listed below common use-cases and their associated neural network architecture.

Common architectures for NLP use-cases:

Sentiment analysis	LSTM, transformer
Text summarization	RNN, transformers, mamba
Machine Translation	RNN, transformers, mamba
Text generation	LSTM, transformers, mamba
QA system	Transformers
Text classification	Transformers, LSTM, GRU
LLMs	Transformers, MoE

Table 3: NLP use-cases

Computer vision applications:

Object detection	CNN, ResNet
Image classification	CNN
OCR	RNN
Video classification	LSTM
Image denoising	autoencoder
Image compression	autoencoder
Image generation (text to image)	GANs, diffusion, VAE
Image upsampling	GANs
Image segmentation	ResNet, SAM

Table 4: Computer vision applications

Audio applications:

Speech recognition	LSTM, transformers
Video classification	LSTM

Table 5: Audio applications

Biology applications:

Protein Folding	Alphafold, Deep residual network
Antigen–antibody affinity prediction	ConvNeXt network
Drug Target Interaction prediction	GNNs
Molecular Properties Prediction	GNNs

Table 6: Biology applications

Statistics applications:

Time series prediction	RNN, LSTM, GNNs
Dimension reduction	autoencoder
Anomaly detection	Autoencoder, VAE

Table 7: Statistics applications

Chapter 6: Model Repositories

Model repositories serve as centralized platforms where developers and researchers can discover, share, and deploy pre-trained ML models. These repositories provide a vast collection of models, from image recognition and NLP to generative models, allowing users to save time and computational resources by leveraging models that have already been trained on large datasets. The introduction of these model hubs has significantly streamlined the process of experimenting with and deploying ML models, making cutting-edge AI more accessible to a wider audience. In the following, we highlight some of the most widely used repositories today.

Huggingface

Hugging Face is a popular ecosystem that offers machine learning tools, libraries, and services for development and deployment, connecting cutting-edge ML research with real-world applications. As of 2024, Hugging Face hosts over 900k models, 200k datasets, and 300k demo applications (known as spaces). It supports a wide range of model architectures, such as transformers, diffusion models, and more, while enabling seamless integration with popular ML frameworks such as PyTorch and TensorFlow. All resources are open-source and publicly available as GitHub repositories, enabling anyone to explore and leverage ML technologies. Repositories in this context are structured spaces where code and assets are stored, facilitating individual and collaborative work while promoting sharing within the community.

There are three key features of Hugging Face Hub:

- Model Hub: A repository of pre-trained models designed for various tasks, including NLP, computer vision, and audio processing.
- Datasets Hub: A vast collection of datasets curated for training and evaluating ML models.
- Spaces: A feature that allows users to create, host, and share interactive ML applications.

The Model Hub

The Model Hub is a library of a wide range of pretrained ML models. The Hub allows users to search for models by keywords, tags, or categories such as task type (e.g., multimodal, computer vision, NLP, audio), libraries (e.g., PyTorch, TensorFlow), datasets, languages, and licenses (e.g., Apache-2.0, MIT). Each model in the Hub is accompanied by a *Model Card*, which is the first tab of each model page. These cards provide detailed documentation, including the model's purpose, architecture, training datasets, evaluation metrics, and usage examples. This makes it easier to assess the model's suitability for the user's needs. Additionally, users can see download statistics and star ratings to gauge popularity and reliability on the model card. The second tab of each model, *Files and Versions*, contains the specific files of the model and all previous versions, in a similar format as a GitHub repository. Many models are contributed by the community, including individual researchers and organizations. Finally, discussions and pull requests can be found in the third tab, *Community*. The *Community* tab fosters collaboration and feedback, allowing users to engage in discussions, report issues, and contribute to the model's development.

For certain models, the Hub includes interactive widgets that allow users to test model outputs directly in the browser by inputting sample data. It is located under "Inference API" on the *Model Card*. For example, try the widget on BERT here https://huggingface.co/google-bert/bert-base-uncased. You can also review all the widgets on Hugging Face in one place (https://huggingface-widgets.netlify.app/).

The Datasets Hub

The Datasets Hub is a comprehensive resource for discovering and utilizing datasets tailored for ML projects. It offers datasets in diverse formats and for a wide range of tasks such as text analysis, computer vision, and audio processing. Similar to the Model Hub, the Datasets Hub allows users to search for datasets by keywords or apply filters based on modality (e.g., 3D, audio, image, tabular, text, time-series, video), size (from <1K to >1T), format (e.g., json, csv, parquet), libraries (e.g., croissant, polars), task type, language, or license. Each dataset is paired with a *Dataset Card* that provides important details such as the dataset's purpose, source, licensing terms, preprocessing steps, and usage examples. Similar to the *Model Card*, users can quickly understand the dataset's relevance and applicability via the *Dataset Card* as well as the ratings and download statistics. The Datasets Hub hosts contributors from both the Hugging Face team and the community. Many datasets include preprocessing scripts to simplify integration into ML workflows. These scripts can handle tasks like tokenization, splitting, or formatting. The data sets are often explicitly aligned with specific tasks or benchmarks, making it easier to use them directly for training or evaluation.

Many datasets feature a built-in *Dataset Viewer*, which allows users to explore the dataset directly in the browser. This tool provides a tabular view of the data, enabling users to inspect samples, metadata, and structure without downloading the dataset. Most Hugging Face datasets include basic column descriptions (via features) and sometimes explanations of codes or labels in the dataset card (via README). However, full data dictionaries or codebooks are not always provided, so you may need to check the original source or linked paper for detailed meanings. Additionally, not all datasets have a viewer; it depends on the dataset's format and size. Take a look at an example *Dataset Viewer* here: https://huggingface.co/datasets/huggingface/documentation-images.

The Spaces Hub

The Spaces Hub is a feature that allows developers and researchers to quickly build and share ML applications, including models, demos, and even full web applications. They provide a user-friendly interface for deploying models, especially for showcasing the capabilities of ML such as NLP and computer vision. Spaces are powered by Gradio and Streamlit, which are Python libraries for creating interactive, dynamic web-based applications with a simplified process of building user interfaces.

The benefit of Spaces is that they enable a simple, streamlined way to test models directly from a browser. They also allow for collaboration, where others can interact with your model and give feedback. There are two main ways to use Spaces:

1. Public Spaces are open for anyone to see and use. They can be shared freely with the community, making it ideal for showcasing new models, research projects, or prototypes.

2. Private Spaces are only accessible to invited users. They are useful for teams to work on models in a controlled environment before sharing them more broadly.

Spaces can be customized to meet the needs of a variety of projects. Whether it is a simple demo or a more complex app, a range of widgets can be utilized such as text inputs, sliders, and buttons.

Getting Started

To get started with Hugging Face model repositories, begin by exploring the main Hugging Face website and the Model Hub. Whether you're looking to integrate a model into your application, fine-tune it for a specific task, or train one from scratch, Hugging Face provides extensive documentation and tutorials to guide you (e.g., pipeline tutorial, Model Hub tutorial). For example, one of the key resources is the Transformers library, which offers powerful tools for working with NLP and beyond. By following their step-by-step guides and leveraging an active community (e.g., the Transformers community resources), you'll quickly gain the skills and knowledge to harness the power of Hugging Face for your projects. Additionally, there are numerous external resources, such third-party tutorials and YouTube videos, that provide further insights and practical demonstrations to help you master the platform.

References:

1. https://huggingface.co/docs/huggingface_hub/main/en/quick-start
2. https://www.mindfiretechnology.com/blog/archive/an-introduction-to-hugging-face-and-their-pipelines/
3. https://transformersbook.com/
4. https://www.youtube.com/watch?v=QEaBAZQCtwE

TensorFlow Hub

TensorFlow Hub is an open-source platform designed to simplify the sharing and reuse of ML models. It offers a wide variety of ready-to-use models for tasks like image recognition, NLP, and more. Same as the Hugging Face platform, this repository is suitable for different levels of expertise and can save you time and computational resources, as they are pre-trained on large datasets and optimized for various applications. Users can deploy famous trained models such as BERT in just a few lines of code (accessible online in Google Colab or downloadable as a Jupyter Notebook). All models and tutorials are also available on GitHub. What makes this hub stand out is that it is seamlessly integrated with the TensorFlow ecosystem (e.g., TensorFlow and Keras models), making it a natural choice for developers already working within this framework. Although TensorFlow Hub provides a diverse range of models, it is more for general purposes, often featuring TensorFlow-optimized versions of widely used models. To get started, begin with the tutorials provided by TensorFlow Hub.

References:

1. https://www.tensorflow.org/hub

Pytorch Hub

PyTorch Hub is another curated repository of pre-trained models designed to accelerate the development of ML applications. It offers an accessible platform for exploring state-of-the-art models in fields like NLP, computer vision, and audio using the PyTorch framework. Similar to TensorFlow Hub, PyTorch Hub allows developers to quickly download, load, and fine-tune models with seamless integration into PyTorch workflows. By providing tools and resources that align closely with the PyTorch ecosystem, the PyTorch Hub empowers developers to build and iterate on advanced ML solutions efficiently. The platform is particularly well-suited for research and experimentation, as it emphasizes reproducibility and flexibility.

Reference:

https://pytorch.org/hub/

NVIDIA Models

NVIDIA's Model Hub offers a comprehensive collection of pre-trained AI models optimized for deployment across various GPU infrastructures. These models cater to diverse applications, including NLP, computer vision, and scientific research. Developers can integrate these models into their workflows using NVIDIA's Inference Microservices (NIM), a lightweight, scalable tool for deploying and serving AI models in production environments. The platform features models such as Llama-3.2-NV-EmbedQA-1B-V2 for multilingual question-answering retrieval, Audio2Face-3D for real-time facial animation from audio inputs, and FourCastNet for global atmospheric predictions. Similar to other model hubs, each model has a *Model Card*, providing critical technical and non-technical details. Some models allow users to test the model directly through a hosted API endpoint without needing to download or deploy it locally, via the *"Try API"* tab. Some models provide a pre-built Docker container with the model and its dependencies, enabling seamless deployment on local or cloud-based systems. A specific model might also have a *System Card* if its deployment or use requires interaction with a larger, complex system. For example, one of NVIDIA's internal frameworks, Cosmos, serves as a backend system that supports the development and deployment of the models found on NVIDIA's platforms and GPU clouds. Although NVIDIA's models are designed to run best on GPUs, especially NVIDIA's own hardware, they can sometimes run on CPUs with limited performance and functionality. In summary, NVIDIA's model catalog empowers developers to accelerate AI application development by leveraging state-of-the-art models tailored for high-performance GPU environments.

References:

1. https://catalog.ngc.nvidia.com/models
2. https://build.nvidia.com/explore/discover
3. https://build.nvidia.com/nvidia/llama-3_2-nv-embedqa-1b-v2
4. https://build.nvidia.com/nvidia/audio2face-3d
5. https://build.nvidia.com/nvidia/fourcastnet

Azure AI Model Catalog

The Azure AI Model Catalog is a centralized hub that provides access to a wide variety of ready-to-use, state-of-the-art AI models from Microsoft and its partners. Designed to accelerate AI integration, the catalog allows developers and data scientists to browse, test, and deploy models across tasks like text summarization, object detection, translation, speech recognition, and more --- all within the Azure cloud ecosystem. For example, users can quickly fine-tune OpenAI's GPT models for industry-specific use cases, deploy Meta's Llama 2 for chat applications, or integrate NVIDIA's vision models into real-time video analytics. Azure's AI Model Catalog powers Model-as-a-Service (MaaS) by letting users access and fine-tune models directly through Azure endpoints without needing to provision or manage back-end operations. The catalog streamlines experimentation and production at scale with seamless deployment to Azure Machine Learning and Azure AI Studio. It is built for enterprise use, offering robust security, governance, and compliance features to support regulated industries and mission-critical applications.

References:

1. https://azure.microsoft.com/en-us/products/ai-model-catalog
2. https://huggingface.co/blog/microsoft-collaboration
3. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-models-from-huggingface?view=azureml-api-2
4. https://azure.microsoft.com/en-us/products/machine-learning
5. https://azure.microsoft.com/en-us/blog/product/azure-ai-studio/
6. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-online-endpoints?view=azureml-api-2&tabs=cli

Summary

Hugging Face Model Hub, TensorFlow Hub, Pytorch Hub, NVIDIA Models, and Azure Catalog all serve similar purposes in terms of providing access to pre-trained ML models, which can be easily fine-tuned or adapted through transfer learning to suit specific domains, enabling more efficient customization for specialized tasks. These pre-trained models are invaluable for several purposes:

1. Quick Prototyping: Leverage pre-trained models to implement ML solutions with minimal coding effort.

2. Learning and Experimentation: Access a vast array of datasets and pre-built models to enhance your understanding of ML techniques.

3. Collaboration: Share models and code with the global AI community to receive feedback and refine your skills.

4. Real-World Applications: Deploy models in production environments using each platform's robust tools and services.

Each hub caters to different user needs based on their preferred frameworks, hardware requirements, and project focus. For flexibility and variety, Hugging Face is unmatched and currently the most popular repository, while TensorFlow Hub and PyTorch Hub excel in their respective ecosystems. NVIDIA Model Hub is best for production scenarios involving GPU-optimized applications. Azure AI Model Catalog stands out for enterprise-grade deployments, offering seamless integration with Azure services, access to models from other platforms, and built-in tools for responsible AI and model evaluation. Other model repositories include Model Zoo and Caffe, which are relatively lightweight and fast but lack the polished ecosystem of many state-of-the-art models.

The table below summarizes the model repositories discussed so far in this chapter.

Comparison of Model Repositories (Table 8):

	Hugging Face	TensorFlow Hub	Pytorch Hub	NVIDIA Models	Azure AI Model Catalog
Framework Focus	Multi-framework	TensorFlow	Pytorch	NVIDIA	Multi-framework
Model Variety	High	Moderate	Moderate	Moderate	High
Strengths	Largest variety of models and data. Cross-framework compatibility. Extensive community contributions	Seamless integration with TensorFlow/Keras; Community-driven contributions of models. Comprehensive documentation and tutorials	Seamless integration with Pytorch. Community-driven contributions of models. Comprehensive documentation and tutorials	High-performance AI. Designed for use on NVIDIA hardware	Deep integration with Azure ML and cloud services. Enterprise-ready with security, governance, and compliance. Easy deployment and endpoint creation
Limitations	Learning curve to navigate through the Hugging Face ecosystem	Limited cross-framework compatibility. Smaller model selection compared to Hugging Face	Limited cross-framework compatibility. Smaller model selection compared to Hugging Face	Narrow focus on NVIDIA hardware users. Smaller model selection compared with other hubs	Less portable outside Azure (vendor lock-in)

Chapter 7: Parameter Tuning

Parameter tuning is an essential step in building effective deep learning models. Choosing the right hyperparameters can significantly improve the model's performance. In this section, we will introduce some common parameter tuning approaches in deep learning, along with some Python/R code examples.

Before we go into details, we would like to first provide a decision graph to help illustrate under which scenarios you should use different type of hyperparameter tuning approaches:

Scenario	Grid Search	Random Search	Bayesian Optimization
Small search space	Recommended	Optional	Optional
Large search space	Not recommended	Recommended	Recommended
Limited time budget	Not recommended	Recommended	Recommended
Limited computational resources	Not recommended	Recommended	Recommended
Searching for global optima	Optional	Optional	Recommended
High-dimensional hyperparameters	Not recommended	Recommended	Recommended
Continuous hyperparameters	Not recommended	Optional	Recommended
Noisy objective function	Not recommended	Optional	Recommended
Parallelization	Optional	Recommended	Recommended
History of prior evaluations	Not applicable	Not applicable	Recommended
Adaptive exploration-exploitation trade-off	Not applicable	Not applicable	Recommended

Table 9: Comparison of tuning approaches

Grid Search is suitable when you have a limited number of hyper-parameters, and their performance characteristics are known. It is feasible to try all combinations and find the best configuration. It's the most exhaustive but can be computationally expensive when the search space is large.

Random Search is suitable when you have a large search space and it's difficult to determine the range or relationships between hyperparameters. It provides a more efficient search compared to Grid Search by randomly sampling configurations, but it doesn't guarantee finding the optimal solution.

Bayesian Optimization is suitable when you have a large search space and it's difficult to determine the range or relationships between hyperparameters. It builds a probabilistic model of the objective function and intelligently selects configurations to evaluate. It can efficiently find optimal configurations with fewer iterations compared to random or grid search.

Brute Force Grid Search

Grid search is a simple and widely used approach for hyperparameter tuning. It exhaustively searches through a specified parameter space by trying all possible combinations of the given hyperparameters. Scikit-learn provides the GridSearchCV class for performing grid search with cross-validation. See more details here: https://towardsdatascience.com/grid-search-in-python-from-scratch-hyperparameter-tuning-3cca8443727b

Random Search

Random search is a more efficient alternative to grid search. Instead of exhaustively searching through all possible combinations, random search samples a random subset of the hyperparameter space. Scikit-learn provides the RandomizedSearchCV class for performing random search with cross-validation. See more details here: https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

Bayesian Search

Bayesian search is a probabilistic approach that models the unknown objective function (model performance) as a Gaussian process. It iteratively updates the beliefs about the objective function using Bayesian inference and then selects the next point to sample based on acquisition functions. Scikit-optimize provides the BayesSearchCV class for performing Bayesian search with cross-validation. See more details here: https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html

The mlrMBO package in R also provides a framework for parameter tuning using Bayesian optimization. See more details here: https://cran.r-project.org/web/packages/mlrMBO/index.html

Parameter tuning for deep learning models with Optuna in Python

Optuna (for details, see https://optuna.org/) is a popular hyperparameter optimization library (implemented in PyTorch) to tune hyperparameters of a deep learning model. Optuna uses a more advanced approach compared to grid search or random search for hyperparameter optimization. It employs a Tree-structured Parzen Estimator (TPE), a Bayesian optimization method, to efficiently explore the hyperparameter search space. TPE builds a probabilistic model to estimate the distribution of hyperparameters that yield better results and guides the search towards promising areas of the search space. This approach is more efficient than grid search or random search, as it uses the information from previous trials to decide the next set of hyperparameters to evaluate.

Parameter tuning for deep learning models with GenSA in R

One popular optimization technique used for hyperparameter tuning is simulated annealing (SA). SA is inspired by the annealing process in metallurgy and mimics the cooling process of a material to reach a low-energy state. It has been successfully applied to various optimization problems, including parameter tuning in deep learning. In this section, we introduce the Generalized Simulated Annealing (GenSA) method, which is an extension of the SA algorithm (For details, see https://journal.r-project.org/archive/2013/RJ-2013-002/RJ-2013-002.pdf). GenSA is a derivative-free optimization algorithm that uses a modified version of the Metropolis-Hastings algorithm to explore the parameter space. It introduces several enhancements to the original SA algorithm, such as dynamic rescaling, parallel tempering, and adaptive cooling schedules, to improve its convergence and exploration capabilities.

Comparison of Search Methods and Implementation Tools

	PROS	CONS
Brute force grid search	• Can find optimal hyper-parameters with high precision	• Computationally expensive and time-consuming, especially for large parameter spaces
Random search	• Less computationally expensive and faster than grid search • Can achieve similar performance with fewer iterations	• Less precise than grid search
Bayesian search (See more details here: https://arxiv.org/pdf/1012.2599)	• Can find optimal hyper-parameters more efficiently than grid and random search • Incorporates prior knowledge and uncertainty about the objective function	• More complex than grid and random search
Optuna in Python	• Efficient optimization using Bayesian optimization with TPE • Easy integration with popular machine learning and deep learning frameworks • Supports pruning, parallelization, and visualization • Flexible in defining search spaces, objective functions, and optimization algorithms	• More complex than simpler methods like grid and random search • Less suitable for small search spaces • Computationally expensive for large search spaces and complex models
GenSA in R	• Robustness: GenSA is robust in finding global optima in complex search spaces • Exploration-exploitation balance: it efficiently explores the parameter space while focusing on promising regions • Derivative-free optimization: it works without requiring gradient information, making it applicable to a wide range of problems	• Computationally expensive: the GenSA algorithm can be computationally expensive, especially for large-scale optimization problems or when the objective function evaluation is time-consuming • Sensitivity to initial values: the performance of GenSA can be sensitive to the initial parameter values. If the initial values are far from the optimal solution, it may take longer for the algorithm to

		converge, or it may converge to suboptimal solutions. • Limited scalability: while GenSA performs well for many optimization problems, it may face challenges when applied to highly complex or high-dimensional search spaces

Table 10: Comparison of search methods

Chapter 8: Model Evaluation

Model evaluation is a critical step in the machine learning and deep learning process. It allows developers to measure the performance of their models, identify issues like overfitting and underfitting, and make improvements to achieve better results. This section introduces the best practices for conducting model evaluation, including various evaluation metrics, loss functions, the detection of overfitting and underfitting, and possible solutions. We will also discuss the role of subject matter experts in the validation process and the risks associated with subjectivity.

Model Evaluation Metrics

There are several metrics to evaluate the performance of a model, including loss, accuracy, F1 score, precision, and recall. These metrics help us understand different aspects of a model's performance. Before we go into details, we would like to first provide a decision graph to help illustrate under which scenarios you should use the different types of model evaluation metrics:

Scenario	Regression Metrics	Binary Classification Metrics	Multiclass Classification Metrics
General performance evaluation	Mean Squared Error (MSE)	Accuracy	Accuracy
Emphasizing larger errors	Mean Squared Error (MSE)	F1-score	Macro-averaged F1-score
Penalizing overestimation	Quantile Loss (https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/)	Precision	Macro-averaged Precision
Penalizing underestimation	Root Mean Squared Logarithmic Error (RMSLE, see details: https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/)	Recall	Macro-averaged Recall
Interpretability	R-squared (R^2)	Area Under the ROC Curve (AUC-ROC)	One-vs-One ROC AUC
Imbalanced data	Not applicable	F1-score, Precision-Recall Curve (PRC)	Weighted F1-score, per-class metrics
Probability calibration	Not applicable	Brier score (see details: https://en.wikipedia.org/wiki/Brier_score), Log-loss	Log-loss, Brier score (multiclass extension)

Table 11: Comparison of Regression, Binary Classification, and multi-class metrics

For **binary class classification**, when you choose the proper performance metrics, there are multiple considerations:

Consideration	Yes	No
Is the dataset balanced (roughly equal number of samples in each class)?	Accuracy, Sensitivity, Specificity, and ROC-AUC	Precision-Recall Curve, Average Precision, or Balanced Accuracy
Are the classes mutually exclusive (samples belong to only one class)?	Accuracy, Precision, Recall, F1-Score, and Confusion Matrix	Hamming Loss, Exact Match Ratio, or Micro/Macro-Averaged metrics
Are there specific concerns regarding false positives/negatives or class imbalance?	For false positives/negatives: Examine Precision, Recall, and F1-Score. For class imbalance: Consider using Weighted or Balanced Accuracy	Accuracy as the primary metric and consider additional metrics based on the specific problem context.

Table 12: Binary classification considerations

For multi-class classification, a similar strategy applies, but with the necessity to binarize the output. Metrics for each individual class label can be calculated, and specifically, the Micro and Macro F1 scores offer different approaches to compute the F1-score in multiclass classification problems, which is the harmonic mean of precision and recall. Macro F1 calculates the F1-score independently for each class and then averages them, making it suitable when all classes are of equal importance. Micro F1, in contrast, aggregates the true positives, false positives, and false negatives across all classes before computing the F1-score, making it more appropriate for imbalanced class distributions and for assessing overall performance.

See the details of the most commonly used performance metrics for the evaluation of classification models as follows:

- Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

- Precision (Positive Predictive Value)

$$Precision = \frac{TP}{TP + FP}$$

where TP is the number of true positives and FP is the number of false positives.

- Recall (Sensitivity or True Positive Rate)

$$Recall = \frac{TP}{TP + FN}$$

where TP is the number of true positives and FN is the number of false negatives.

- Specificity (True Negative Rate)

$$Specificity = \frac{TN}{TN + FP}$$

where TN is the number of true negatives and FP is the number of false positives.

- F1 Score

$$F_1 = 2 \times \frac{Precision \ \times Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

where Precision is the precision value and Recall is the recall value.

- Negative Predictive Value (NPV)

$$NPV = \frac{TN}{TN + FN}$$

where TN is the number of true negatives and FN is the number of false negatives.

Loss Functions

A loss function in deep learning is a measure of the model's performance by quantifying the difference between predicted and true values. It guides the model's learning process by minimizing the error and updating its parameters. On the other hand, performance metrics evaluate the model's quality and effectiveness, providing additional insights such as accuracy, precision, recall, F1 score, and AUC-ROC. While loss functions are optimized during training, performance metrics are computed on separate evaluation datasets to assess the model's generalization and to compare different models.

In this section, we will discuss some decision rules for **choosing Loss Functions** for deep learning models.

Classification Deep Learning Models

	Balanced scenario (approximately equal number of samples in each class)	Imbalanced scenario (significantly unequal number of samples in each class)
Binary class	1. Binary Cross-Entropy Loss (Log Loss): Suitable for problems with a binary output and when the model outputs class probabilities	1. Weighted Binary Cross-Entropy Loss: Assign higher weights to the minority class to address the class imbalance problem. 2. Focal Loss: Introduce a modulating factor to downweight the easy examples and focus on hard examples, helping to alleviate the impact of class imbalance
Multi-class	1. Categorical Cross-Entropy Loss: Suitable when the model outputs	1. Class Weighted Categorical Cross-Entropy Loss: Assign higher weights

	class probabilities and the classes are mutually exclusive	to the minority classes to address the class imbalance problem. 2. Focal Loss: Introduce a modulating factor to downweight the easy examples and focus on hard examples, helping to alleviate the impact of class imbalance

Table 13: Classification scenarios and their loss functions

If the problem is **multi-label classification** (instances can belong to multiple classes simultaneously), consider using Binary Cross-Entropy Loss: Treat each class as a separate binary classification problem and use binary cross-entropy individually for each class.

Regression Deep Learning Models

Decision Rules for Choosing Loss Function for Regression Models:

Mean Squared Error (MSE)	1. MSE is a common loss function for regression problems and is suitable for most scenarios. 2. Use MSE as a default choice for regression unless there are specific requirements.
Mean Absolute Error (MAE)	1. MAE calculates the average absolute difference between the predicted and actual values. 2. MAE is less sensitive to outliers compared to MSE, making it suitable when outliers are present or when the absolute error is more important than the squared error.
Huber Loss (See more details here: https://en.wikipedia.org/wiki/Huber_loss)	1. Huber loss combines properties of MSE and MAE. 2. It is less sensitive to outliers like MAE while having quadratic behavior for small errors like MSE. 3. When the dataset contains outliers and you want a balance between the robustness of MAE and the smoothness of MSE.
Log-Cosh Loss (See more details here: https://arxiv.org/pdf/2208.04564)	1. Another option that combines properties of MSE and MAE. 2. Similar characteristics to Huber loss but with a smoother transition. 3. It can be suitable when the dataset contains outliers, and you want a smooth loss function.
Custom Loss functions	1. Depending on the specific requirements of your regression problem, you can create custom loss functions tailored to the task. 2. Can incorporate domain-specific knowledge or address specific challenges in the data.

Table 14: Regression Loss Functions

As we discussed above, loss is a measure of the difference between the predicted values and the actual values. The goal is to minimize the loss function during training. Different types of loss functions are more suitable for different types of problems. Here, we will discuss four common loss functions: Mean Squared Error (MSE), Mean Absolute Error (MAE), Cross-Entropy Loss, and Log Loss.

Mean Squared Error (MSE) is expressed as $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$,where y_i is the true label, $\hat{y}_i$ is the predicted value, for i = 1 to n (number of samples). MSE computes the average squared difference between predicted and actual values, with a smaller value indicating a more accurate model.

Mean Absolute Error (MAE) is given by $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|$, it uses the same parameters as MSE but instead calculates the average absolute difference. MAE is particularly effective in regression tasks, being less sensitive to outliers compared to MSE.

Cross-Entropy Loss is vital for classification tasks, and its formula is $CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij})$, where y_{ij} is the indicator function (1 if sample i belongs to class j and 0 otherwise), p_{ij} is the predicted probability of sample i belonging to class j, n is the number of samples, and m is the number of classes. Cross-Entropy Loss is widely used for classification tasks. It measures the performance of a classification model by calculating the difference between the predicted probabilities and the actual class labels. For binary classification, the loss function is called Binary Cross-Entropy Loss, and for multi-class classification, it is called Categorical Cross-Entropy Loss.

Log Loss, or Binary Cross-Entropy Loss, is tailored for binary classification tasks,
Log Loss $= -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$, where y_i is the true label (0 or 1), p_i is the predicted probability of the positive class (between 0 and 1), and n is the number of samples. Log Loss, or Binary Cross-Entropy Loss, is used for binary classification tasks. It calculates the loss between the predicted probabilities and the actual binary labels.

Detection of Overfitting and Underfitting

In the context of deep learning, neural network models often face the challenges of overfitting and underfitting during training. Overfitting occurs when a model becomes overly complex and starts memorizing the training data rather than learning to understand the underlying patterns or relationships. This leads to excellent performance on the training data but poor generalization of new, unseen data. The telltale signs of overfitting include a model performing markedly better on training data than on validation or test data, exhibiting low training loss but high validation loss, and capturing noise or random fluctuations in the training data. To detect overfitting, one can plot learning curves to observe the divergence between training and validation loss, or one can evaluate the model on unseen data to assess its generalization capability.

Conversely, underfitting is when a neural network model is too simplistic, failing to grasp the complexities in the data, and showing subpar performance on both training and validation data. Characteristics of underfitting are poor performance across both training and validation datasets, high losses in both cases, and an inability to discern patterns in the data. Detecting underfitting involves similar methods as overfitting: analyzing learning curves where both training and validation losses remain high and show minimal improvement, and scrutinizing performance metrics like accuracy. Consistently low performance on training data is a red flag for underfitting, suggesting the need for a different architectural approach.

Solutions for Overfitting and Underfitting

In deep learning, solving overfitting involves several strategies. Increasing the training data by obtaining more labeled data can expand the training set, allowing for better generalization and reducing overfitting. Data augmentation techniques such as random rotations, translations, or flips can artificially enlarge and vary the training data, aiding the model in learning more robust features. Regularization is key, with techniques such as L1 and L2 regularization, which penalize large weights in the loss function and reduce dependency on specific features. Dropout regularization involves randomly deactivating neurons during training, encouraging diverse representations, while early stopping halts training when validation loss increases, finding the optimal generalization point. Reducing model complexity by simplifying architecture, like fewer layers or neurons, also helps in preventing overfitting.

Conversely, solving underfitting in deep learning might require increasing model complexity. This could involve adding more layers or neurons or employing non-linear activations to capture complex data patterns. Incorporating domain knowledge through feature engineering provides more meaningful data representations, enhancing model performance. Experimenting with different architectures, like convolutional neural networks (CNNs) for image data or recurrent neural networks (RNNs) for sequential data, can offer better fits. Extending training time allows the model to learn from data more comprehensively, especially when signs of underfitting are present. Hyperparameter tuning, including adjustments to learning rate, batch size, optimizer choice, or activation functions, is crucial for fine-tuning performance.

Addressing overfitting and underfitting is iterative, often requiring a combination of techniques and adjustments for optimal model performance. Continuously monitoring and modifying the model based on its performance with unseen data is essential for effective mitigation of these issues in deep learning models.

Validation by Subject Matter Experts

Leveraging Subject Matter Experts (SMEs) to evaluate Deep Learning and AI models in the pharmaceutical industry is essential to ensure the validity, accuracy, and relevance of the models to the specific domain. SMEs bring invaluable domain knowledge that can significantly enhance the development and evaluation of AI models tailored to the unique challenges and requirements of the pharma industry.

To effectively utilize SMEs, their input should be sought at different stages of model development, including data curation, feature selection, model training, and validation. During the data curation stage, SMEs can help identify the most relevant and high-quality data sources, ensuring that the model is trained on representative and diverse data. This is particularly important in the pharma industry, where data quality and integrity are critical for the success of AI-driven drug discovery and clinical trials.

SMEs can guide the selection of appropriate features and variables to include in the model, identifying key relationships in the data that might not be evident to data scientists. This ensures that the AI models are tailored to capture the intricacies of the pharmaceutical domain, such as complex molecular interactions, patient populations, and regulatory requirements.

During the model training and validation stages, SMEs can provide insights into the most appropriate performance metrics and evaluation criteria, ensuring that the model's predictions align with the underlying scientific principles and industry expectations. They can help in interpreting the model outputs, identifying potential areas of improvement, and validating the results against established benchmarks.

Advantages of involving SMEs in the evaluation process include increased confidence in the model's accuracy and relevance, improved model explainability, and better alignment with industry regulations and standards. Involving SMEs can also foster a sense of ownership and trust among stakeholders, leading to better acceptance and adoption of AI-driven solutions in the pharma industry.

However, there are also potential cons and pitfalls. SMEs may have limited availability, which could lead to delays in the model development and evaluation process. They may also not be well-versed in AI and deep learning concepts, which could lead to communication challenges and slower progress. Additionally, relying heavily on expert input might introduce biases and limit the potential for novel insights or discoveries that AI models can offer.

To overcome these challenges, organizations should invest in educating SMEs about AI and deep learning concepts and foster a collaborative environment between data scientists and domain experts. This can be achieved through cross-functional teams, joint workshops, and regular knowledge-sharing sessions. It is also crucial to ensure a balanced approach to model evaluation, considering both expert input and data-driven insights to maximize the potential benefits of AI in the pharmaceutical industry.

Lastly, organizations should be aware of the potential risks associated with over-reliance on SMEs and strive to maintain a healthy balance between human expertise and data-driven AI insights. By combining the strengths of both SMEs and AI models, the pharmaceutical industry can unlock the full potential of AI-driven innovation, leading to faster drug discovery, improved clinical trial outcomes, and ultimately, better patient care.

Chapter 9: Data augmentation

Data augmentation is a technique widely employed in machine learning to artificially increase the diversity of training data. It involves creating variations of the original data by applying a set of transformations. This section explores data augmentation methods in the context of Natural Language Processing (NLP) and Computer Vision (CV).

The purpose of this section is to provide a comprehensive overview of common data augmentation methods in NLP and CV, highlighting their pros and cons. By understanding these techniques, practitioners can make informed decisions about which augmentation strategies to apply in their specific applications.

Data Augmentation in NLP

In the realm of Natural Language Processing (NLP), data augmentation can be achieved through various methods. Synonym replacement is one such method, where words in a sentence are replaced with their synonyms, maintaining the original meaning. Mask language modeling, another technique, involves inserting random words into a sentence, increasing its length and complexity. For instance, in the sentence "The quick brown fox jumps over the lazy dog," replacing a word with a mask might result in "The quick brown [MASK] jumps over the lazy dog," and a language model could then predict the missing word. Random deletion and text shuffling are also employed, with the former removing random words to create a concise version and the latter rearranging words to form new structures. Back translation is another strategy, involving translation between languages and back to the original to create variations. See some code examples here: https://github.com/makcedward/nlpaug

These NLP augmentation methods come with pros and cons. The advantages include enhanced model robustness and generalization, reduced risk of overfitting, improved performance in low-resource languages or domains, and better handling of out-of-vocabulary words. However, they also have downsides, such as the risk of introducing grammatical errors or nonsensical text, potential unsuitability for specific tasks like translation or summarization, and increased computational costs due to larger datasets. Below is a table to summarize Pros and Cons for NLP data augmentation:

Pros	Cons
• Enhanced model robustness and generalization. • Reduced risk of overfitting. • Improved performance on low-resource languages or domains. • Better handling of out-of-vocabulary words.	• Risk of introducing grammatical errors or nonsensical text. • May not be suitable for all NLP tasks (e.g., translation or summarization). • Increased computational cost due to increased dataset size. • Medical/Legal domains may have special considerations to retain meaning during augmentation

Table 15: Pros and Cons for NLP data augmentation

Data Augmentation in CV

Similarly, in Computer Vision (CV), data augmentation encompasses techniques like image rotation, which involves rotating an image by a random angle to aid object recognition from various perspectives. Image flipping, either horizontally or vertically, creates mirror images, enhancing dataset diversity. Zooming and cropping allow models to focus on specific image regions, while color jitter introduces random color variations to bolster model robustness against lighting and color changes. Gaussian noise and adding random pixel-level noise simulates real-world imperfections. See some code examples here: https://github.com/AISangam/Image-Augmentation-Using-OpenCV-and-Python

The pros and cons of CV data augmentation parallel those in NLP to some extent. The benefits include improved model generalization, enhanced performance in object recognition and classification tasks, and increased robustness against variations in lighting, orientation, and noise. However, the drawbacks involve the risk of generating unrealistic images, heightened computational costs due to larger training datasets, and limited effectiveness for specific image types, such as medical images. Below is a table of the Pros and Cons for CV data augmentation:

Pros	Cons
• Improved model generalization. • Enhanced performance on object recognition and classification tasks. • Increased robustness to variations in lighting, orientation, and noise.	• Risk of generating unrealistic images. • Increased computational cost due to larger training datasets. • Limited effectiveness for certain types of images (e.g., medical images).

Table 16: Pros and Cons of CV data augmentation

Data Augmentation Best Practices

In the area of data augmentation, several best practices and considerations emerge. First and foremost, the selection of the most appropriate augmentation techniques should be driven by the unique characteristics of the dataset, the specific task at hand, and domain knowledge. It is advisable to experiment with various augmentation methods to determine which ones yield the most significant improvements, but please keep in mind that some augmentations may not be appropriate based on which domain you are studying. For example, in the medical domain, using the wrong synonym could lead to misinterpretation and potentially dangerous outcomes. Additionally, fine-tuning augmentation hyperparameters, such as rotation angles, noise levels, or the extent of text modifications, can significantly impact the augmentation process's effectiveness. When implementing data augmentation, a rigorous evaluation and validation process should be in place to ensure that the augmented datasets maintain data quality and do not introduce biases that could affect model performance. Moreover, ethical considerations must be at the forefront, particularly in sensitive domains like healthcare, to ensure that augmented data does not compromise privacy or fairness, adhering to responsible and ethical AI practices.

Data augmentation is a crucial technique for improving model generalization and robustness in NLP and CV. This white paper has provided an overview of common augmentation methods, their pros and cons, and their applications in both domains. Understanding and applying data augmentation effectively can lead to more reliable and accurate deep learning models in various applications.

Chapter 10: Improving Model Performance of Pretrained Models

Transfer learning

Transfer learning is the idea of utilizing a pretrained model in one domain and transferring that knowledge to a related domain. Since we expect low level features will be beneficial for both domains, transfer learning is typically achieved by utilizing the first layers of the pre-trained model (representing low level features of an images) in the new transfer learning model. However, if the two domains are not related or low-level features are not in common for the two domains, transfer learning won't be that useful.

Transfer learning has some key strengths compared to training the whole model. First, samples required for training in the related domain are typically much less than the samples used for training the pre-trained model. Additionally, the training time needed for transfer learning is typically less than training time needed for the whole model. Lastly, the model performance on the related domain can improve after applying transfer learning.

Transfer learning architecture

In detail, we illustrate the architecture of transfer learning below:

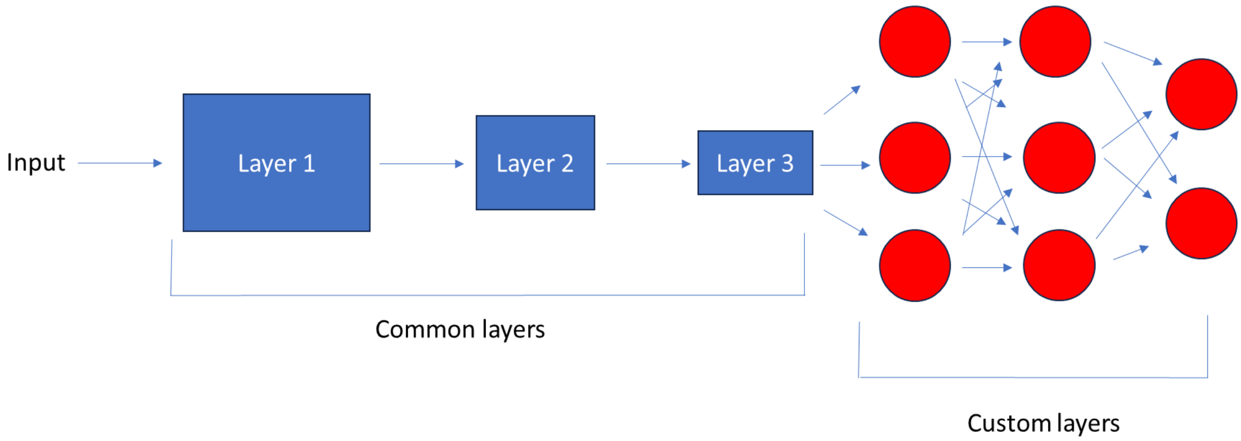


Figure 12: Transfer learning architecture

In our transfer learning model (custom model), the first layers are frozen, and we utilize the weights/biases from the pretrained model. Since we expect the high-level features to differ across domains, the FC layers (red) are not frozen and are retrained with random weights initialization. The output layer may need to be replaced if the number of classes is different in the target domain. This architecture is applicable if the domains are related and the number of training examples from the target domain is small.

Fine tuning

In the scenario where the number of training examples in the target domain is large and the domains are similar; we can utilize the fine-tuning approach. This architecture begins similarly as above: red FC layers are not frozen and are initialized with random weights whereas the output layer is modified for the number of the classes in the target domain. Next, the common inner layers are unfrozen and initialized with weights from the pre-trained model. The entire custom model is trained on examples from the target domain. We called this methodology, fine tuning, since the common layers which are initialized with pre-trained weights are just fine tuned on the target domain. We expect the model performance to improve in the target domain after fine tuning and the training time should be reduced compared to the scenario of retraining the entire model with random initialization.

LORA

LORA, or low rank adaptation of large language models, allows for efficient fine tuning of large language models. Typically, these models can have 100s of billions of parameters making them computationally expensive to fine tune. However, it has been observed that the weight matrices of an LLM can be represented by a low rank representation. With LORA, we can represent the update of the weight matrices of the attention layers of an LLM as: delta W = B*A where B*A is a low rank decomposition of delta W. During training, only A and B are learned whereas the original weights are frozen. Since the rank decomposition matrices have much fewer parameters than W, training of LLMs is possible with much less VRAM and computational resources. Another advantage of LORA is efficient task switching since you only need to provide new LORA weights per task. However, LORA has some disadvantages as well: 1) the low rank adaptation procedure may lose information, and 2) the performance of LORA can be impacted by the selection of the LORA rank.

References

1. https://www.ml6.eu/blogpost/low-rank-adaptation-a-technical-deep-dive
2. https://arxiv.org/abs/2106.09685
3. https://huggingface.co/docs/diffusers/main/en/training/lora
4. https://www.entrypointai.com/blog/lora-fine-tuning/

Adapter transfer learning

Standard fine tuning of a pretrained neural network model is inefficient when your goal is to fine tune multiple new downstream tasks requiring new models for each task and tuning of 100% of the model weights for each task. Houlsby et al propose the idea of adapters, neural network modules with a small number of weights, which are fine tuned for each new task, while pre-existing neural network weights are frozen. This results in a dramatic reduction in the number of trainable parameters per task and a high degree of parameter sharing across task models. Even though adapters have two orders of magnitude fewer parameters than full fine tuning, Houlsby et al illustrated similar performance between adapters and full fine tuning.

Specifically for the transformer architecture, adapter modules are added to the transformer layers (see below left). These adapter modules consist of a bottleneck layer, an up-projecting layer, and a skip-connection (see below right). The adapter has a single hyperparameter, namely the number of units in the bottleneck layer. With this architecture, adapters modules, layer norm layers, and the final classification layer are tuned per task while all other layers are frozen. Adapter learning has some advantages compared to previous learning approaches: 1) compared to the continual learning method, adapter tuning will not forget previous tasks during retraining (aka catastrophic forgetting) and tasks will not interact; and 2) compared to multi-task learning, adapter learning does not require simultaneous access to datasets for all tasks.

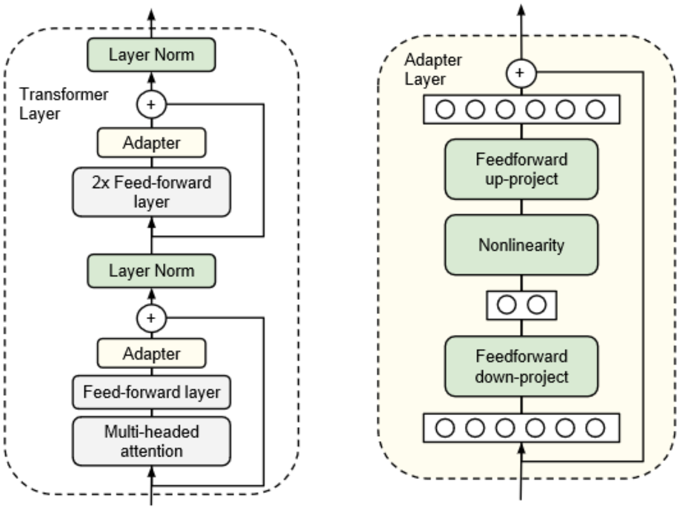


Figure 13: Adapter Learning Architecture from https://arxiv.org/pdf/1902.00751.pdf

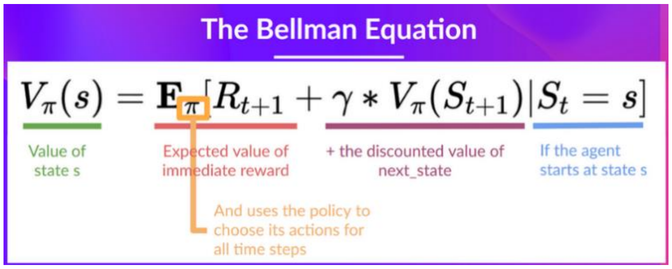
References

1. Houlsby, Giurgiu, et al. “Parameter Efficient Transfer Learning for NLP”. https://arxiv.org/pdf/1902.00751.pdf

Reinforcement learning

Reinforcement learning is a machine learning framework where an AI agent explores an environment with a set of possible actions based on its current state and tries to maximize a sum of rewards. This framework has been applied to a diverse set of applications including robot navigation, factory process control, games, and operations research. To gain practice with reinforcement learning, an ML practitioner can use the test environments of the openAI gym (https://github.com/Farama-Foundation/Gymnasium).

More specifically, we denote the finite space of possible actions in the environment as A and the finite set of possible states in the environment as S. However, only part of the environmental state is actually observable by the agent at a given time. For example, in reinforcement learning applied to poker, the environment state is all of the players’ hands and community cards, whereas the observable state is just the agent’s hand and community cards (https://ai.stackexchange.com/questions/38977/when-is-it-necessary-to-explicitly-define-both-the-state-and-observation-space-i). Based on an agent’s current observations, the agent will take an action based on its policy $\pi(a_t|s_t)$ resulting in a reward and new observations. During reinforcement learning, the policy is updated to try to maximize the sum of rewards, $\sum_{t=0}^{T} \gamma^t r_t$, where gamma is the discounting rate. Alternatively, the sum of rewards can be represented recursively using the Bellman equation:



The Bellman Equation

$$V_\pi(s) = \mathbf{E}_\pi[R_{t+1} + \gamma * V_\pi(S_{t+1}) | S_t = s]$$

Value of state s

Expected value of immediate reward

+ the discounted value of next_state

If the agent starts at state s

And uses the policy to choose its actions for all time steps

Figure 14: Bellman Equation from: https://huggingface.co/learn/deep-rl-course/unit2/bellman-equation

This equation says that the value of states is equal to an immediate reward plus a discounted value of the next state while the agent follows policy pi to choose its actions.

Some specific methods for achieving reinforcement learning are Q learning, deep Q learning, and E3. The Q function or the state-action value function attempts to model the expected return or the discounted sum of rewards. The Q function can be represented as either a table of values (the Q table) for each combination of (s,a) or can be approximated by a neural network, aka deep Q-learning.

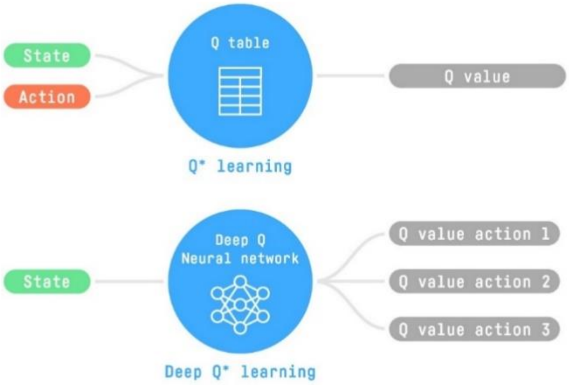


Figure 15: Q learning and Deep Q learning from: https://huggingface.co/learn/deep-rl-course/unit3/from-q-to-dqn

Additionally, we want to balance exploration of the environment versus exploitation (performing an action that maximizes Q). This balance can be achieved by taking the optimal action with probability, 1 – epsilon, and taking a random action (exploration) with probably epsilon. Over time, epsilon is reduced so the algorithm focuses on the optimal action. To achieve learning with Q tables, the Q function is updated after an agent takes an action using the Q update rule:

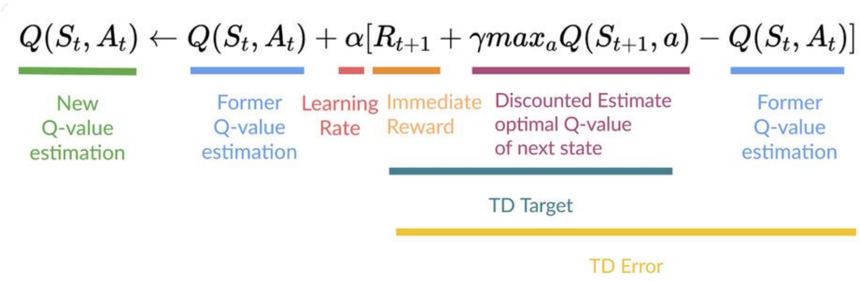

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
New Q-value estimation
Former Q-value estimation
Learning Rate
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation
TD Target
TD Error

Figure 16: Q update equation from: https://huggingface.co/learn/deep-rl-course/unit2/q-learning

whereas in deep Q learning, gradient descent minimizes the q-loss:

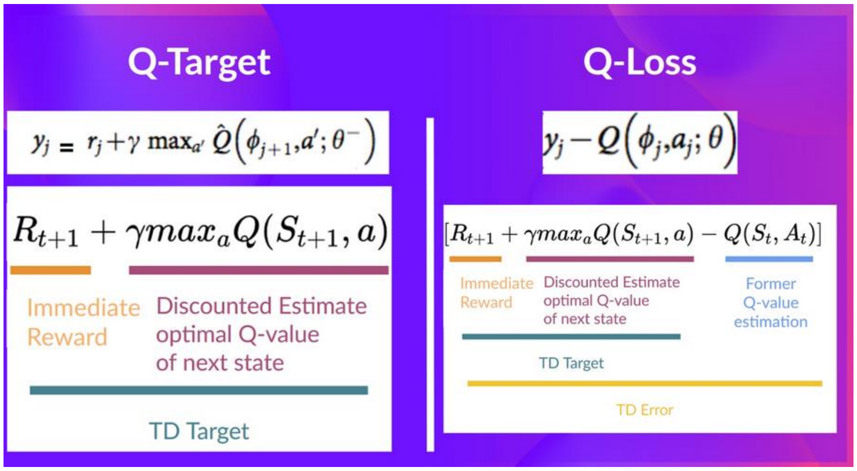

Q-Target
$y_j = r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right)$
$R_{t+1} + \gamma max_a Q(S_{t+1}, a)$
Immediate Reward
Discounted Estimate optimal Q-value of next state
TD Target
Q-Loss
$y_j - Q\left(\phi_j, a_j; \theta\right)$
$[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation
TD Target
TD Error

Figure 17: Q-Target and Q-loss from: https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm

References

1. https://web.eecs.umich.edu/~baveja/NIPS05RLTutorial/NIPS05RLMainTutorial.pdf
2. https://www.tensorflow.org/agents/tutorials/0_intro_rl
3. https://huggingface.co/learn/deep-rl-course/

Other Learning Methods

Auxiliary task learning

Some other recent learning methods in the literature include auxiliary task learning, prefix fine tuning, prompt fine tuning, and the IPT method (intrinsic prompt tuning). Briefly, auxiliary task learning consists of learning to accomplish auxiliary tasks with the main objective of having good performance on one or more primary tasks. For example, in computer vision, a primary task could be “Identifying facial landmarks on face pictures” with an auxiliary task of “Estimating head pose and predicting facial attributes” (from ref below). Formally, in auxiliary task learning, we use the loss gradient as: $G = \nabla\mathcal{L}_{primary} + \lambda G_{auxiliary}$ and $\nabla\mathcal{L}_{primary}$ is the gradient of the loss for the primary task with the auxiliary loss gradient as:

Method	Adjusted auxiliary loss gradient $G_{\text{auxiliary}}$
Unweighted cosine	$\begin{cases} \nabla\mathcal{L}_{\text{auxiliary}} & \text{if } \cos(\nabla\mathcal{L}_{\text{primary}}, \nabla\mathcal{L}_{\text{auxiliary}}) \geq 0 \\ 0 & \text{if } \cos(\nabla\mathcal{L}_{\text{primary}}, \nabla\mathcal{L}_{\text{auxiliary}}) < 0 \end{cases}$
Weighted cosine	$\max(0, \cos(\nabla\mathcal{L}_{\text{primary}}, \nabla\mathcal{L}_{\text{auxiliary}})).\, \nabla\mathcal{L}_{\text{auxiliary}}$
Projection	$\nabla\mathcal{L}_{\text{auxiliary}} - \min(0, \nabla\mathcal{L}_{\text{auxiliary}} \cdot \frac{\nabla\mathcal{L}_{\text{primary}}}{\lVert\nabla\mathcal{L}_{\text{primary}}\rVert}) \cdot \frac{\nabla\mathcal{L}_{\text{primary}}}{\lVert\nabla\mathcal{L}_{\text{primary}}\rVert}$

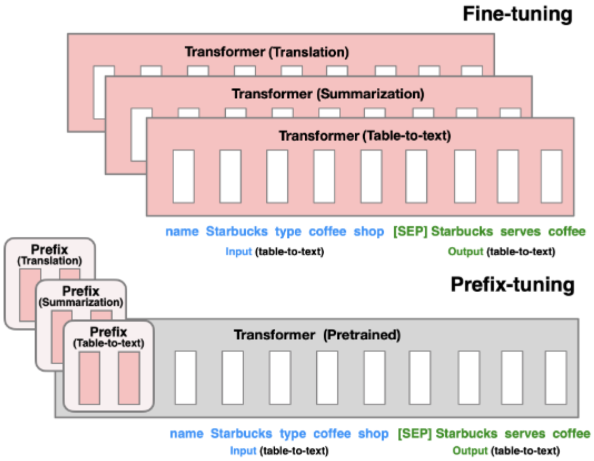
Table 17: Loss Gradient equation from below reference

Reference

1. https://vivien000.github.io/blog/journal/learning-though-auxiliary_tasks.html#auxiliary-tasks-in-machine-learning

Prefix finetuning

Prefix finetuning is an alternative strategy to fine tuning that more efficiently tunes multiple tasks. In standard fine tuning, a copy of all model parameters is needed for each task:



Fine-tuning
Transformer (Translation)
Transformer (Summarization)
Transformer (Table-to-text)
name Starbucks type coffee shop [SEP] Starbucks serves coffee
Input (table-to-text)
Output (table-to-text)
Prefix-tuning
Prefix (Translation)
Prefix (Summarization)
Prefix (Table-to-text)
Transformer (Pretrained)
name Starbucks type coffee shop [SEP] Starbucks serves coffee
Input (table-to-text)
Output (table-to-text)

Figure 18: Schematic of prefix tuning vs fine tuning (https://arxiv.org/pdf/2101.00190.pdf)

In contrast, prefix tuning keeps the neural network model parameters frozen and only requires tuning of a small prefix ("sequence of continuous task-specific vectors", see ref below) for each downstream task. Literature experiments illustrate prefix tuning is effective for table to text generation while only requiring 0.1% of the total parameters for tuning. Additionally, prefix tuning performs well in low-data settings and extrapolation settings. However, in the summarization setting, prefix tuning achieved lower performance when compared to fine tuning.

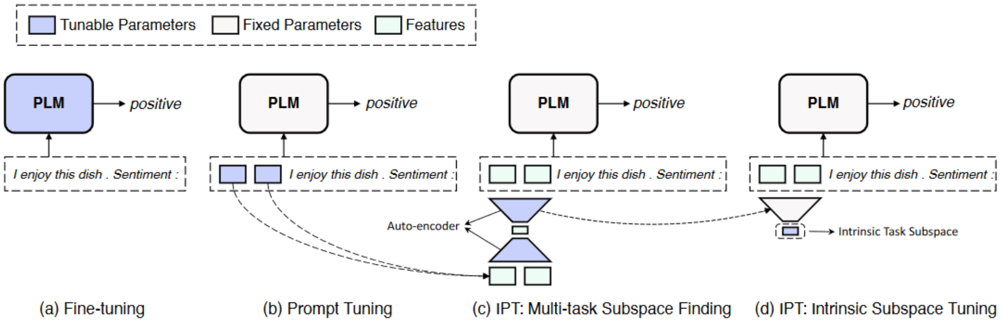
Reference

1. https://arxiv.org/pdf/2101.00190.pdf

Prompt tuning and intrinsic prompt tuning

Prompt tuning is an alternative strategy to standard prompt engineering (hard prompts) where a person handcrafts a prompt to adapt a large language model to a specialized task. In contrast, in prompt tuning also called soft prompting, the AI tunes an embedding vector for each downstream task. Unfortunately, soft prompts are not interpretable in contrast to the interpretability of hard prompts.

Intrinsic prompt tuning (or IPT) is a variant of soft prompting for multi-task learning. IPT has two key stages: 1) identifying a low-dimensional intrinsic task subspace for multiple tasks, and 2) adapting the neural network to new tasks by tuning a small number of parameters in the identified subspace.


Tunable Parameters
Fixed Parameters
Features
PLM
positive
I enjoy this dish . Sentiment :
PLM
positive
I enjoy this dish . Sentiment :
PLM
positive
I enjoy this dish . Sentiment :
Auto-encoder
PLM
positive
I enjoy this dish . Sentiment :
Intrinsic Task Subspace

(a) Fine-tuning (b) Prompt Tuning (c) IPT: Multi-task Subspace Finding (d) IPT: Intrinsic Subspace Tuning

Figure 19: Comparison of fine tuning, prompt tuning, and IPT from: https://arxiv.org/pdf/2110.07867.pdf

Literature experiments show that the intrinsic task subspace identified across 100 tasks which is then tuned within this subspace achieves 90% of the performance of full prompt tuning (relative performance) when evaluated on unseen data. In addition, IPT achieves 83% relative performance when evaluated on unseen tasks while only using 250 tunable parameters.

References

1. https://research.ibm.com/blog/what-is-ai-prompt-tuning
2. https://arxiv.org/pdf/2110.07867.pdf

Semi-supervised Learning

Semi-supervised learning (SSL, see more details: https://arxiv.org/abs/2103.00550) is an approach that uses a combination of labeled and unlabeled data to improve the performance of pretrained models. The rationale behind SSL is that even though labeled data can be scarce and expensive to obtain, there is usually an abundance of unlabeled data available. By leveraging this unlabeled data, SSL can improve the generalization capabilities of the pretrained models.

One way to use semi-supervised learning with pretrained models is to fine-tune the model using a combination of labeled and unlabeled data. This can be achieved by employing several semi-supervised learning techniques, such as:

i. Self-training: In this approach, the pretrained model is initially fine-tuned using labeled data. The model is then used to predict labels for unlabeled data. The most confident predictions are added to the labeled dataset and used for further fine-tuning of the model.
ii. Pseudo-labeling: Generate "pseudo-labels" for unlabeled data using the pretrained model's predictions and add the pseudo-labeled data to the training set. This expands the labeled dataset to improve the model.
iii. Consistency regularization: This technique aims to encourage the model to produce consistent predictions for different perturbations of the same input data. By enforcing this consistency, the model can learn to generalize better from the available labeled and unlabeled data.
iv. Mean Teacher: The Mean Teacher approach maintains two versions of the same model: a student and a teacher. The student model is trained on labeled data, while the teacher model is an exponential moving average of the student model. The consistency between the predictions of the student and teacher models is enforced, which helps in utilizing the information from the unlabeled data. (see details: https://github.com/CuriousAI/mean-teacher)
v. Adversarial Attacks: Using adversarial attacks on pretrained models with semi-supervision has shown substantial advances in classifying images. (see details: https://arxiv.org/abs/2308.04018)

Caution is essential to prevent the reinforcement of errors when utilizing unlabeled data. In summary, semi-supervised learning serves as an effective method for harnessing the potential of unlabeled datasets and enhancing the performance of deep learning models.

RLHF (Reinforcement learning from human feedback)

Reinforcement learning from human feedback (RLHF, see details: https://huggingface.co/blog/rlhf) is a technique that involves training a model using feedback from humans to improve its performance. In the context of pretrained models, RLHF can be used to fine-tune and adapt the model to specific tasks or domains by incorporating human expertise and preferences.

The RLHF process involves three core steps: “pretraining a language model (LM), gathering data and training a reward model, and fine-tuning the LM with reinforcement learning” (see above ref). The pretraining phase doesn't result in a perfect model; the model is expected to make mistakes and generate incorrect outputs. However, it provides a substantial starting point upon which RLHF can build, making the model more accurate, safe, and useful.

The process of using RLHF to improve the performance of pretrained models involves the following steps:

i. Collect human feedback: Obtain expert feedback on model predictions or behavior. This can be done by presenting model-generated outputs to experts and asking them to rate or rank the outputs based on their quality, relevance, or correctness.
ii. Create a reward model: Train a reward model using the collected human feedback. The reward model captures the relationship between the model's outputs and the feedback provided by the human expert.

iii. Fine-tune the pretrained model: Use the reward model as a guide to fine-tune the pretrained model using reinforcement learning techniques. This process involves updating the model's parameters to maximize the expected reward, which is estimated using the reward model.

iv. Iterate the process: The process of collecting human feedback, training the reward model, and fine-tuning the pretrained model can be iterated multiple times to continually improve the model's performance.

By incorporating human feedback and using reinforcement learning techniques, pretrained models can be fine-tuned to perform better in specific tasks or domains. This approach allows for the seamless integration of human expertise into the model, leading to improved performance and more reliable results. The common package in Python to implement RLHF is “TRL - Transformer Reinforcement Learning” (see details: https://huggingface.co/docs/trl/en/index).

Gradual unfreezing/Chain-thaw

The gradual unfreezing and chain-thaw approaches are alternatives to fine turning and transfer learning to improve model performance. Gradual unfreezing is discussed in Howard et al (https://arxiv.org/pdf/1801.06146) and Yosinski et al (https://arxiv.org/pdf/1411.1792). The idea here is to unfreeze the last layer and train for 1 epoch. Then unfreeze the next to last layer and train for 1 epoch the unfrozen layers. This process continues until all layers are fine tuned. This approach has advantages to full fine tuning in which it minimizes catastrophic forgetting.

A related method is called chain-thaw as introduced in Felbo et al (https://arxiv.org/pdf/1708.00524). In contrast to gradual unfreezing, the chain-thaw approach trains a single layer at a time and trains each layer sequentially until convergence with all other layers frozen. In detail, chain-thaw trains any new layers with the rest of the layers frozen. Then layer 1 is trained with the remaining layers frozen; Layer 2 is trained with remaining layers frozen, etc; and finally, the whole network is fine tuned. This procedure allows us to transfer knowledge (ie transfer learning) to a new domain while minimizing overfitting.

QLoRA (Quantized Low-Rank Adaptation)

QLoRA (Quantized Low-Rank Adaptation) is an efficient fine-tuning method that reduces memory usage to finetune large language models (LLMs) on a single 48GB GPU while keeping full 16-bit finetuning task performance. QLoRA combines quantization and Low-Rank Adapters (LoRA) in such a way that a pretrained model is first quantized to 4-bit and then a small set of learnable LoRA weights are added. QLoRA aims to make fine-tuning more accessible and resource-efficient by reducing the size and complexity of LLM parameters. Hence, it allows the user to fine-tune massive models with billions of parameters on relatively small GPUs. For instance, QLoRA reduces the average memory usage for finetuning a 65B parameter model from > 780 GB of GPU memory to < 48 GB without performance loss, compared to a 16-bit fully finetuned reference.

To make the model more compact and faster to execute, QLoRA reduces the numerical precision of a model’s tensors via quantization. QLoRA made multiple efforts to save memory without sacrificing performance:

- 4-bit NormalFloat (NF4) quantization, which transforms all weights to a fixed normal distribution that fits within the range of NF4 (-1 to 1).
- Double quantization, which reduces the average memory footprint by quantizing the quantization constants.
- Paged optimizers to manage memory spikes during gradient checkpointing.

See https://github.com/artidoro/qlora?ref=blog.lancedb.com for tutorials and demonstrations on how to run QLoRA.

Overall, LoRA and QLoRA were developed to fine-tune LLMs more efficiently, but the following differences may be considered to choose between the two techniques.

Comparison of LoRA and QLoRA (Table 18):

Feature	LoRA	QLoRA
Parameter reduction	Low-rank approximation of ΔW	Quantization of LoRA adapter weights
Memory footprint	Reduced	Further reduced
Fine-tuning speed	Fast	Slightly slower than LoRA
Performance	Close to traditional fine-tuning	Similar to LoRA

REFERENCES

1. https://medium.com/@sujathamudadla1213/difference-between-qlora-and-lora-for-fine-tuning-llms-0ea35a195535#:~:text=Choosing%20between%20LoRA%20and%20QLoRA,a%20good%20balance%20between%20both.
2. https://wandb.ai/sauravmaheshkar/QLoRA/reports/What-is-QLoRA---Vmlldzo2MTI2OTc5
3. https://www.brev.dev/blog/how-qlora-works
4. https://arxiv.org/pdf/2305.14314
5. https://github.com/artidoro/qlora?ref=blog.lancedb.com

LoRA combined with Prefix-tuning

For the orthogonality of LoRA to the other finetuning methods, LoRA can be combined with prefix tuning within the Parameter Efficient Fine-tuning (PEFT) framework to implement specific fine-tuning with minimal data in such that LoRA reduces overall parameters, and prefix-tuning provides task-specific control. LoRA combined with prefix-tuning is well suited for adapting LLMs to specific domains using limited in-domain data. Likewise, this combination unlocks the full potential of LLMs for users with more efficient computation and constrained data resources.

References

1. https://sebastianraschka.com/blog/2023/llm-finetuning-lora.html
2. https://toloka.ai/blog/prefix-tuning-vs-fine-tuning/
3. https://arxiv.org/pdf/2303.15647
4. https://arxiv.org/pdf/2101.00190

LoRA in Stable Diffusion

LoRA in the stable diffusion model is a special type of model used in image processing and AI. The model is based on the diffusion process approach where information about different pixels in an image is continuously exchanged. This concurrent diffusion generates new images and stylized based on an existing image. In addition to capturing/specifying the different styles and characteristics of images, via the fine-tuning of LoRA, the model is gradually optimized to achieve accurate and realistic results. The model optimization runs based on the smaller number of parameters to estimate than the original model. For example, for a model with a matrix with 1k rows and 2k columns, LoRA breaks down the matrix into a 1k-by-2 matrix and a 2-by-2k matrix. Compared to the original 2M numbers stored in the model file, only 6k numbers are needed in the LoRA files, which is 333 times fewer numbers. The LoRA Stable Diffusion model is stable (generates consistent and high-quality images even when using new prompts or styles).

References

1. https://neuroflash.com/blog/lora-stable-diffusion/#:~:text=LoRA%20stable%20diffusion%20models%20are,and%20unique%20patterns%20and%20textures.
2. https://www.datacamp.com/tutorial/mastering-low-rank-adaptation-lora-enhancing-large-language-models-for-efficient-adaptation
3. https://www.mercity.ai/blog-post/fine-tuning-llms-using-peft-and-lora

Prompt Engineering

Prompt engineering refers to the technique of carefully constructing the prompts or inputs fed into deep learning models to improve their performance. As deep learning models are trained on large datasets to recognize patterns, the way the input data is framed impacts how the model processes it.

With advanced large language models like GPT, LLaMa, etc., the prompt serves as the main mechanism for guiding the model - essentially the prompt defines the task for the model. As such, better prompts can enhance what the model pays attention to and improve the relevance of its response. For instance, consider the task of sentiment analysis. Instead of simply providing sentiment labeled sentences, the prompt could frame it as: “I will provide student survey responses. Please analyze if the response shows a positive or negative sentiment regarding the grading policy. Positive means students liked the setting of grading policy. Negative means students were unhappy about it. Please return ‘Irrelevant’ if the response is not related to the grading policy. Let’s analyze the following comment: I love this class, but it would be better if grading policy can be more transparent.”

Some best practices for prompt engineering include (see details: https://guides.library.georgetown.edu/ai/prompts):

- Using clear task-defining instructions - Prompts should clearly state the task the model is expected to perform rather than just providing input data. For example, "Please translate the following text from French to English" is more effective than just providing the text.
- Providing relevant context - Adding related contextual details allows the model to better interpret the intent and make connections.
- Structuring with examples - Providing the model with a few examples of desired inputs and outputs helps define the expected response format.
- Simplifying input data - Avoiding complex terminology or phrasing generally produces better responses from the model.
- Iteratively testing and tweaking - Start with a simple prompt and test model outputs, then tweak parts like the instructions, examples, or data simplification to steadily improve results.

Tracking model performance across prompt variations allows us to determine optimal prompts. Continual prompt tuning and A/B testing prompts on new model versions enables us to engineer increasingly effective prompts over time. The careful art of prompt engineering allows us to take advantage of the capabilities already within pretrained models instead of extensively retraining them, serving as a powerful tool for practitioners applying deep learning techniques.

Chapter 11: Model Interpretability

Perturbation-based methods

LIME

LIME (Local Interpretable Model-agnostic Explanations, Ribeiro et al. 2016) can interpret ANY trained black-box model using a local surrogate mechanism. The local surrogate model with interpretability constraint is expressed by:

$$explanation(x) = \underset{g\in G}{\operatorname{argmin}}\, L(f, g, \pi_x) + \Omega(g(K))$$

- g is the explanation model, for instance x (e.g. linear regression model)
- L is a loss function to be minimized (e.g. mean squared error, MSE)
- f is the original model (e.g. an xgboost model)
- $\Omega(g(K))$ is the model complexity (e.g. fewer features = smaller K) such as LASSO and forward or backward selection
- π_x is the proximity measure to define the boundary of the neighborhood around instance x for the explanation
- K is the number of features

Figure 20 illustrates the original model (f)'s binary decision (dark blue vs. yellow) non-linear function. Given the instance being explained (bold red cross, **X**), a linear model of dashed line (g) is learned from f's prediction based on the perturbed instances in the vicinity of **X**. Be aware that g is trustful locally around **X**, not globally.

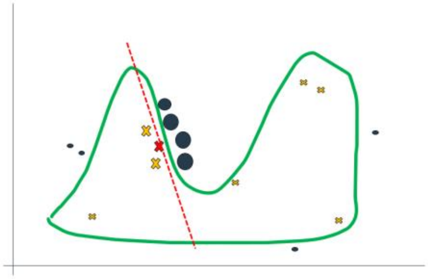
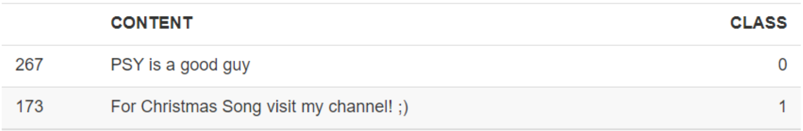


Figure 20: LIME schematic

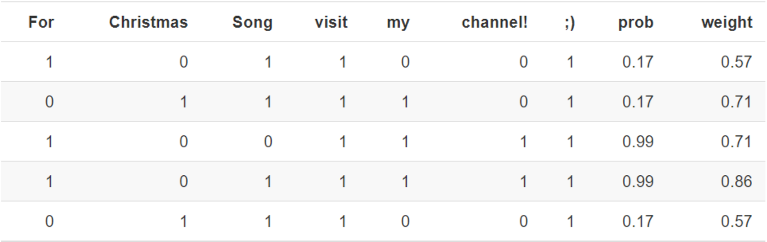
LIME is a well-known interpretation method for classification models by identifying important features in the binary or multi-level class(es)-prediction. For example, when dichotomizing the following two sentences in such binary classes (1 for spam and 0 for normal comment), word(s) affecting the binary decision model may be of interest.



	CONTENT	CLASS
267	PSY is a good guy	0
173	For Christmas Song visit my channel! ;)	1

Table 19: Classification of each sentence (example from: https://christophm.github.io/interpretable-ml-book/lime.html)

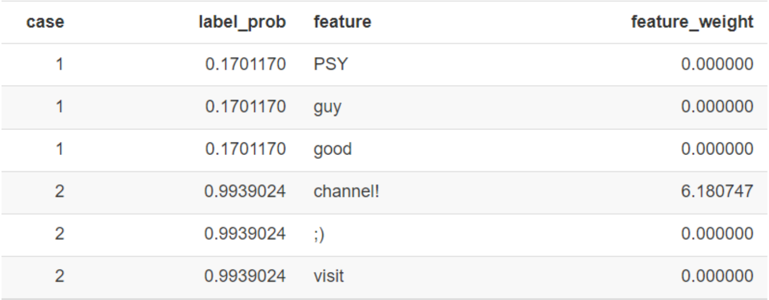
To run the model-agnostic algorithm, multiple new texts are created by arbitrarily excluding words from the original sentence. For each new text, the prediction probability of class == 1 (prob) and the proximity of the new text to the original sentence (weight) are computed.



For	Christmas	Song	visit	my	channel!	;)	prob	weight
1	0	1	1	0	0	1	0.17	0.57
0	1	1	1	1	0	1	0.17	0.71
1	0	0	1	1	1	1	0.99	0.71
1	0	1	1	1	1	1	0.99	0.86
0	1	1	1	0	0	1	0.17	0.57

Table 20: Probability and weight table (Example from: https://christophm.github.io/interpretable-ml-book/lime.html)

Based on the probability and weight estimations, LIME evaluated local weights for each feature (word) as follows:



case	label_prob	feature	feature_weight
1	0.1701170	PSY	0.000000
1	0.1701170	guy	0.000000
1	0.1701170	good	0.000000
2	0.9939024	channel!	6.180747
2	0.9939024	;)	0.000000
2	0.9939024	visit	0.000000

Table 21: Feature weights (Example from: https://christophm.github.io/interpretable-ml-book/lime.html)

The word “channel!” resulted in a significant contribution with non-zero feature_weight == 6.18 in a high probability of spam (label_prob = 0.99).

In addition to the text mining on classification prediction, LIME has also an explanation capability of general prediction model trained by tabular data, consisting of numerical inputs and values. For example, the following figure shows LIME explanations with two features (numerical temp and categorical weather situation or season) from the trained model for two instances (a classification random forest with 100 trees): warmer temperature and good weather situation make a positive influence on the average number of bicycles rented above a trend line, which is a linear prediction from a multi-variable regression.

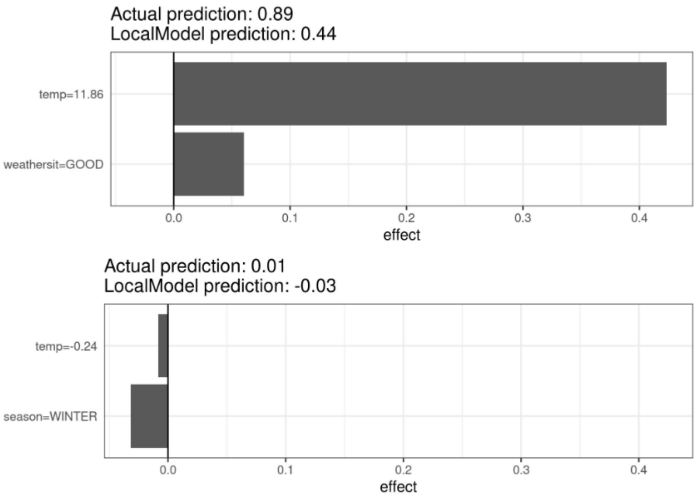


Figure 21: Effect of temperature and season (image from: https://christophm.github.io/interpretable-ml-book/lime.html)

- Image Data

  Let's see the example below: left showing the original image of a home-made bread, the middle presenting better prediction and explanation for "Bagel" with prediction probability of 77%, and the right displaying worse prediction and explanation for "Strawberry" with a prediction probability of 4% given two LIME explanations for "Bagel" and "Strawberry".

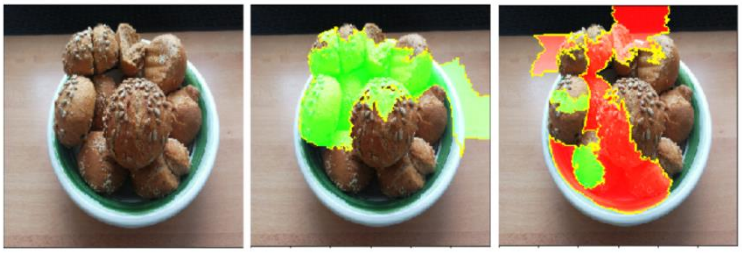


Figure 22: LIME explanations for bagel and strawberry (permission from author to use image from: https://christophm.github.io/interpretable-ml-book/lime.html)

References

1. Ribeiro et al, ""Why Should I Trust You?" Explaining the Predictions of Any Classifier, https://arxiv.org/pdf/1602.04938
2. https://christophm.github.io/interpretable-ml-book/lime.html

SHAP

SHAP (SHapley Additive exPlanations, Lundberg et al. 2017) is the explanation model in SHAP is basically expressed as

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z_j{}'$$

- g is the explanation model.
- $z' \in \{0,1\}^M$ is the coalition vector containing simplified input features (e.g. for image data, if there are four aggregated super-pixels (i.e. M = 4) and a coalition {1,3} is simulated, the coalition vector should be {1,0,1,0} where 0 and 1 mean that the corresponding feature values are "present" and "absent", respectively).
- M is the maximum coalition size.
- $\phi_j \in \mathbb{R}$ is the feature attribution for a feature j, the Shapley values.

Examples:

- SHAP explanation force plots (explanations for individual predictions) using SHAP values to interpret the predicted cervical cancer probabilities of two women.

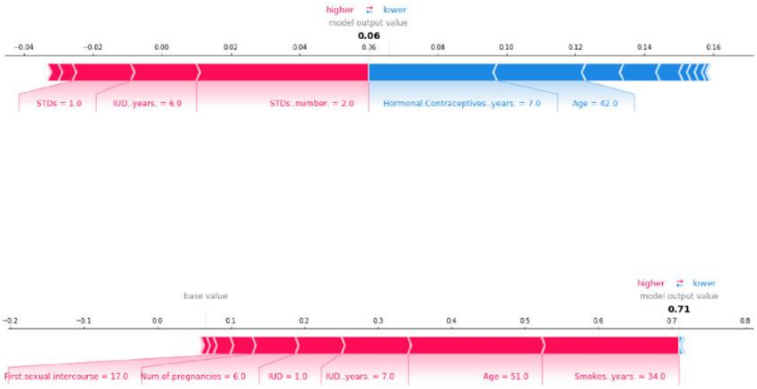


Figure 23: Example of SHAP force plots (image from: https://christophm.github.io/interpretable-ml-book/shap.html)

A risk for cervical cancer was predicted by training a random forest classifier (whether the biopsy result is "Healthy" or "Cancer") with 100 trees. Given the baseline (the predicted probability on average) of 0.066, the first woman (on top) returns a low predicted risk of 0.06 while the second woman (on bottom) shows high predicted risk of 0.71. For an example of the first woman, the low predicted risk was evaluated from the offset between increasing effects such as STDs (number of STD diagnoses) and IUD (number of years with an intrauterine device) and decreasing effects such as years of hormonal contraceptives and age.

- SHAP Feature Importance

  Features with large absolute Shapley values are significant. Global explanation is driven by averaging the absolute Shapley values per feature across the matrix with each row per instance (i) and each column per feature (j).

$$I_j = \frac{1}{n}\sum_{i=1}^{n}\left|\phi_j^{(i)}\right|$$

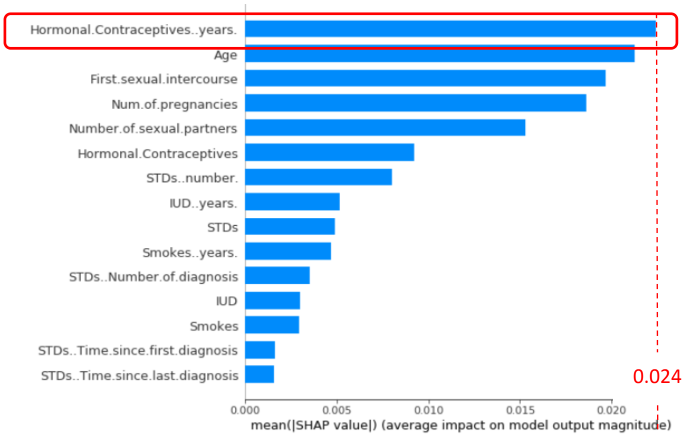


Figure 24: SHAP feature importances (image from: https://christophm.github.io/interpretable-ml-book/shap.html)

For instance, years with hormonal contraceptives is the most significant feature, adjusting the predicted probability of risk by 2.4% points on average.

- SHAP Summary Plot

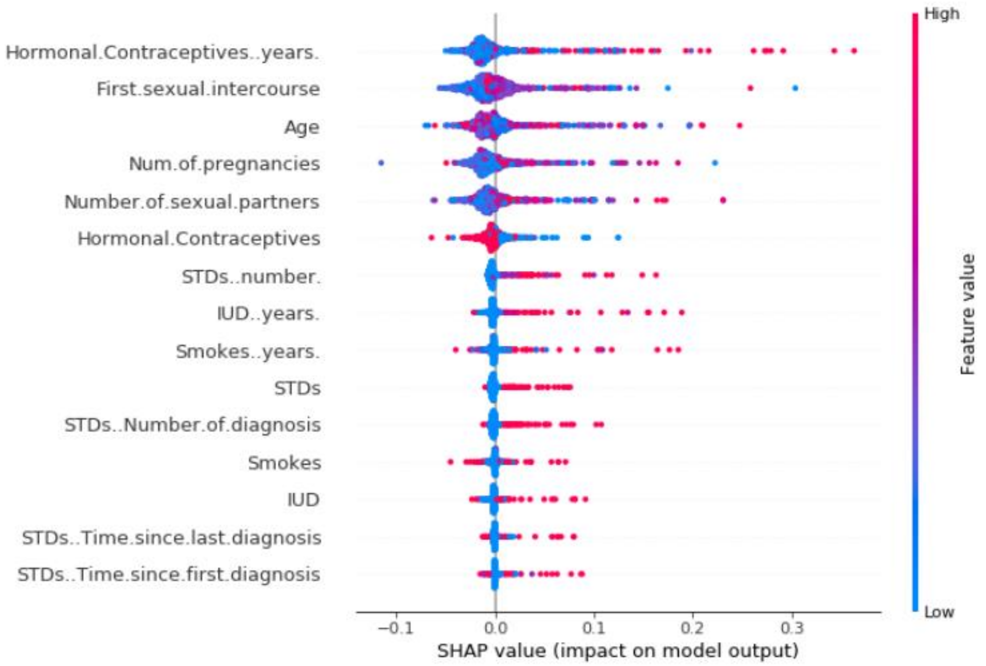


Figure 25: SHAP summary plot (image from: https://christophm.github.io/interpretable-ml-book/shap.html)

The above SHAP summary plot shows jittered points along with horizontal line where each point corresponds to Shapley value per instance per feature. For example, there is a relationship between the low predicted probability of risk and low number of years on hormonal contraceptives. Note that the color represents the feature value from low in blue to high in red.

References

1. Lundberg et al, "A Unified Approach to Interpreting Model Predictions." https://arxiv.org/pdf/1705.07874
2. https://christophm.github.io/interpretable-ml-book/shap.html

Counterfactual Explanations (CEs)

CEs aim to find factors that make a pivotal influence on a change in outcomes from "what if" questions. Given the binary outcome of either Y or Z, one may wonder *"why was the outcome Y, not Z, in a situation of X?"*. What changes to the situation of X would induce the alternative decision of Z? That is – *"if X was X', would the outcome be Z, not Y?"*. For example, one might be interested in the safety question regarding image classification, *"why did the self-driving car misidentify the fire hydrant as a stop sign?"*. To answer the above questions, multiple methods (Goyal et al. 2019, Arrieta and Ser 2020, Wang and Vasconcelos 2020, Zhao 2020, Kenny and Keane 2021, Vermeire and Martens 2022) based on underlying counterfactual explanations have been developed to interpret the decisions of deep computer vision systems by detecting regions of an input image of which change will generate a specific outcome.

Unlike LIME that is unclear on how we can extrapolate the local model for the interpretation, the interpretation of counterfactual explanation is very clear illustrating which features have been changed from the instance of interest (c) to the counterfactual instance (c'). When access to data and the model is prevented, the counterfactual method can provide users with explanations only based on the model's prediction function. The counterfactual explanation method for many types of data including text, image, and tabular is easy to implement. A Python package, Alibi, runs various counterfactual methods. Nevertheless, a practical challenge using this interpretability method can be multiple counterfactual explanations for a single instance (class) if users should report only one explanation.

References

1. Goyal Y, Wu Z, Ernst J, Batra D, Parikh D, Lee S (2019) Counterfactual visual explanations. In: Proceedings of the 36th international conference on machine learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, PMLR, Proceedings of machine learning research, vol 97, pp 2376–2384
2. Guidotti, R. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Min Knowl Disc* **38**, 2770–2824 (2024). https://doi.org/10.1007/s10618-022-00831-6
3. https://christophm.github.io/interpretable-ml-book/counterfactual.html
4. https://github.com/SeldonIO/alibi

Anchors (High-precision model-agnostic explanations)

Anchor is a novel LIME method that not only leverages a perturbation-based strategy to produce local explanations but also provides precision credibility as well. For text data analysis, using two exemplary instances of "This movie is not bad" and "This movie is not very good" for two sentiment predictions generated by an LSTM, although LIME explanations provide weights on the model (e.g. -0.24 of "bad" and 0.10 of "not" for the first sentence and -0.38 of "not" and 0.20 of "good" for the second sentence), Anchor explanations provide more human understandable interpretability (e.g. their coverage like "when does "not" have a positive influence on sentiment?"). In this example, $A(x = \text{"}This\ movie\ is\ not\ bad\text{"}) = 1$ where a rule of $A = \{\text{not, bad}\}$. That is, A is a sufficient rule for the model of $f(x) = positive$ with high probability. While LIME interpretability identifies the linear line to approximate the model with some local weighting under the same $\mathcal{D}$, Anchors derives more precise local approximation than LIME by evaluating the degree of precision with respect to the conditional probability.

Compared to LIME, the anchor's explanation is more human understandable as rules (A) are easy to interpret. The approach is less risky to underfit the model from reinforcement learning and works well even when a model is non-linear or complex with multi-collinearity. Computational efficiency can be achieved by running in parallel, leveraging MABs (Multi-Armed Bandits) that support batch sampling (e.g. BatchSAR). However, the approach suffers from some disadvantages in common with other perturbation-based methods (e.g. a highly configurable setup and running time inefficiency). In addition, to avoid too specific results and hence low coverage, be aware of your data prior to deciding on how to discretize data. In other words, to make the anchor more concise, understanding data is critical to determine which features to be iteratively added to a null rule until we maximize the coverage. The approach can be implemented via Anchor Python package (GitHub - marcotcr/anchor: Code for "High-Precision Model-Agnostic Explanations" paper) or alibi (GitHub - SeldonIO/alibi: Algorithms for explaining machine learning models).

References

1. https://christophm.github.io/interpretable-ml-book/anchors.html
2. Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. “Anchors: high-precision model-agnostic explanations”. AAAI Conference on Artificial Intelligence (AAAI), 2018
3. https://cs.carleton.edu/cs_comps/2324/explainable-ai/final-results/Anchors/Introduction

Table 22: Counterfactual Explanations vs Anchors

	PROS	CONS
Counterfactual Explanations	• Clear interpretation of underlying mechanism unlike LIME • Explanations ONLY based on the model’s prediction function, with no data access • Many types of data applicable	• “Rashomon effect” where multiple, potentially contradictory, counterfactual scenarios can exist for a single outcome, making difficult to choose the most accurate or relevant explanation
Anchors	• More human interpretable than LIME • Less risky to underfit the model • Works well even when a model is non-linear or complex with multi-collinearity	• Disadvantages in common with other perturbation-based methods (e.g. a highly configurable setup and inefficient computation time)

Gradient-based methods

Saliency Maps (Simonyan et al. 2014)

For class c, $S_c(I)$ represents the predicted output of a neural network (e.g. CNN) for image I. The linear score model for the class c can be expressed by

$$S_c(I) = w_c^T I + b_c$$

- I is the input image in the form of a numeric vector.
- w_c is a weight vector.
- b_c is a bias of the model.

Interpretation is determined by the gradient of class score (S_c) of interest with respect to the input image (I) at specific pixel (I_0):

$$w_c = \frac{\partial S_c}{\partial I} |_{I=I_0}$$

Then, the gradients will be visualized, displaying the absolute values or highlighting negative and positive impacts.

In addition to image interpretability, Saliency maps can be utilized for natural language explanations (NLEs) by discovering important features (tokens) and spans that are the most salient. A verbalized saliency map S_V (see: https://arxiv.org/pdf/2210.07222) is expressed by,

$$v\big(W, S = e(W, m)\big) = S_V$$

- v is any verbalizer (e.g. IG: Integrated Gradients), which discretizes attribution scores/saliency map (S) and produces a natural language representation, S_v.
- $W = (w_1, \dots, w_n)$ is a vector of source (input) tokens.
- e is a feature explanation method (e.g. a BERT model), which generates a saliency map.
- m is an underlying (explained) black-box model which predicts outcomes.

To make S_V concise for a human explanation, a coverage metric is defined by the proportion of coverage by the tokens involved in S_V out of the total attribution in S (see: https://arxiv.org/pdf/2210.07222),

$$Coverage(S_V) = \frac{\sum |v_i| \in S_V}{||S||}$$

The example of Feldhus et al. (2022) (https://arxiv.org/pdf/2210.07222v1) illustrates the candidate generation and selection process with an instance from IMDb. This is an example when the sentiment analyzer (BERT model) wrongly predicted the negative movie review from IMDb as positive sentiment by focusing on the "wrong" tokens. The colors and darkness in each word indicate how important it was for the analyzer to predict "positive sentiment" by running instruction-based verbalizations SMV (Saliency Map Verbalizer) using GPT-3.5. Depending on the negative or positive saliency (importance) score for each word, the word was colored in blue or red. The darker color represents more departure from zero, hence more importance (see Table 1 of Feldhus et al. 2022 for more details).

Additional references:
1. https://christophm.github.io/interpretable-ml-book/pixel-attribution.html

Grad-CAM (Gradient-weighted Class Activation Mapping, Selvaraju 2019)

Grad-CAM is utilized to analyze which area is activated in the feature maps of the last convolutional layers by estimating importance weights. Grad-CAM decides the importance weights for each of the k feature maps to a class c of interest.

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i \sum_j}^{\text{global average pooling}} \quad \underbrace{\frac{\delta y^c}{\delta A_{ij}^k}}_{\text{gradients via backprop}}$$

- y^c is the score of class c.
- A_{ij}^k is the activaction at pixel location (i, j) of the feature map k activation of a convolutional layer, A^k.
- α_k^c represents a partial linear transformation of the deep network downstream from A, and captures the "importance" of feature map k for a target class c.

- Z is the total number of dimensions encompassing $u \times v$ (i.e. $Z = \sum_i \sum_j 1$) when an input image is composed of $u \times v$ (width X height) pixels.

  For visual examples of the grad-cam algorithm, please see: https://arxiv.org/pdf/1610.02391 and https://christophm.github.io/interpretable-ml-book/pixel-attribution.html

Additional references:

1. https://christophm.github.io/interpretable-ml-book/pixel-attribution.html

Occlusion

Occlusion image interpretability leverages region-based segmentation, dividing an image into meaningful regions by considering both local and global information. It refers to a technique in computer vision where parts of an image are strategically blocked or "occluded" to understand which areas are most important for a machine learning model's prediction, necessarily revealing which parts of the image the model relies on the most to make its decision (classification); it is a way to interpret how a model "reads" an image by seeing how its prediction changes when key areas are hidden. A small area of an image is masked or replaced with a neutral color (like black or gray), and the model's prediction is observed; this process is repeated by sliding the occlusion across the entire image to identify which areas significantly impact the prediction.

For example, in MATLAB, the Deep Learning Toolbox offers the `occlusionSensitivity` function to compute occlusion sensitivity maps for deep neural networks. The function permutes small areas of the input image by replacing it with an occluding mask. The mask moves across the image and the change in probability score for a given class is measured as a function of mask position, i.e., when that part of the image is occluded, the probability score for the predicted class will be dropped dramatically.

By seeing how the model's prediction changes when parts of the image are occluded, users can identify whether it is making decisions based on relevant information. If a model is misclassifying images, occlusion analysis can help locate which areas are causing the errors. Understanding which areas are most significant can guide the development of new features or data augmentation strategies to improve the model's performance. However, evaluating the model's prediction for many occluded versions of an image can be computationally expensive. While occlusion provides insights into important areas, users should make sure their network focuses on the correct features of the input data. A solution to the misclassification issue is to re-train the network with more labeled data, covering a broad range of observations of the misclassified class.

References

1. Understand Network Predictions Using Occlusion
2. https://medium.com/@1marcusangella/mastering-the-art-of-image-segmentation-a-comprehensive-guide-to-image-segmentation-techniques-545a75010a35

3. Valois, Pedro H. V. and Niinuma, Koichiro and Fukui, Kazuhiro. Occlusion Sensitivity Analysis With Augmentation Subspace Perturbation in Deep Feature Space. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). January 2024: 4829-4838
4. https://medium.com/@dr.saad.laouadi/comparing-grad-cam-and-occlusion-sensitivity-for-medical-image-interpretation-a-brain-tumor-a2036878f88d
5. Huff DT, Weisman AJ, Jeraj R. Interpretation and visualization techniques for deep learning models in medical imaging. Phys Med Biol. 2021 Feb 2;66(4):04TR01. doi: 10.1088/1361-6560/abcd17. PMID: 33227719; PMCID: PMC8236074.
6. https://medium.com/information-creation-and-consciousness/explainable-ai-for-deep-metric-learning-an-occlusion-based-approach-495c5ebd4084

Implementation

	Python	R packages
LIME	https://github.com/marcotcr/lime	**lime** R-package https://cran.r-project.org/web/packages/lime/lime.pdf **iml** R-package https://cran.r-project.org/web/packages/iml/iml.pdf
SHAP	https://github.com/slundberg/shap	**Shapper** R-package https://modeloriented.github.io/shapper/ **fastshap** R-package https://github.com/bgreenwell/fastshap
Saliency Maps	https://pypi.org/project/tf-keras-vis/ https://github.com/albermax/innvestigate https://github.com/marcoancona/DeepExplain	N/A
Grad-CAM		
Occlusion	occlusionSensitivity - Explain network predictions by occluding the inputs - MATLAB	N/A

Table 23: Software packages for LIME, SHAP, Saliency Maps, Grad-CAM, and Occlusion

Comparisons

	PROS	CONS
LIME	Regardless of machine learning model used for training, the **same** local and interpretable machine learning model can be used for explanation. LIME makes human-friendly and interpretable explanations, not like principal components as an example.	Hard to define optimal neighborhood when using LIME with tabular data (trying different kernel settings may be helpful until finding reasonable explanations) Instability of the explanations stemming from the sampling process Possibility of manipulation of explanations to hide bias

	LIME works for tabular data, text, and images. **Easy to implement** in Python and R	
SHAP	Feature values are fairly evaluated from a solid theoretical foundation. **Fast implementation** for TreeSHAP, allowing to compute many Shapley values needed for the **global model interpretations**	KernelSHAP is super slow and ignores dependency among features. From TreeSHAP, unreasonable features can be selected as important from unlikely data points sampling. Possibility of manipulation of explanations to hide bias
Saliency Maps	Easy and fast to recognize the important regions (super-pixels) of the image. Compute faster than model-agnostic methods such as LIME and SHAP	Constant bias to all images can produce unreliable explanations for Saliency Maps. No evaluation method to assess their performance. No R package published yet
Grad-CAM		
Occlusion	Can be applied to any model without requiring architectural changes or knowing its internal workings Effectively pinpoints the regions most critical to a particular prediction Help in debugging models by highlighting regions Compare which parts of the input a network identifies as evidence for different classes	Computationally expensive by perturbing and re-evaluating the model numerous times Can mislead results by perturbing inputs that may lie outside the original data distribution the model was trained on This local interpretability may not fully reflect the model's overall behavior to other inputs. Effective application can be challenging, when categories overlap or have complex relationships. The way features are occluded can introduce artifacts or distort the input. Selecting larger occlusion patch sizes to reduce computational burden can lead to lower-resolution heatmaps.

Table 24: Comparison Table for LIME, SHAP, Saliency Maps, Grad-CAM, and Occlusion; citations for the table information can be found in the references for the respective sections above

Decision rule

If a user is much more comfortable with coding in R than Python, LIME and/or SHAP will be more feasible for the user. However, due to SHAP and LIME's computational inefficiency, Saliency Maps (or Grad-CAM) would be recommended to explain the model for NLP or computer vision in Python.

There are various evaluation methods for interpretable deep-learning in terms of three aspects: fidelity with respect to the original model, comprehensibility of the model, stability of interpretation (see details in https://www.mdpi.com/2078-2489/14/8/469). By examining computational metrics, the best explainable method for NLP, Computer Vision, or tabular data can be selected among candidates.

References

1. Ribeiro et al. “Why should I trust you?” explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016: 1135-1144.

2. Lundberg et al. A unified approach to interpreting model predictions. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

3. Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” arXiv preprint arXiv:1312.6034v2 (2014).

4. Selvaraju et al. Grad-CAM: Visual explanations from deep networks via gradient-based localization. 2019, arXiv.

5. https://christophm.github.io/interpretable-ml-book/interpretability.html

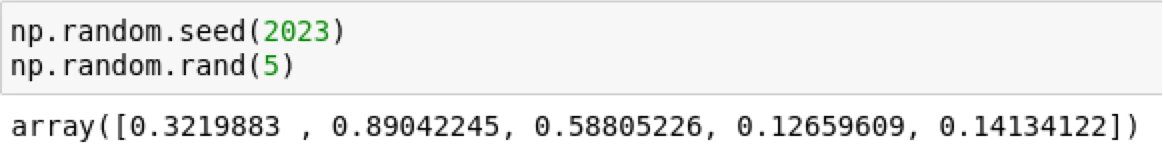
6. Feldhus, N., Hennig, L., Nasert, M.D., Ebert, C., Schwarzenberg, R., & Moller, S. (2022). Constructing Natural Language Explanations via Saliency Map Verbalization. ArXiv, abs/2210.07222.

Chapter 12: Model Reproducibility

Seed values

To create reproducible work (e.g. when validating trained models using cross-validation), controlling seed values is crucial for others to generate the same outputs from original programmer's codes. The most general approach is to use np.random as below. The global random seed set by np.random.seed(number) affects all uses to the np.random.* module.

NumPy's Global Random Seed

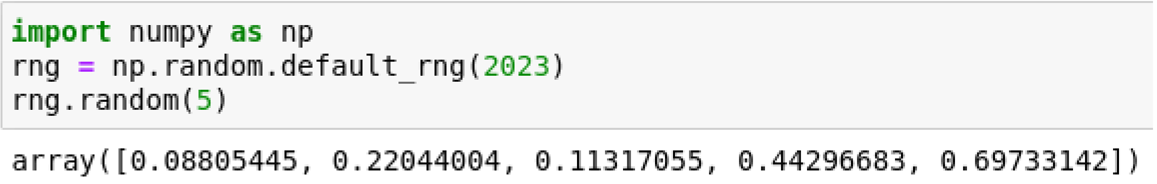


```
np.random.seed(2023)
np.random.rand(5)
```

```
array([0.3219883 , 0.89042245, 0.58805226, 0.12659609, 0.14134122])
```

But, some imported packages or other scripts might reset the global random seed to another in larger projects. The unexpecting changes will affect outputs and hence make results unreproducible. To avoid the issue, the best practice is to create a random number generator.

Best Practice



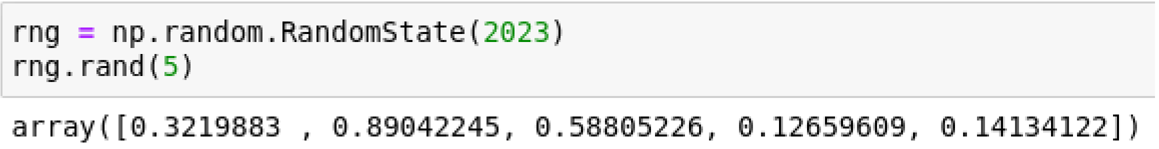
```
import numpy as np
rng = np.random.default_rng(2023)
rng.random(5)
```

```
array([0.08805445, 0.22044004, 0.11317055, 0.44296683, 0.69733142])
```

Another benefit of using the random number generator is transparency where we can see exactly what random number generator is used in each part of project by passing a random number generator to functions that need to be reproducible. For details, see: https://builtin.com/data-science/numpy-random-seed

The generated numbers from the best practice are different from the global random seed, as we can see. For such a case where a user wants to replicate the old global random seed results, try the following module,

A generator for old method



```
rng = np.random.RandomState(2023)
rng.rand(5)
```

```
array([0.3219883 , 0.89042245, 0.58805226, 0.12659609, 0.14134122])
```

In addition to "NumPy" package, visit the website (https://wandb.ai/sauravmaheshkar/RSNA-MICCAI/reports/How-to-Set-Random-Seeds-in-PyTorch-and-Tensorflow--VmlldzoxMDA2MDQy) for the code snippets which can be used in the codebase to set up random seeds for all the involved libraries in a PyTorch or Tensorflow pipeline.

Non-deterministic algorithm: GPT

It has been well-known that later versions of GPT-4 and GPT-3.5-turbo are non-deterministic, even at temperature=0.0. In general, (Sparse) MoE (Mixture of Experts: combining multiple experts and a trainable gating network that chooses a subset of experts for each sequence example) approaches are regarded as the reason of the inherent non-determinism. Nevertheless, in practice, by summarizing multiple GPT outputs via GPT again, we anticipate the summary output to become less variable across different implementations.

Caching in Python

By freezing the state of data/outputs/results in cache, reproducibility of code is guaranteed. In practice, multiple caching methods with example Python code are introduced (**https://www.scaler.com/topics/python-cache**): using Python dictionary, using @lru_cache decorator, and using external caching services. The example displays utilization of the decorator @lru_cache (part of the functools module in Python), which is useful in the case of reproducibility and recursion making it faster.

The decorator stores the output of each function call (findNthFib), and when it encounters a recursive function call whose output is present in the cache, it does not compute the output of the recursive call again. It instead uses the output stored in the cache, making the computation time faster and outputs reproducible.

Python Markdown

Firstly, install Python-Markdown under your environment in the terminal by running the command line, **$pip install markdown**.

- Create Markdown string.
- Convert the markdown (markdown_string) to HTML (html) using the markdown method from the markdown package.
- Create a sample.html file and write the HTML (html) to it.

For more detailed examples, see https://www.scaler.com/topics/python-cache to convert a Markdown file to HTML in Python.

R Quarto

Quarto is a multi-language, next generation version of R Markdown from RStudio, with many new features and capabilities. The latest release (download from https://posit.co/download/rstudio-desktop/) of RStudio is highly recommended when running Quarto within RStudio. See the reference (https://quarto.org/docs/computations/r.html#chunk-options) for more details.

Differences between R Markdown and R Quarto

Quarto **chunk options** are typically included in special comments at the top of code chunks rather than within the line that begins the chunk. Quarto uses this approach to both better accommodate longer options like “fig-cap” as well as to make it straightforward to edit chunk options within more structured editors that only have a difficult way to edit chunk metadata.

Quarto includes many more built-in **output formats** (and many more options for customizing each format). Quarto also has native features for special project types like websites, books, and blogs (rather than relying on external packages). In Quarto, “format” key rather than the “output” key is used as below.

R Quarto	R Markdown

title: "My Document"
format:
  html:
    toc: true
    number-sections: true
      css: styles.css
	title: "My Document"
	output:
	    html_document:
	      toc: true
	      number_sections: true
	      css: styles.css

Table 25: Comparison of R Quarto and R Markdown

References

1. https://builtin.com/data-science/numpy-random-seed
2. https://quarto.org/docs/computations/r.html
3. https://enjoymachinelearning.com/blog/is-gpt-3-deterministic/?expand_article=1
4. https://152334h.github.io/blog/non-determinism-in-gpt-4/
5. https://wandb.ai/sauravmaheshkar/RSNA-MICCAI/reports/How-to-Set-Random-Seeds-in-PyTorch-and-Tensorflow--VmlldzoxMDA2MDQy
6. https://realpython.com/lru-cache-python/#:~:text=One%20way%20to%20implement%20an,the%20least%20recently%20used%20entry.
7. https://www.scaler.com/topics/python-cache/

Chapter 13: Deep Learning Applications

Deep Learning in Chemistry

Compound Design in Chemistry

In chemistry, compound design utilizes deep learning algorithms to generate novel molecules with desired properties by analyzing vast chemical databases, predicting their potential interactions, and suggesting modifications to existing compounds, effectively accelerating the drug discovery process by identifying promising candidates for further testing in a virtual environment.

Generative models

Deep learning (DL) models like generative adversarial networks (GANs) can generate new molecular structures that adhere to specific chemical rules while optimizing for desired properties like binding affinity to a target protein.

Predicting properties

DL can predict various chemical properties of a molecule like solubility, toxicity, and metabolic stability, allowing for early screening and optimization of potential compounds.

Virtual screening

By computationally docking molecules into a target protein, DL can rapidly identify promising compounds from large chemical libraries for further experimental validation.

Lead optimization

DL can suggest modifications to existing lead compounds to enhance their potency and selectivity while minimizing unwanted side effects.

Retrosynthesis planning

DL can assist chemists in designing synthetic routes by predicting the necessary starting materials and reaction steps to access a desired molecule.

Table 26: Benefits and Challenges for DL in Compound Design

Benefits	Challenges & considerations
Increased efficiency: DL can significantly accelerate the drug discovery process by rapidly generating and evaluating large numbers of potential compounds. Cost reduction: By identifying promising candidates early in the development process, DL can reduce the cost of experimental testing. Novel molecule discovery: DL can generate unique molecular structures that might not be readily accessible through traditional methods.	Data quality: Training DL models requires large datasets of high-quality chemical information to ensure accurate predictions. Interpretability: Understanding how DL makes decisions and why it selects certain molecules can be challenging, requiring additional analysis to validate results. Synthetic feasibility: AI-generated molecules should be considered for their synthetic accessibility to ensure practicality in real-world applications.

References

1. Galushka, M., Swain, C., Browne, F. *et al.* Prediction of chemical compounds properties using a deep learning model. *Neural Comput & Applic* **33**, 13345–13366 (2021). https://doi.org/10.1007/s00521-021-05961-4
2. Miljković F, Rodríguez-Pérez R, Bajorath J. Impact of Artificial Intelligence on Compound Discovery, Design, and Synthesis. ACS Omega. 2021 Nov 29;6(49):33293-33299. doi: 10.1021/acsomega.1c05512. PMID: 34926881; PMCID: PMC8674916.
3. Han, Minhi and Joung, Joonyoung F. and Jeong, Minseok and Choi, Dong Hoon and Park, Sungnam. Generative Deep Learning-Based Efficient Design of Organic Molecules with Tailored Properties. ACS Central Science. **11**(2), 219-227 (2025). https://doi.org/10.1021/acscentsci.4c00656

DeepChem

DeepChem is an open-source DL platform/ML library (https://github.com/deepchem/deepchem) designed for drug discovery, materials science, quantum chemistry, and biology. DeepChem provides a comprehensive set of tools for harnessing DL across various examples of drug development, including prediction of solubility of small drug-like molecules and prediction of binding affinity for small molecules to protein targets.

In DeepChem, "solubility" refers to the ability of a molecule to dissolve in a solvent, typically water, in which DL models within DeepChem are trained to predict based on a molecule's structure, estimating how much of a given molecule can dissolve in a specific amount of solvent. It is commonly accomplished using datasets like the Delaney solubility dataset which contains measured solubility values for various molecules.

To predict binding affinity to protein targets in DeepChem, the "DeepDTA" model can be utilized, leveraging convolutional neural networks (CNNs) to learn representations from protein sequence information and ligand features, to predict the binding affinity between a protein and a small molecule based on their molecular structures. Data preparation for model training includes protein sequences (e.g., as FASTA files) and corresponding ligand structures (e.g. SMILES strings) along with their known binding affinities (pIC50 or Kd values).

DeepChem leverages graph neural networks (GNNs) which effectively represent molecular structures, where atoms are nodes and bonds are edges, and learns complex relationships between molecular features and solubility values, allowing for accurate predictions based on a molecule's chemical structure alone. A large dataset of known drug-like molecules with experimentally measured solubility values is required to train the model. The model can predict the solubility of a new molecule as a continuous value (e.g., log(solubility)) or categorize it into solubility classes (e.g., high/medium/low). By analyzing the model's weights, insights can be gained about which molecular features most significantly influence solubility.

Notably, DeepChem can be implemented for virtual screening by learning a vast number of molecules libraries (e.g. MoleculeNet suite of curated datasets, containing properties of > 700k compounds) to identify the structures that are most likely to show the desired characteristics.

REFERENCES

1. https://deepchem.readthedocs.io/en/latest/get_started/installation.html
2. https://deepchem.io/

DeepTox

DeepTox is a pipeline to predict toxic effects of chemical compounds. First, DeepTox merges and normalizes the chemical representations of compounds then derives distinct fragments for data cleaning and quality control. Second, it computes many chemical descriptors that are used as input to Deep Neural Networks (DNNs). Third, it trains models including DNNs and complementary machine-learning models such as SVMs, random forests, and elastic nets. DeepTox evaluates the chemicals by cluster cross-validation and combines the best of the models by ensembling. Lastly, DeepTox predicts the toxicity of new compounds, by discovering toxicophores being represented in hidden neurons. Go to the DeepTox website (https://ml.jku.at/research/DeepTox/index.html) for more details including software.

REFERENCE

1. https://www.frontiersin.org/journals/environmental-science/articles/10.3389/fenvs.2015.00080/full

Chemical Language Models (CLMs)

To generate novel chemical entities with specific properties in chemical space is a time-consuming process and expensive task due to the irregular distribution of molecules, where even small molecular changes can derive substantial changes in physicochemical properties. DL can learn chemistry rules for prediction or generative tasks by using different molecular representations that can be comprehensive in their natural form by humans such as SMILES, InChl, DeepSMILES, and SELFIES.

DL methods have proven to be successful in multiple fields of chemistry, such as QSAR (Quantitative Structure-Activity Relationships), QSPR (Quantitative Structure-Property Relationships), and molecular generative models. Particularly, deep generative models have the capability to learn implicit chemical knowledge from data by identifying structural patterns such as valency rules, reactive groups, molecular conformations, hydrogen bond donors and acceptors to produce molecules with specific properties. Unlike the classic methods which require human knowledge, deep generative models are independent and less prone to generating molecules that are unavailable for chemical synthesis due to unstable groups. Several DL architectures like RNNs, transformers, VAEs, GANs can generate targeted molecules and investigate regions of the chemical domain via biased models, to produce new molecules that meet specific conditions, such as toxicity, solubility, and other relevant characteristics.

REINVENT4 is a modern AI-driven generative molecular design framework, which is an open-source framework for the de novo design of small molecules by generating novel chemical structures from scratch. Utilizing RNN and transformer architectures, the software enables researchers to perform tasks like de novo molecule design, R-group replacement, library design, linker design, scaffold hopping, and molecule optimization, essentially accelerating the drug discovery process by creating new molecules with desired properties based on user-defined constraints. REINVENT4 utilizes reinforcement learning algorithms where the generated molecules are evaluated based on their predicted properties, navigating the model to create better molecules in subsequent iterations. Small molecules represented as SMILES strings, are processed by the AI model to generate new molecules. Although not strictly necessary, using a GPU can significantly speed up the molecule generation process.

NVIDIA developed MolMIM, as a part of the NVIDIA BioNeMo framework, which is a latent variable generative model for small molecule drug development. It is trained in an unsupervised way over a large-scale dataset of molecules in the form of SMILES strings. MolMIM utilizes a transformer (a probabilistic autoencoder) architecture to learn an informative fixed-size latent space using Mutual Information Machine (MIM) learning. This facilitates generation of new small molecule drug candidates by optimizing molecules based on user-defined properties, while creating molecules with desired characteristics for drug discovery and leveraging a "latent space" to manipulate molecular structures effectively. The input molecule as a SMILES string is encoded into a fixed-length latent representation. Then, the latent representation is manipulated using optimization algorithms such as CMA-ES to create new molecules with desired properties. Lastly, the optimized latent representation is decoded back into a SMILES string, representing a new molecule. MolMIM accelerates the process of identifying potential drug candidates by generating diverse molecules with desired properties and allows users to define their own scoring functions to prioritize specific molecular properties.

Although DL architectures can generate large virtual libraries of molecules that are grammatically correct, creating molecules is a significant challenge for DL, when working with biological targets or rare molecules that lack sufficient data for model training. To overcome these challenges, biased strategies such as Reinforcement learning, Transfer learning, and Conditional learning, have been incorporated with DL models. The incorporation of DL models enables the exploration of chemical space in specific directions and the production of molecules that are feasible to synthesize or have desired properties.

RNNs and their variants, like LSTMs and GRUs, brought attention to understanding the distribution of sequential data, such as SMILES strings, because the methods can learn and remember patterns over time. SMILES strings are inherently sequential, representing the structure of a molecule as a series of characters. RNNs are designed to process the sequential data by feeding the output of one-time step back as input to the next, allowing them to capture the order and dependencies within the string. However, the popularity of RNNs as molecular generative models has diminished due to the long training time and risk of degraded performance associated with long-term dependencies within the sequence. On the other hand, implementation of transformers shows a major increase as molecule generation models in recent years, due to their self-attention mechanism.

Meanwhile, for exploring specific regions of chemical space, there are CLMs based on biased model weights to incorporate knowledge on the general grammar and syntax of molecular representation, in addition to the specific configuration and patterns of molecules that exhibit specific physicochemical properties or bioactivities. The biased models are primarily utilized to generate targeted molecules when the available data alone is not sufficient to train unbiased DL models. They are crucial for rare or de-novo targets that do not have enough data to train DL models. In conclusion, among the biased models, transfer leaning has been the most utilized technique, when ~1,000 molecular entities are available for optimization. Also, combining multiple biased models into deep generative models is a promising method for targeted molecular generation, with improved chemical properties or bioactivities and longer mobility within chemical space.

One of the most recent DL architectures is **DRAGONFLY**, which leverages a LSTM as CLMs to generate novel bioactive molecules (i.e. ligands with desired bioactivity profiles) and graph neural networks to encode the information of interaction between ligands and one or multiple specific macromolecular targets. The method integrates information about receptors from graph neural networks, which are utilized to train the LSTM and learn the distribution of SMILES- and SELFIES-strings that encompasses each ligand based on its spatial 2D and 3D information about the receptor. DRAGONFLY optimizes the drug-target interactome-based de novo design between drug targets and their ligands while maintaining high-quality distribution metrics. Python codes are available in a public github, GitHub - ETHmodlab/dragonfly_gen at Dragonfly_1.0.

References

1. Flores-Hernandez, H., Martinez-Ledesma, E. A systematic review of deep learning chemical language models in recent era. *J Cheminform* **16**, 129 (2024). https://doi.org/10.1186/s13321-024-00916-y
2. Atz, K., Cotos, L., Isert, C. *et al.* Prospective de novo drug design with deep interactome learning. *Nat Commun* **15**, 3408 (2024). https://doi.org/10.1038/s41467-024-47613-w
3. Loeffler, H.H., He, J., Tibo, A. *et al.* Reinvent 4: Modern AI–driven generative molecule design. *J Cheminform* **16**, 20 (2024). https://doi.org/10.1186/s13321-024-00812-5
4. https://github.com/MolecularAI/REINVENT4
5. https://docs.nvidia.com/nim/bionemo/molmim/latest/overview.html
6. Reidenbach, D., Livne, M., Ilango, R.K., Gill, M., Israeli, J. Improving small molecule generation using mutual information machine. arXiv (2023). https://arxiv.org/abs/2208.09016

General Computer Vision: Image Segmentation

Image segmentation is a computer vision system that divides a digital image into discrete clusters of pixels, known as image segments, for object detection. While traditional image segmentation techniques exercise high-level visual feature values of each pixel (e.g., brightness, color, contrast, or intensity) with simple ML algorithms to quickly discover object boundaries and background regions, DL-based segmentation methods trained by annotated datasets can provide greater accuracy and more complicated image analysis.

The **Segment Anything Model (SAM)** is a cutting-edge AI model that can automatically detect and segment objects in images with or without real-time performance. It was developed by Meta AI and released in April 2023 (Segment Anything | Meta AI). SAM uses deep learning neural networks to analyze images and segment an entire image or specific objects within an image. SAM can create multiple masks for ambiguous prompts and adapt to new images and tasks without additional training. SAM was trained on the expansive 11 million carefully curated images and 1.1 billion segmentation masks (SA-1B). This massive volume of training resources makes the model highly robust in identifying object boundaries and differentiating between various objects across domains, even when it might have never seen them before.

It's open source and released under an Apache 2.0 license. SAM can be implemented through ArcGIS (Segment Anything Model (SAM) - Overview), Ultralytics YOLO Docs (https://docs.ultralytics.com/models/sam/), or GitHub (https://github.com/facebookresearch/segment-anything).

In general, previous segmentation models need to be trained from scratch using a dataset labeled with objects of interest, which is time-consuming. In contrast, SAM aims to generate a fundamental model that can be utilized to segment anything using zero-shot transfer and generalize across domains without additional training. SAM can be fine-tuned using SamLoRA architecture in ArcGIS (see Finetune Segment Anything Model (SAM) using SamLoRA | ArcGIS API for Python for more details).

REFERENCE

1. https://www.ibm.com/think/topics/image-segmentation#:~:text=Image%20segmentation%20is%20a%20computer,of%20interest%20in%20satellite%20images.

Medical Image Analysis

Various medical imaging modalities are used in clinical practice, each offering unique information and serving specific diagnostic purposes. Some of the most common modalities include magnetic resonance imaging (MRI), positron emission tomography (PET), chest X-rays, CT, and fundus imaging. DL techniques have been successfully applied to these modalities for tasks such as image segmentation, classification, reconstruction, and registration.

Foundation Models in Medical Imaging

Foundation models (FMs) are large-scale, pre-trained AI models that learn from vast, unlabeled datasets without relying solely on manual annotation. They learn general-purpose visual features that can be adapted to specific clinical imaging tasks with minimal labeled data (limited additional training). FMs tackle the challenge of data scarcity in medical imaging and have shown promise in pathology, radiology, and ophthalmology.

Two powerful backbone architectures, like Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), are the most widely used in FMs for medical image analysis. Their strengths and weaknesses are:

Table 27: CNNs vs ViTs

CNNs	ViTs
Strengths • Efficient local feature extraction (locality): leverage convolutional kernels to capture local patterns like edges and textures, which are crucial for tasks where fine details and spatial hierarchies are important. • Parameter sharing reduces the number of learnable parameters, making CNNs less complex and easier to train. • Computational efficiency: CNNs are optimized for image data, allowing CNNs to be generally more efficient than ViTs for most tasks. • Better performance on smaller datasets: generalize well even when dealing with limited amounts of data due to the built-in inductive biases of locality and parameter sharing, for tasks where local patterns are significant. • Translation invariance: CNNs are inherently robust to small changes in object position within an image. **Weaknesses** • Difficulty capturing long-range dependencies: CNNs struggle to capture relationships across the entire image due to their localized receptive fields. • CNNs may not be ideal for certain tasks, like anomaly detection where detailed spatial information is critical. • Fixed kernel sizes can limit the ability to detect features at various scales.	**Strengths** • Capture long-range dependencies: effectively capture global context by processing images as sequences of patches and using self-attention mechanisms to model relationships between them. • Flexibility and scalability: scalability with larger datasets and deeper architectures makes ViTs highly effective for large-scale vision tasks. Also, different tasks and domains can be adapted. • Better generalization: ViTs show potential in generalizing unseen data, though this may require extensive fine-tuning. • Analyzing large-scale patterns: suitable for tasks requiring understanding of global context. **Weaknesses** • Computationally intensive: the self-attention mechanism can be computationally expensive for high-resolution images. • Require large datasets: to perform well and avoid overfitting. • Minimal inductive bias: its flexibility makes it harder to train on smaller datasets. • ViTs may not be as effective in capturing the fine-grained details in images that are crucial for certain tasks.

Self-Supervised Learning

The next step, a core component of FMs, is self-supervised learning (SSL) that enables the models to learn from unlabeled data by solving auxiliary tasks, like predicting missing information or identifying similar representations. There are three different strategies: discriminative, generative, and multimodal SSL.

Discriminative SSL aims to learn representations that are discriminative, meaning that it captures the differences between instances or variations of data, like classification, detection, or segmentation. The discriminative nature makes it more aligned with traditional classification tasks, but it avoids the need for manual labels.

Generative SSL focuses on learning a comprehensive understanding of the data's structure and characteristics, the underlying data distribution, to be able to generate or reconstruct new data. The model learns to reconstruct the original input or predict missing information based on the available parts. For example, Masked Image Modeling (MIM) masks out parts of an image and training trains the model to reconstruct the missing information. The approach shares similarities with generative models like VAEs and GANs.

Multimodal SSL learns representations that capture the shared information across multiple modalities (e.g., medical images and text reports) simultaneously. This approach leverages the relationships between different modalities. For instance, the model might be trained to predict medical terms from an image or generate image descriptions from text. It can lead to more comprehensive and robust representations that can be used for various tasks, like image-to-text generation or image retrieval.

Adaptation

Lastly, adaptation approaches are executed to adapt FMs for downstream clinical tasks such as prompt engineering (guiding the model using natural language or structured text inputs), linear probing (training a simple linear classifier on top of the pre-trained FM's embeddings), task-specific heads (adding smaller, task-specific neural networks on top of the frozen FM layers), parameter-efficient fine-tuning (PEFT: updating only a limited number of the model's parameters), and full fine-tuning (updating all of the model's parameters for a specific task). [See chapter 10 for additional details on DL adaptation]

Applications

Application of FMs in pathology, radiology, and ophthalmology, includes:

1. Automated image interpretation and report generation
2. Improved patient communication through simplified medical reports
3. Advanced diagnostic support, including lesion segmentation and disease classification
4. Streamlined workflow and data management
5. Unlocking population insights and disease prediction

Various SSL techniques and FMs for **pathology** image analysis have been developed. For example, self-distillation methods like DINO (Distillation with NO labels) are used to learn representations by comparing different augmented views of the same image from aggregating visual tokens across different scales. FMs are proposed to capture global context in WSIs (Whole Slide Images). FMs and SSL are showing promise in various pathology tasks, including disease detection and classification, organ transplant assessment, rare disease analysis, automated reporting, and personalized medicine. However, challenges related to data quality, model robustness, and regulatory hurdles need to be overcome for successful widespread adoption.

Research on SSL and FM development in **radiology** is driven by the vast amounts of unlabeled data available in radiology archives and the challenge of obtaining expert annotation. SSL utilizes the abundance of unlabeled medical images in radiology archives to train models on the data itself and significantly decreases the reliance on time-consuming and expensive manual annotations by radiologists. FMs, often trained with SSL, show promise in radiology by learning generalizable representations that can be fine-tuned for specific tasks with limited labeled data. Notable examples include RadCLIP (Lu et al. 2024) and RadFM (Wu et al. 2023) for vision-language FMs, leveraging multi-domain radiology.

SSL and FM development in **ophthalmology** explore data annotation challenges, enhancement of diagnostic accuracy, and patient-care improvement: 1. Learning discriminative representation, so called contrastive learning, by comparing different views or samples of the same or similar ophthalmic data, 2. Reconstructing masked regions of an image to learn valuable features, 3. Leveraging properties of the image itself as a supervisory signal. FMs are capable of handling diverse ophthalmic imaging modalities and tasks, such as RETFound (an open-source FM specifically designed for retinal images in various tasks, including disease diagnosis, prognosis, and predicting systemic events, Zhou et al. 2023), VisionFM (it has been trained on a massive dataset of 3.4M images from over 0.5M individuals, covering diverse ophthalmic diseases, imaging modalities, and devices, Qiu et al. 2023), and Multimodal FMs (e.g., fundus photography and optical coherence tomography – OCT and incorporating textual domain knowledge).

Benefits and challenges of FM applications in medical image analysis (Table 28):

Benefits	Challenges
• Improved diagnostic accuracy • Enhanced generalizability • Reduced annotation burden • Facilitating research and clinical applications	• Data heterogeneity • Model interpretability • Computational resources • Regulatory and ethical considerations

References:

1. Veldhuizen, Vivien & Botha, Vanessa & Lu, Chunyao & Erdal Cesur, Melis & Groot Lipman, Kevin & Jong, Edwin & Horlings, Hugo & Sanchez, Clárisa & Snoek, Cees & Mann, Ritse & Marcus, Eric & Teuwen, Jonas. (2025). Foundation Models in Medical Imaging -- A Review and Outlook. 10.48550/arXiv.2506.09095.
2. Chaoyi Wu, Xiaoman Zhang, Ya Zhang, Yanfeng Wang, and Weidi Xie. Towards Generalist Foundation Model for Radiology by Leveraging Web-scale 2D&3D Medical Data, November 2023. URL http://arxiv.org/ abs/2308.02463.
3. Zhixiu Lu, Hailong Li, and Lili He. Radclip: Enhancing radiologic image analysis through contrastive languageimage pre-training. arXiv preprint arXiv:2403.09948, 2024
4. Yukun Zhou, Mark A. Chia, Siegfried K. Wagner, Murat S. Ayhan, Dominic J. Williamson, Robbert R. Struyven, Timing Liu, Moucheng Xu, Mateo G. Lozano, Peter Woodward-Court, Yuka Kihara, Andre Altmann, Aaron Y. Lee, Eric J. Topol, Alastair K. Denniston, Daniel C. Alexander, and Pearse A. Keane. A foundation model for generalizable disease detection from retinal images. Nature, 622(7981):156–163, October 2023. ISSN 1476-4687. doi:10.1038/s41586-023-06555-x.
5. Jianing Qiu, Jian Wu, Hao Wei, Peilun Shi, Minqing Zhang, Yunyun Sun, Lin Li, Hanruo Liu, Hongyi Liu, Simeng Hou, Yuyang Zhao, Xuehui Shi, Junfang Xian, Xiaoxia Qu, Sirui Zhu, Lijie Pan, Xiaoniao Chen, Xiaojia Zhang, Shuai Jiang, Kebing Wang, Chenlong Yang, Mingqiang Chen, Sujie Fan, Jianhua Hu, Aiguo Lv, Hui Miao, Li Guo, Shujun Zhang, Cheng Pei, Xiaojuan Fan, Jianqin Lei, Ting Wei, Junguo Duan, Chun Liu, Xiaobo Xia, Siqi Xiong, Junhong Li, Benny Lo, Yih Chung Tham, Tien Yin Wong, Ningli Wang, and Wu Yuan. Visionfm: a multi-modal multi-task vision foundation model for generalist ophthalmic artificial intelligence, 2023.

Deep Learning models for MRI (Magnetic Resonance Imaging)

Recent deep learning developments in MRI analysis focus on improving accuracy and efficiency in tasks like tumor detection, classification, and segmentation, with advancements in areas like attention mechanisms, data augmentation, and multi-modal approaches.

Deep learning models are used for tumor segmentation, quantification, and classification in brain MRI. MONAI (https://github.com/Project-MONAI/MONAI/) is the most widely used PyTorch-based, open-source framework for deep learning in healthcare imaging. Moreover, researchers explore the potential of deep learning to predict tumor type, grade, genetic mutations, and patient survival outcomes for personalized medicine. Studies have demonstrated the effectiveness of automated deep learning in 7T MRI analysis for enhancing image quality via noise reduction and artifact suppression, accelerating scans from sparse sampling, and improving analysis, particularly in neuroscience and musculoskeletal imaging to address challenges of ultra-high-field MRI. 7T MRI, combined with deep learning, can help identify neurodegenerative changes and potential biomarkers in conditions like Alzheimer's disease and Parkinson's disease.

Researchers are developing fully automated brain tumor detection models using deep learning algorithms like YOLOv7 (https://www.kaggle.com/code/rohitgadhwar/brain-tumor-object-detection-yolov7 for tutorials). Attention mechanisms, like CBAM (Convolutional Block Attention Module), can be combined with YOLOv7 to further enhance the performance of brain tumor detection models. Data augmentation techniques are used to improve performance on limited datasets. Deep learning models can automatically extract meaningful features from medical images, enabling efficient and accurate interpretation.

Deep learning networks are used for the prognosis assessment of patients regarding cancer or non-cancer diseases using MR imaging. It has been shown that machine learning can be helpful for response prediction in intensity-modulated radiation therapy of the prostate using radiomic features prior to irradiation, as well as post-irradiation.

Deep learning is also being used for medical image enhancement, including denoising, super-resolution, and MRI bias field correction. Deep learning reconstruction enables faster MRI scans and improves image quality.

Lastly, deep learning is being used for the prediction of biochemical recurrence after radical prostatectomy. Further applications include prediction of MSI (Microsatellite Instability) status of rectal cancer preoperatively and image registration, such as the registration of prostate MRI and histopathology images.

References:

1. Dorfner, F.J., Patel, J.B., Kalpathy-Cramer, J. *et al.* A review of deep learning for brain tumor analysis in MRI. *npj Precis. Onc.* **9**, 2 (2025). https://doi.org/10.1038/s41698-024-00789-2
2. Liu Z, Zhou X, Tao S, Ma J, Nickel D, Liebig P, Mostapha M, Patel V, Westerhold EM, Mojahed H, Gupta V, Middlebrooks EH. Application of Deep Learning Accelerated Image Reconstruction in T2-weighted Turbo Spin Echo Imaging of the Brain at 7T. AJNR Am J Neuroradiol. 2025 Jan 20:ajnr.A8662. doi: 10.3174/ajnr.A8662. Epub ahead of print. PMID: 39832954.
3. Pulkit Khandelwal, Michael Tran Duong, Shokufeh Sadaghiani, Sydney Lim, Amanda E. Denning, Eunice Chung, Sadhana Ravikumar, Sanaz Arezoumandan, Claire Peterson, Madigan Bedard, Noah Capp, Ranjit Ittyerah, Elyse Migdal, Grace Choi, Emily Kopp, Bridget Loja, Eusha Hasan, Jiacheng Li, Alejandra Bahena, Karthik Prabhakaran, Gabor Mizsei, Marianna Gabrielyan, Theresa Schuck, Winifred Trotman, John Robinson, Daniel T. Ohm, Edward B. Lee, John Q. Trojanowski, Corey McMillan, Murray Grossman, David J. Irwin, John A. Detre, M. Dylan Tisdall, Sandhitsu R. Das, Laura E. M. Wisse, David A. Wolk, Paul A. Yushkevich; Automated deep learning segmentation of high-resolution 7 Tesla postmortem MRI for quantitative analysis of structure-pathology correlations in neurodegenerative diseases. *Imaging Neuroscience* 2024; 2 1–30. doi: https://doi.org/10.1162/imag_a_00171
4. Gassenmaier S, Küstner T, Nickel D, Herrmann J, Hoffmann R, Almansour H, Afat S, Nikolaou K, Othman AE. Deep Learning Applications in Magnetic Resonance Imaging: Has the Future Become Present? Diagnostics (Basel). 2021 Nov 24;11(12):2181. doi: 10.3390/diagnostics11122181. PMID: 34943418; PMCID: PMC8700442.
5. Abdusalomov AB, Mukhiddinov M, Whangbo TK. Brain Tumor Detection Based on Deep Learning Approaches and Magnetic Resonance Imaging. Cancers (Basel). 2023 Aug 18;15(16):4172. doi: 10.3390/cancers15164172. PMID: 37627200; PMCID: PMC10453020.
6. Abdusalomov AB, Mukhiddinov M, Whangbo TK. Brain Tumor Detection Based on Deep Learning Approaches and Magnetic Resonance Imaging. Cancers (Basel). 2023 Aug 18;15(16):4172. doi: 10.3390/cancers15164172. PMID: 37627200; PMCID: PMC10453020.

Deep Learning Models for PET (Positron Emission Tomography)

Like MR image data analysis, harnessing deep learning for PET image data analysis concentrates on improving image quality, reducing radiation exposure, and automating tasks, including lesion detection and segmentation, with advancements in techniques such as denoising, reconstruction, and multi-modal data analysis.

DL methods, like GAN and UNET, are very popular methods to reduce the noise of low-dose PET images and to restore the image quality to the same level as a full-dose PET. DL methods outperform pre-existing baselines on post-processing and iterative reconstruction in terms of higher image quality, shorter processing time, and greater generalizability. By learning a data-driven mapping from low-quality to high-quality images, DL models effectively denoise low-count PET images, from reducing noise and improving image quality. For instance, neural networks were implemented for post-processing (denoising), direct reconstruction, and iterative reconstruction methods. DL methods such as 3D UNET can also predict standard-dose PET images from low-dose images and generate virtual high-dose images, which improve lesion detectability and hence clinical diagnosis. Low-dose PET imaging can be achieved by reducing the amount of injected tracer and scan time while keeping image quality and lesion detectability.

The PET scanner requires reconstruction algorithms to obtain a tomographic representation. Physics-informed DL can be integrated into the image reconstruction process, replacing conventional components with data-driven alternatives, such as for regularization, to resolve the limitations caused in traditional methods. Also, feature extraction and lesion detection can be automated through improving diagnostic efficacy and reducing the workload for radiologists.

For multi-modal data integration, PET and CT data can be integrated by recent DL models like attention-based models, generative models, multi-modal models, graph convolutional networks, and transformers for improved diagnosis and treatment planning. To predict patient outcomes and personalize treatment, radiomic features from PET images can be extracted and analyzed by DL.

Some DL models aim to automate tumor detection and segmentation in PET images, for improving the accuracy and efficiency of the tasks. Also, DL can be used to classify different types of tumors and predict patient response to treatment based on PET imaging data. Lastly, DL models are proposed to automatically evaluate the quality of PET images from which clinical research is potentially accelerated.

NiftyPET is a high-throughput software platform and Python package designed for the reconstruction, manipulation, processing, and analysis of PET images, particularly for brain imaging in dementia using amyloid tracers and whole-body imaging with PET and MRI data (https://niftypet.readthedocs.io/en/latest/#:~:text=NiftyPET%20is%20a%20software%20platform,the%20use%20of%20amyloid%20tracers.). In addition, other open-source tools exist for DL-powered PET imaging analysis, such as PyTomography (https://github.com/PyTomography/PyTomography), CASToR (https://www.castor-project.org/links), STIR (https://github.com/UCL/STIR/wiki/), and MCGPU-PET (https://github.com/DIDSR/MCGPU-PET/).

References:

1. Hashimoto, F., Onishi, Y., Ote, K. *et al.* Deep learning-based PET image denoising and reconstruction: a review. *Radiol Phys Technol* **17**, 24–46 (2024). https://doi.org/10.1007/s12194-024-00780-3
2. Seyyedi, N., Ghafari, A., Seyyedi, N. *et al.* Deep learning-based techniques for estimating high-quality full-dose positron emission tomography images from low-dose scans: a systematic review. *BMC Med Imaging* **24**, 238 (2024). https://doi.org/10.1186/s12880-024-01417-y
3. Liu J, Ren S, Wang R, Mirian N, Tsai YJ, Kulon M, Pucar D, Chen MK, Liu C. Virtual high-count PET image generation using a deep learning method. Med Phys. 2022 Sep;49(9):5830-5840. doi: 10.1002/mp.15867. Epub 2022 Aug 13. PMID: 35880541; PMCID: PMC9474624.
4. Artesani, A., Bruno, A., Gelardi, F. *et al.* Empowering PET: harnessing deep learning for improved clinical insight. *Eur Radiol Exp* **8**, 17 (2024). https://doi.org/10.1186/s41747-023-00413-1
5. Illimoottil M, Ginat D. Recent Advances in Deep Learning and Medical Imaging for Head and Neck Cancer Treatment: MRI, CT, and PET Scans. Cancers (Basel). 2023 Jun 21;15(13):3267. doi: 10.3390/cancers15133267. PMID: 37444376; PMCID: PMC10339989.
6. Maryam Fallahpoor, Subrata Chakraborty, Biswajeet Pradhan, Oliver Faust, Prabal Datta Barua, Hossein Chegeni, Rajendra Acharya, Deep learning techniques in PET/CT imaging: A comprehensive review from sinogram to image space, Computer Methods and Programs in Biomedicine, Volume 243, 2024, 107880, ISSN 0169-2607, https://doi.org/10.1016/j.cmpb.2023.107880.
7. Lee J, Ha S, Kim REY, Lee M, Kim D, Lim HK. Development of Amyloid PET Analysis Pipeline Using Deep Learning-Based Brain MRI Segmentation-A Comparative Validation Study. Diagnostics (Basel). 2022 Mar 2;12(3):623. doi: 10.3390/diagnostics12030623. PMID: 35328176; PMCID: PMC8947654.

Deep Learning Models for X-rays

Deep learning developments in X-ray imaging data analysis are not much different from the other two image modalities above in improving accuracy, efficiency, and accessibility, with applications ranging from disease detection and diagnosis to image reconstruction and workflow optimization.

DL algorithms are proposed to detect abnormalities in chest X-rays, such as lung diseases, with high accuracy, potentially matching or exceeding the performance of radiologists. AI-powered tools can analyze X-ray images more quickly, while reducing diagnosis time and improving workflow efficiency. DL can be used to automate image analysis and reporting and improve the quality and resolution of X-ray images, even in challenging conditions, by leveraging high-resolution power and advanced X-ray detectors. Moreover, AI streamlines the X-ray workflow, leading to fewer repeat exams, exam rejects, and patient positioning errors, increasing throughput and capacity. Kumar Mall et al. exclusively summarized examples of DL applications for various tasks, including detection, segmentation, and registration across multiple diseases.

Deep neural networks-based Computer-Aided Detection (CAD) systems benefit radiologists and physicians by enhancing precision in identification of subtle patterns or irregularities in X-ray images that might be missed by human observers. The degree of precision in segmentation of anatomical structures or areas of interest in X-ray images has been improved for accurate quantitative analysis. In a similar way, DL is a popular approach to learn to recognize and categorize different anatomical features, lesions, or anomalies in X-ray images. For non-medical usage, DL is developed for X-ray security imaging to detect threats in baggage, even across different scanner types, by using meta-transfer learning-driven tensor-shot detectors.

TorchXRayVision (https://github.com/mlmed/torchxrayvision) is a public library designed for working with chest X-ray datasets and DL models, providing pre-trained models and tools for efficient analysis. MCGPU (Monte Carlo GPU, https://github.com/DIDSR/MCGPU) is a GPU-accelerated Monte Carlo simulation tool that is developed by the FDA and used for simulating X-ray imaging devices and processes. It's particularly useful for simulating X-ray image formation and radiation-dose distribution in various imaging modalities like CT (Computed Tomography), DBT (Digital Breast Tomosynthesis), and PET (Positron Emission Tomography).

References:

1. Pawan Kumar Mall, Pradeep Kumar Singh, Swapnita Srivastav, Vipul Narayan, Marcin Paprzycki, Tatiana Jaworska, Maria Ganzha, A comprehensive review of deep neural networks for medical image processing: Recent developments and future opportunities, Healthcare Analytics, Volume 4, 2023, 100216, ISSN 2772-4425, https://doi.org/10.1016/j.health.2023.100216.
2. Anderson, P.G., Tarder-Stoll, H., Alpaslan, M. *et al.* Deep learning improves physician accuracy in the comprehensive detection of abnormalities on chest X-rays. *Sci Rep* **14**, 25151 (2024). https://doi.org/10.1038/s41598-024-76608-2
3. https://www.gehealthcare.com/insights/article/deep-learning-image-reconstruction-improving-iq-and-patient-outcomes-in-radiology#:~:text=AI%2Denabled%20X%2Dray%20image,variability%2C%20increasing%20throughput%20and%20capacity.
4. https://signalprocessingsociety.org/publications-resources/blog/recent-advances-deep-learning-within-x-ray-security-imaging#:~:text=The%20previous%20models%20had%20certain,the%20SIXray%20and%20GDXray%20datasets.
5. https://cdrh-rst.fda.gov/mcgpu-gpu-accelerated-monte-carlo-x-ray-imaging-simulator#:~:text=MCGPU%20%5B1%2C2%5D%20is,virtual%20imaging%20trial%20project7.

Deep Learning and CT (Computed Tomography) scans

Recent developments in DL for CT scans include advancements in CNNs and their application to various tasks such as tumor detection, organ segmentation, and disease classification. In addition, there has been a push towards using RNNs for sequential data analysis and understanding dynamic changes in CT images over time. RNNs can track changes in image features over time, predict future states, and handle variable-length input sequences. Combining the strengths of CNNs and RNNs, researchers are developing hybrid architectures for analyzing dynamic and complex patterns in CT images. Furthermore, new generative AI models are explored for reconstructing high-quality 3D CT scans from sparse views (a smaller number of views), potentially reducing radiation exposure.

Reference:

1. Ahmad IS, Dai J, Xie Y, Liang X. Deep learning models for CT image classification: a comprehensive literature review. Quant Imaging Med Surg. 2025 Jan 2;15(1):962-1011. doi: 10.21037/qims-24-1400. Epub 2024 Dec 30. PMID: 39838987; PMCID: PMC11744119.

Deep Learning for Fundus imaging

Fundus imaging is another imaging modality where deep learning techniques have been applied. Fundus imaging (also known as fundus photography) allows an eye specialist to take a photograph of the fundus region of the eye. This test allows eye specialists to examine a patient's retina, macula, optic nerve, and choroid. These images may be part of a routine eye exam or may be indicated for certain eye diseases or injuries.

Recent developments in applying deep learning to fundus photography allow for early detection of neurodegenerative diseases as well as prediction of age, gender, smoking status, and cardiovascular factors. Furthermore, deep learning models can detect eye diseases such as glaucoma and assign severity scores for age-related macular degeneration (AMD). However, these methods are black boxes and end users would like explainability in their models. To this end, Grad-CAM can be utilized to highlight which fundus features contributed to the model's predictions.

Specifically at Biogen, we are interested in the application of fundus photography to the early detection of diseases such as multiple sclerosis (MS) and Parkinson's disease (PD). This non-invasive test coupled with deep learning provides a quick and low-cost diagnostic tool. FundusNet was developed at Biogen to address these objectives. FundusNet is an ensemble deep learning model consisting of several CNN and vision transformer models. Majority voting of the ensemble was used for classification tasks and averaging of the ensemble was used for prediction tasks. To address potential overfitting issues, techniques such as dropout and image augmentation were applied. Our final model was applied to data from the UK Biobank and achieved a MAE of 2.55 years for age prediction and an AUC of 0.98 for gender classification. FundusNet as applied to neurodegenerative disease diagnosis showed an AUC 0.75 ± 0.03 for Parkinson's disease and an AUC 0.77 ± 0.02 for multiple sclerosis. To aid in explainability, Grad-CAM highlighted image features contributing to disease diagnosis.

References:

1. https://my.clevelandclinic.org/health/diagnostics/fundus-photography
2. https://www.mdpi.com/2306-5354/12/1/57
3. https://www.sciencedirect.com/science/article/abs/pii/S0161642018321857
4. https://www.nature.com/articles/s41551-018-0195-0
5. https://www.sciencedirect.com/science/article/pii/S0161642019318755
6. https://ieeexplore.ieee.org/abstract/document/8359118

Multi Modal Fusion

Furthermore, DL frameworks based on integrating information from multiple imaging modalities and clinical (e.g. EHR)/omics data is nowadays deemed promising for more comprehensive and accurate diagnoses for precision medicine. Multimodal data integration leveraging DL methods such as transformers and graph convolutional networks, can improve the accuracy of cancer detection and diagnosis by combining information from different imaging modalities (e.g., CT scan, MRI, or PET) with genomic or clinical data. The data fusion efforts have been made in various target diseases: cancer diagnosis and prognosis, neurodegenerative disease research, cardiovascular disease management, and infectious disease diagnosis. Integrating multimodal data can help personalize treatment plans and predict patient outcomes by considering various factors, including imaging data, clinical data, and genomic information. Multimodal DL frameworks in healthcare with open-source code, several platforms (a bundle of data, algorithms, and pre-trained foundation models), and tools include MONAI (https://github.com/Project-MONAI/MONAI), OpenMEDLab (https://github.com/openmedlab), and Molmo (https://github.com/allenai/molmo).

To compare fusion techniques in multimodal machine learning (i.e., how modalities are combined for a final prediction, **Table 29):**

	Early Fusion	Intermediate Fusion	Late Fusion
Concept	Multiple input modalities are combined before training a single machine learning model.	Different data modalities are first processed by individual models, before the extracted features are combined and fed into a final prediction model.	Different models are trained on separate data modalities, then the resulting predictions are merged via an aggregation function.
Examples	Integrating features from CT and PET scans with demographic data for lung cancer diagnosis (Hyun et al. 2019). Integrating MRI and PET images with demographic and genetic data for Alzheimer's disease prediction (Dwivedi et al. 2022).	Combining X-ray images with demographic information, biomarkers and clinical measurements in cardiology (Baltruschat et al. 2019). Combining MRI images with demographic information and other clinical and genetic information to predict brain disorders (Spasov et al. 2018).	Diagnosis of cognitive impairment by combining MRI image predictions with demographic data and cognitive assessment scores (Qiu et al. 2018). Merging MRI images with various biomarkers in oncology prediction (Reda et al. 2018). Integrating CT scans with demographic information and clinical measurements for COVID prediction (Zhou et al. 2021).
Advantages	Preserving the original information from each modality without losing substantial details. Simple model architecture and reduced computational complexity.	Capturing more complex interactions between modalities. Allowing separate processing pathways for different data types prior to fusion.	Easily dealing with missing data for patients.

Disadvantages	Potentially an imbalance of data richness from each modality.	Requiring a huge amount of data to precisely learn the additional interactions and combinations.	Impossible to model interactions and relationships between different modalities, with potentially loss of information. Integration of results from different models will be complex.

References:

1. Paverd, H., Zormpas-Petridis, K., Clayton, H. *et al.* Radiology and multi-scale data integration for precision oncology. *npj Precis. Onc.* **8**, 158 (2024). https://doi.org/10.1038/s41698-024-00656-0
2. Mohsen F, Ali H, El Hajj N, Shah Z. Artificial intelligence-based methods for fusion of electronic health records and imaging data. Sci Rep. 2022 Oct 26;12(1):17981. doi: 10.1038/s41598-022-22514-4. PMID: 36289266; PMCID: PMC9605975.
3. Lipkova J, Chen RJ, Chen B, Lu MY, Barbieri M, Shao D, Vaidya AJ, Chen C, Zhuang L, Williamson DFK, Shaban M, Chen TY, Mahmood F. Artificial intelligence for multimodal data integration in oncology. Cancer Cell. 2022 Oct 10;40(10):1095-1110. doi: 10.1016/j.ccell.2022.09.012. PMID: 36220072; PMCID: PMC10655164.
4. https://www.bentoml.com/blog/multimodal-ai-a-guide-to-open-source-vision-language-models
5. S. Dwivedi, T. Goel, M. Tanveer, R. Murugan and R. Sharma, "Multimodal Fusion-Based Deep Learning Network for Effective Diagnosis of Alzheimer's Disease," in *IEEE MultiMedia*, vol. 29, no. 2, pp. 45-55, 1 April-June 2022, doi: 10.1109/MMUL.2022.3156471.
6. Hyun, Seung Hyup MD, PhD∗; Ahn, Mi Sun MD†; Koh, Young Wha MD, PhD‡; Lee, Su Jin MD, PhD§. A Machine-Learning Approach Using PET-Based Radiomics to Predict the Histological Subtypes of Lung Cancer. Clinical Nuclear Medicine 44(12):p 956-960, December 2019. | DOI: 10.1097/RLU.0000000000002810
7. Baltruschat, I.M., Nickisch, H., Grass, M. *et al.* Comparison of Deep Learning Approaches for Multi-Label Chest X-Ray Classification. *Sci Rep* **9**, 6381 (2019). https://doi.org/10.1038/s41598-019-42294-8
8. S. E. Spasov, L. Passamonti, A. Duggento, P. Liò and N. Toschi, "A Multi-modal Convolutional Neural Network Framework for the Prediction of Alzheimer's Disease," *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, HI, USA, 2018, pp. 1271-1274, doi: 10.1109/EMBC.2018.8512468.
9. Shangran Qiu, Gary H. Chang, Marcello Panagia, Deepa M. Gopal, Rhoda Au, Vijaya B. Kolachalama, Fusion of deep learning models of MRI scans, mini–mental state examination, and logical memory test enhances diagnosis of mild cognitive impairment, in: Alzheimer's & Dementia: Diagnosis, Assessment & Disease Monitoring, vol. 10, Elsevier, 2018, pp. 737–749.
10. Jinzhao Zhou, Xingming Zhang, Ziwei Zhu, Xiangyuan Lan, Lunkai Fu, Haoxiang Wang, Hanchun Wen, Cohesive multi-modality feature learning and fusion for COVID-19 patient severity prediction, IEEE Trans. Circuits Syst. Video Technol. 32 (5) (2021) 2535–2549.
11. Islam Reda, Ashraf Khalil, Mohammed Elmogy, Ahmed Abou El-Fetouh, Ahmed Shalaby, Mohamed Abou El-Ghar, Adel Elmaghraby, Mohammed Ghazal, Ayman El-Baz, Deep learning

role in early diagnosis of prostate cancer, Technol. Cancer Res. Treat. 17 (2018) 1533034618775530.

Digital Twins

Digital twins (DTs) can be defined as a virtual representation of a person (an identity) which enables dynamic simulation of potential treatment strategies, monitoring and prediction of health trajectories, and early intervention and prevention, based on multi-scale modeling of multi-modal data such as clinical, genetic, molecular, environmental, and social factors and so on. To classify a system as a DT, it must consist of three components (a physical entity, a virtual replica, and a connection between the two) and be 5I's (individualized, interconnected, interactive, informative, and impactful). There are various applications of DTs for healthcare: hospital management & care coordination, device design, biomarker and drug discovery, bio-manufacturing, surgical planning, clinical trials, personalized medicine, and wellness.

For instance, DT modeling shortens the pharmaceutical processes and makes realistic input-output predictions for biochemical reactions by identifying drug targets that are more likely to succeed. Specifically, some examples of harnessing DL methods to DTs in preclinical drug discovery are scGEN and scGPT at the single-cell level. scGEN used a VAE-based DT model to predict transcriptional response to unseen drugs, drug combinations, and dosages in high-throughput screens. The scGPT model predicted transcriptional responses to genetic perturbation and the source of genetic perturbation for a given transcriptional state based on a generative pretrained transformer.

Harnessing DTs for prognostic modeling has been popular for early detection of failures, optimized maintenance scheduling, cost reduction, and improved decision-making. The benefits of DTs are driven from a comprehensive understanding of disease progression and the customization and optimization of treatment plans via virtually modeling complex interactions between genetic factors and environmental influences.

Within DTs, Digital Human Twins (DHTs) are specialized in simulating human physiology, enabling personalized medicine. The personalized medicine can be planned based on their predicted future health outcomes and estimated personalized "prognostic score" representing their likelihood of experiencing a specific disease progression or event. DHTs can predict an illness more accurately and help physicians make early decisions by simulating the treatment effects on the patients (Hassani et al., Allen et al., Pang et al.).

NVIDIA's DTs, primarily powered by their Omniverse platform, allow users to create highly realistic virtual replicas of physical systems, enabling them to simulate and optimize designs, processes, and operations in real-time, leading to improved efficiency, cost reduction, and better decision-making by providing a safe environment to test scenarios before implementing them in the real world. To use NVIDIA's DTs, mainly through the "NVIDIA Omniverse" platform, users need to create a 3D model of their physical system using existing CAD software or build it directly in Omniverse, integrate real-time data streams from sensors into the model, utilize NVIDIA's simulation tools to run scenarios and analyze behaviors within the DT, and finally leverage AI capabilities to enhance the simulation and make data-driven decisions based on the results. Importantly, building and running complex DTs often require high-performance computing capabilities like NVIDIA RTX GPU's and customization to enhance the functionality of user's DTs are available, depending on user's specific needs. See https://docs.omniverse.nvidia.com/digital-twins/latest/index.html for more details.

Advantages and benefits of digital twins are known by better R&D (enabling more effective research and design of products with abundance of data) and greater efficiency (helping mirror and monitor production systems to achieve peak efficiency). Among many applications and use cases, healthcare services can become more efficient by using digital twins where data are used to track various health indicators and generate key insights.

However, the following challenges may impede the DT developments in healthcare: data acquisition and integration, data privacy and security, data quality and accuracy, data bias and fairness, modeling, computing infrastructure, and business models. To support data privacy and security, data governance must be in place to safeguard the rights of individuals that have DTs, to foster both transparency and fairness in the usage of data. As systems often have diverse data sources and formats, data interoperability standards and protocols must be established to enable seamless data exchange and integration across different systems, devices, and platforms. Lastly, the infrastructure of DT implementation is required while demanding significant computational resources and infrastructure to handle the increasing volume, velocity, and complexity of data.

References

1. Bordukova M, Makarov N, Rodriguez-Esteban R, Schmich F, Menden MP. Generative artificial intelligence empowers digital twins in drug discovery and clinical trials. Expert Opin Drug Discov. 2024 Jan-Jun;19(1):33-42. doi: 10.1080/17460441.2023.2273839. Epub 2024 Jan 8. PMID: 37887266.
2. Lotfollahi, M., Wolf, F.A. & Theis, F.J. scGen predicts single-cell perturbation responses. *Nat Methods* **16**, 715–721 (2019). https://doi.org/10.1038/s41592-019-0494-8
3. Cui, H., Wang, C., Maan, H. *et al.* scGPT: toward building a foundation model for single-cell multi-omics using generative AI. *Nat Methods* **21**, 1470–1480 (2024). https://doi.org/10.1038/s41592-024-02201-0
4. Katsoulakis, E., Wang, Q., Wu, H. *et al.* Digital twins for health: a scoping review. *npj Digit. Med.* **7**, 77 (2024). https://doi.org/10.1038/s41746-024-01073-0
5. Papachristou K, Katsakiori PF, Papadimitroulas P, Strigari L, Kagadis GC. Digital Twins' Advancements and Applications in Healthcare, Towards Precision Medicine. J Pers Med. 2024 Nov 11;14(11):1101. doi: 10.3390/jpm14111101. PMID: 39590593; PMCID: PMC11595921.
6. Hassani, H.; Huang, X.; MacFeely, S. Impactful Digital Twin in the Healthcare Revolution. *Big Data Cogn. Comput.* **2022**, *6*, 83. https://doi.org/10.3390/bdcc6030083
7. Allen, A.; Siefkas, A.; Pellegrini, E.; Burdick, H.; Barnes, G.; Calvert, J.; Mao, Q.; Das, R. A Digital Twins Machine Learning Model for Forecasting Disease Progression in Stroke Patients. *Appl. Sci.* **2021**, *11*, 5576. https://doi.org/10.3390/app11125576
8. Pang, T.Y.; Pelaez Restrepo, J.D.; Cheng, C.-T.; Yasin, A.; Lim, H.; Miletic, M. Developing a Digital Twin and Digital Thread Framework for an 'Industry 4.0' Shipyard. *Appl. Sci.* **2021**, *11*, 1097. https://doi.org/10.3390/app11031097

Summary for Digital Twins (Table 30):

	Advantage	Disadvantage
Digital Twins	• Enables more effective R&D and design of products with abundance of data • Helps monitor production systems to achieve peak efficiency	• Demands significant computational resources to handle the increasing volume, velocity, and complexity of data

Omics Applications

scGPT

Cui et al developed scGPT (https://www.nature.com/articles/s41592-024-02201-0), a foundation deep learning model for single cell tasks, to capture the essence of single cell biology. Their transformer model was trained on over 33 million human cells spanning multiple tissues (brain, blood, lung, etc). In more detail, they stacked transformer layers and applied multi-head attention as their model architecture. This design allowed for generation of both cell and gene embeddings simultaneously. Their model had two training phases. In the first stage, the model was pretrained on scRNAseq data from normal individuals from the cellxgene data collection. Transfer learning was then applied to this model for each downstream task (cell type annotation, prediction of gene perturbation responses, reverse perturbation prediction, multi-batch integration, multi-omic integration, and gene network inference). After completion of the two training phases, the authors claim the final model has state of the art performance on the tasks of multi-omic integration, batch correction, cell type annotation, and genetic perturbation prediction.

Foundation Models for Spatial Transcriptomics

The use of foundation models in spatial transcriptomics (ST) is a rapidly emerging and evolving trend, driven by the increasing availability of large-scale spatial omics data. The trend of using foundation models in spatial transcriptomics reflects a shift towards more sophisticated, data-driven approaches to understanding tissue organization, cell-cell communication, and disease mechanisms in their spatial context. These models are expected to unlock new dimensions of spatial omics and revolutionize biomedical research and clinical applications.

Integration with scRNA-seq

scGPT-spatial and Nicheformer are both foundation models designed to analyze spatial transcriptomics data combined with scRNA-seq, but they differ in their approach to incorporating spatial information and their downstream tasks: scGPT-spatial builds upon the existing scGPT model by incorporating spatial context through continual pretraining on a large spatial transcriptomics dataset while Nicheformer is specifically designed to integrate both single-cell and ST data and was pre-trained on a massive dataset of over 110M cells.

scGPT-spatial offers a comprehensive and powerful framework for ST data analysis. It can effectively integrate data from diverse ST technologies, including both imaging-based (e.g., MERFISH, Xenium) and sequencing-based protocols (e.g., Visium, Visium HD). Its specialized training strategy by grouping nearby cells/spots into "patches" allows it to learn and capture complex spatial co-localization patterns of cell states and microenvironments within tissue sections. The model extensively pre-trained on a massive and diverse dataset, consisting of 30M human cells/spots across multiple organs, tissues, and disease conditions, contributes to the model's ability to generalize effectively across different datasets and biological contexts. For downstream applications, scGPT-spatial demonstrates superior performance in various tasks: spatial domain clustering, multi-modal and multi-slides integration, cell-type deconvolution, gene expression imputation, and zero-shot integration.

Nicheformer's transformer architecture and attention mechanisms excel at capturing the non-linear spatial relationships and long-range dependencies between cells, providing a richer understanding of tissue organization. Nicheformer can leverage the rich spatial information learned from ST data to annotate dissociated scRNA-seq datasets, even in the absence of explicit spatial information in the latter. This capability facilitates the integration of different SC data types and helps researchers analyze existing scRNA-seq datasets from a spatial perspective. Nicheformer's superior performance shines in a variety of downstream tasks, such as predicting cell type, niche, and region labels, as well as predicting neighborhood cell composition and density. Leveraging spatial patterns learned, specific regions or cell-types for future ST experiments can be prioritized, while maximizing the potential for novel biological insights.

Table 31: Comparison of scGPT-spatial vs Nicheformer

scGPT-spatial	Nicheformer
• Extends the existing scGPT model by fine-tuning it on the SpatialHuman30M dataset, which contains 30M ST. • Uses a novel sampling strategy that groups nearby cells into "patches" and employs a neighborhood-based imputation objective to capture cell-cell interactions. • Utilizes a MoE (Mixture of Experts) architecture in its decoders to handle different data formats and training methods. • Avoids explicitly encoding spatial coordinates, which allows it to generalize across different datasets and technologies. • Handles various data formats and training strategies, making it versatile for different ST technologies. • GitHub - bowang-lab/scGPT-spatial	• Pretrained on over 110M cells from both dissociated and ST datasets. • Directly integrates SC and ST data. • Enables the prediction of the spatial context of dissociated cells, allowing transfer of spatial information to traditional scRNA-seq datasets. • Introduces novel spatial tasks like spatial density prediction and niche/region label prediction. • The transformer architecture is designed to capture complex spatial relationships. • GitHub - theislab/nicheformer: Repository for Nicheformer: a foundation model for single-cell and spatial omics

Integration with pathology images

STPath and OmiCLIP are two recently developed AI-powered tools aiming to integrate visual histology data with ST, offering a more complete understanding of tissue structure and gene expression patterns in health and disease.

STPath is a recently proposed foundation model for integrating ST data, especially with Whole Slide Images (WSIs). STPath offers various advantages, including its capability to integrate various data modalities like histological images and gene expression. It can also directly predict gene expression for numerous genes and organs without the need for additional fine-tuning. The model benefits from large-scale pretraining and a novel geometry-aware Transformer architecture that captures spatial relationships. Moreover, STPath utilizes a unique masked gene expression prediction objective to balance gene-gene dependencies and prediction quality. It has demonstrated versatility across various tasks and shows promise for scalable applications.

OmiCLIP effectively connects gene expression profiles (derived from RNA sequencing) with corresponding histological images (H&E stained), allowing for a more comprehensive understanding of tissue characteristics and associated biological processes. This integration provides a richer context for interpreting gene function within the physical structure of tissues. OmiCLIP helps scientists and pathologists gain a more complete picture of how genes operate across different tissue regions and how these patterns relate to disease processes, by linking visual tissue features to gene activity. This has the potential to improve diagnostics, prognostics, and treatment strategies. OmiCLIP can potentially reduce the reliance on extensive laboratory testing by extracting rich gene expression data directly from standard pathology images, thereby streamlining complex analyses. OmiCLIP was trained on a massive dataset of 2.2M tissue images and gene expression pairs from 32 different organs, enabling it to learn robust associations between visual and transcriptomic data. A key strength lies in transforming transcriptomic data into "sentences" by listing the most highly expressed genes in each tissue patch, making it easier for the AI to learn the link between visual features and gene activity.

Table 32: STPath vs OmiCLIP

STPath	OmiCLIP
• A generative foundation model for integrating whole-slide images (WSIs) with ST. • Pretrained on a large collection of WSIs with paired ST data. • Utilizes a geometry-aware Transformer architecture to integrate various data modalities including histological images, gene expressions, organ type, and sequencing technology information. • Predicts the expression levels of 38,984 genes from histology images without further fine-tuning. • Used for gene expression prediction, spot imputation, spatial clustering, gene mutation prediction, and survival prediction. • Enables accurate gene expression prediction and reveals important pathological structures within tissue samples. • https://github.com/Graph-and-Geometric-Learning/STPath	• A visual-omics foundation model designed to bridge omics data and hematoxylin and eosin (H&E) images. • Trained on a dataset of 2.2M tissue images and gene expression pairs from 32 different organs. • Connects gene expression data (RNA sequencing) with histology images (H&E stained). • Learns the relationship between tissue image features and underlying gene activity by converting transcriptomic data into "sentences" listing highly expressed genes in each tissue patch. • Built upon OmiCLIP, the Loki platform offers five key functions: tissue alignment, annotation, cell-type decomposition, histology image-transcriptomics retrieval, and spatial gene expression prediction from H&E images. • Provides a more complete picture of gene usage across different regions of tissue and its relation to disease processes. • https://github.com/GuangyuWangLab2021/Loki (Pretrained OmiCLIP weights are available via HuggingFace).

In summary, STPath focuses on the scalable generation of ST from WSIs, providing broad gene coverage and requiring no fine-tuning. However, OmiCLIP offers a more nuanced understanding of gene function within the tissue context and provides a framework for advanced analyses through the Loki platform.

References

1. Wang, Chloe Xueqi and Cui, Haotian and Zhang, Andrew Hanzhuo and Xie, Ronald and Goodarzi, Hani and Wang, Bo. scGPT-spatial: Continual Pretraining of Single-Cell Foundation Model for Spatial Transcriptomics. bioRxiv (2025).
2. Schaar, A.C., Tejada-Lapuerta, A., et al. Nicheformer: a foundation model for single-cell and spatial omics. bioRxiv (2024). doi: https://doi.org/10.1101/2024.04.15.589472
3. Huang, Tinglin and Liu, Tianyu and Babadi, Mehrtash and Ying, Rex and Jin, Wengong. STPath: A Generative Foundation Model for Integrating Spatial Transcriptomics and Whole Slide Images. bioRxiv (2025).
4. Chen, W., Zhang, P., Tran, T.N. *et al.* A visual–omics foundation model to bridge histopathology with spatial transcriptomics. *Nat Methods* **22**, 1568–1582 (2025). https://doi.org/10.1038/s41592-025-02707-1.

Chapter 14: Introduction to LLMs

LLMs have transformed the landscape of NLP by enabling sophisticated text generation capabilities across diverse domains. From chatbots and virtual assistants to content creation and scientific research, LLMs generate human-like text by predicting the most probable next words based on vast amounts of training data. More recently, multimodal LLMs can understand and generate text, audio, video, and images. While these models demonstrate remarkable fluency and versatility, their deployment requires careful consideration of multiple factors, including latency, cost, privacy, toxicity, tone, bias, security, and efficiency. We will explore each of these factors in general considerations as they play a crucial role in ensuring that LLMs operate responsibly, produce reliable outputs, and align with ethical standards.

General Considerations

Latency and Cost

Deploying LLMs in real-time applications requires balancing response speed (latency) and computational expenses (cost). Latency is critical for user-facing applications such as chatbots, search engines, and voice assistants, where delays can degrade the user experience. Metrics like "time to first token (TFT)" and "output tokens per second (OTPS)" measure how quickly a model starts responding and sustains generation speed.

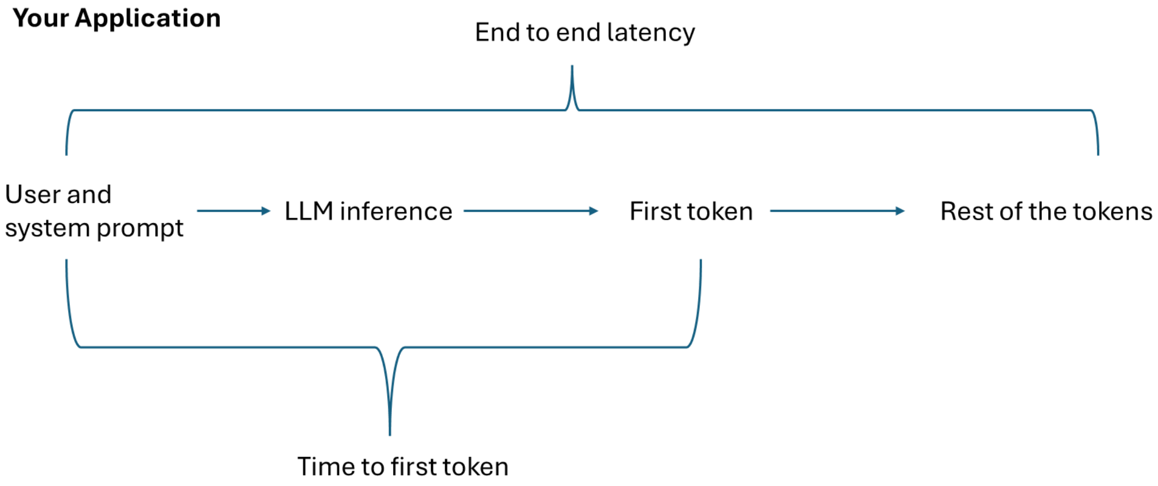


Figure 26: LLM latency illustration

Several factors influence latency, including:

- Prompt and response length – Longer inputs and outputs increase processing time.
- Model size – Larger models require more computation per token.
- Hardware efficiency – GPU/TPU memory limits and bandwidth constraints can create bottlenecks.

- Retrieval complexity – Architectures like RAG introduce additional latency from document search and embedding computations before token generation begins.

To mitigate latency while maintaining performance, optimization techniques include:

- Model quantization – Reduces memory usage and speeds up inference by lowering numerical precision (e.g., 8-bit or 4-bit).
- Caching – Stores frequent computations (e.g., precomputed embeddings, partial sequences, prompt caching) to avoid redundant processing.
- Parallel processing – Runs token generation and retrieval tasks concurrently across multiple GPUs or TPUs.
- Speculative decoding – Predicts multiple possible next tokens at once, improving efficiency.
- Model distillation – Trains a smaller, faster model to approximate the performance of a larger model

Cost is another key constraint, particularly for large-scale or cloud-based deployments where inference costs scale with token usage. Both training and fine-tuning require extensive computational resources, contributing to high energy consumption and environmental impact. Additionally, API-based models charge for both input and output tokens, making cost management crucial.

Several strategies help optimize cost and efficiency:

- Prompt engineering – Using concise, well-structured prompts reduces token consumption while improving task efficiency.
- Smaller or distilled models – Fine-tuned smaller models can handle simpler queries efficiently, reducing costs.
- Adaptive inference – Dynamically selecting model sizes based on query complexity optimizes resource allocation.
- Token management techniques – Batching, chunking, token recycling, caching, and compression reduce unnecessary token usage.
- Automated monitoring – Continuous tracking of performance, cost, and latency helps fine-tune deployment strategies.

As a real-world example, Amazon Bedrock's Intelligent Prompt Routing automatically selects the most cost-effective and performant model based on request complexity, preventing developers from committing to a single model for all queries. Such automated model selection approaches enable better trade-offs between speed, cost, and quality in real-world applications.

References:

1. https://www.proxet.com/blog/llm-has-a-performance-problem-inherent-to-its-architecture-latency
2. https://medium.com/@bijit211987/prompt-optimization-reduce-llm-costs-and-latency-a4c4ad52fb59
3. https://aws.amazon.com/blogs/machine-learning/optimizing-ai-responsiveness-a-practical-guide-to-amazon-bedrock-latency-optimized-inference

Privacy

Privacy is a critical concern in LLM response generation to prevent improper storage, sharing, or processing of user data. Privacy risks stem from two main sources during generation: the training data used to develop LLMs and the user-provided prompts during interactions. Unlike conventional systems where data can be explicitly deleted, LLMs encode information within their parameters, making it nearly impossible to isolate or remove specific data. This limitation clashes with privacy regulations like General Data Protection Regulation (GDPR), which emphasize the "right to be forgotten." Sensitive data in training sets may persist throughout a model's lifecycle, raising compliance and security challenges.

To enhance privacy, keeping sensitive data localized is essential. Federated learning trains models collaboratively across multiple devices without sharing raw data. Each device trains a local model with its data and sends only updates (e.g., gradients) to a central server for aggregation, preserving privacy. For instance, a smartphone keyboard app can improve its autocomplete suggestions by training on individual typing patterns without uploading the user's messages to a central server. Similarly, on-device computation processes tasks like inference and training directly on user devices, ensuring sensitive data never leaves the device. This approach relies on optimized, lightweight models to balance performance with resource constraints while protecting user privacy. For example, voice assistants like Apple's Siri or Google Assistant increasingly process voice commands locally on the device for improved privacy and responsiveness.

To further safeguard sensitive information, data protection mechanisms can be implemented, such as encryption (both in transit and at rest), access control using role-based access control (RBAC) and multi-factor authentication (MFA), and data anonymization techniques like tokenization or pseudonymization during processing. These measures help prevent unauthorized access, reduce risks associated with data breaches, and ensure compliance with privacy standards.

References:

1. https://www.skyflow.com/post/private-llms-data-protection-potential-and-limitations
2. https://gdpr.eu/
3. Hua, S., Yang, K., & Shi, Y. (2019, September). On-device federated learning via second-order optimization with over-the-air computation. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)* (pp. 1-5). IEEE.

Toxicity

Toxicity in LLM-generated responses refers to harmful, offensive, or inappropriate content that could hurt users or spread misinformation. This includes hate speech, biased statements, and any language that targets individuals or groups based on sensitive characteristics such as gender, race, ethnicity, religion, or sexual orientation. Toxicity can have severe consequences, from perpetuating stereotypes and misinformation to causing emotional distress and hindering inclusive communication. Mitigating toxicity involves several strategies:

- Fine-Tuning Models: Training on curated datasets designed to reduce biases and harmful language.
- Toxicity Filters and Cleaning: Implementing automated systems to detect and remove inappropriate content.
- Adversarial Testing: Using prompts designed to provoke toxic responses to evaluate and improve model robustness.
- Human Oversight: Monitoring outputs and intervening when necessary to address subtle or context-dependent toxicity.

Two examples of toxicity mitigation strategies are:

1. Reinforcement Learning with Human Feedback (RLHF) is a proactive approach that reduces the likelihood of harmful outputs by aligning model responses with human ethical standards during training.
2. Post-generation safety layers complement this by acting as a reactive safeguard, identifying and removing harmful language in real time (e.g., hate speech or offensive content).

Lastly, transparency about the language model’s training data, algorithms, and decision-making processes is crucial. Clear communication about these aspects helps users understand the model's limitations, fosters trust, and enables accountability for its outputs.

References:

1. https://www.hyro.ai/blog/mitigating-toxicity-large-language-models-llms/#:~:text=for%20mitigating%20it.-,What%20Is%20LLM%20Toxicity?,distress%20and%20hindering%20inclusive%20communication
2. https://medium.com/@aisagescribe/a-comprehensive-guide-to-understand-bias-toxicity-in-llms-bae01b8e58a7
3. https://www.hyro.ai/blog/mitigating-toxicity-large-language-models-llms/

Tone

The tone of LLM-generated responses is the overall attitude or feeling conveyed through the language and style used in the output. Tone impacts clarity, trust, user engagement, and overall user satisfaction, playing a crucial role in the user experience. Effective models must dynamically adapt their tone to suit the context, such as distinguishing between formal and casual settings. For instance, a reassuring tone is common in healthcare interactions to alleviate user anxiety, while a formal tone is more appropriate for official documentation or professional communications.

Models should tailor their tone based on factors such as user intent, query type, prompt instruction, domain, and the desired outcome of the interaction. For ambiguous or unclear contexts, adopting a neutral, polite tone minimizes the risk of miscommunication or unintended offense. Additionally, integrating user preferences, cultural nuances, and emotional sensitivity into tone adjustments can lead to more personalized and contextually appropriate interactions, enhancing the overall experience. Failure to align the tone with user expectations or situational demands can result in misunderstandings, reduced trust, or dissatisfaction. Therefore, tone alignment is not just a matter of improving user experience but also a key consideration for ethical, inclusive, and effective AI design.

References:

1. https://tokenomics.ie/topics/the-tone-of-the-prompt
2. https://originality.ai/blog/study-popular-llms-make-content-neutral-sentiment
3. https://cacm.acm.org/research/the-science-of-detecting-llm-generated-text

Bias

LLMs could exhibit biases in many dimensions, including gender, racial and ethnicity, age, socioeconomic status, geography, language, politics and ideology, recency (i.e., more weight to recent inputs while overlooking earlier context), and gaps in knowledge due to training data limitations (e.g., lacking awareness of events after the training cutoff date). These biases arise from various factors, including training data, model architecture, and fine-tuning processes. Human and societal biases are often unintentionally embedded during data collection, causing models to adopt undesirable behaviors or reflect societal stereotypes and disparities. These biases often lead to skewed or unfair responses in the generated outputs. Despite challenges such as limited diversity in data sources and inherent imperfections in models, several benchmarks—such as BBQ, BOLD, and JobFair—have been developed to identify and assess biases in LLMs. For example, BBQ is a dataset designed to assess how NLP models exhibit social biases in question-answering tasks. It evaluates models in both under-informative and adequately informative contexts using metrics such as accuracy and bias scores to systematically quantify biased responses, showing that models often rely on stereotypes when uncertain and tend to select answers that align with biases even when sufficient context is provided.

For instance, one evaluation compares the semantic distance of sentence completions containing male and female tokens (e.g., he/him/his vs. she/her/hers) to calculate the proportion of responses inclined toward one gender across various occupations. Studies using models like BERT and GPT-2 revealed that sentences related to males were disproportionately associated with science and technology roles, whereas sentences about females were more commonly linked to healthcare and medicine. A similar trend was observed in the original Wikipedia data used for training, highlighting how biases present in source material are inherited by language models.

Addressing bias in LLMs is an ongoing and complex challenge because these biases reflect the realities of human society. Mitigating bias requires curating diverse and representative datasets, applying debiasing techniques, and conducting regular audits to identify and rectify discriminatory patterns. One approach, adversarial training, involves using an adversary that predicts sensitive attributes (e.g., gender or race) from model outputs. The model is then trained to minimize the adversary's accuracy, reducing its reliance on these attributes. Data augmentation, another debiasing approach, expands training data with more diverse examples, exposing the model to a broader range of perspectives. Additionally, prompts can be designed to explicitly instruct the model to produce unbiased outputs and post-processing filters can remove potentially biased content from generated text based on predefined criteria.

In a nutshell, gaining a deeper understanding of the extent and causes of bias is essential before implementing solutions. Transparency in data sources and decision-making processes is essential for fostering trust and accountability in LLM systems.

References:

1. https://medium.com/@aisagescribe/a-comprehensive-guide-to-understand-bias-toxicity-in-llms-bae01b8e58a7#
2. https://www.holisticai.com/blog/assessing-biases-in-llms
3. https://www.linkedin.com/pulse/llm-implementations-types-biases-mitigation-prabhanjan-pandurangi-bpd5c
4. Parrish, A., Chen, A., Nangia, N., Padmakumar, V., Phang, J., Thompson, J., ... & Bowman, S. R. (2021). BBQ: A hand-built bias benchmark for question answering. arXiv preprint arXiv:2110.08193. https://aclanthology.org/2022.findings-acl.165.pdf

Security

Like any data source or platform, generative AI must be deliberately safeguarded against security threats, including unauthorized access, adversarial attacks, data breaches, and confidential data exposure to public facing chatbots. Without robust security measures, sensitive information—such as patient or customer data and proprietary business information—could be exposed, leading to privacy violations, financial losses, and reputational harm. Compromised systems could return incorrect medical information, resulting in erroneous diagnoses or treatment plans that might endanger patients' lives. Furthermore, malicious actors can manipulate AI prompts or modify outputs before delivery to end users, potentially spreading misleading or toxic information. Failure to comply with data protection laws and regulations—such as the Health Insurance Portability and Accountability Act (HIPAA), General Data Protection Regulation (GDPR), and California Consumer Privacy Act (CCPA)—can result in substantial fines, legal penalties, and loss of business, trust, and reputation.

A multi-layered approach is crucial for addressing data protection, model integrity, and compliance. In addition to the data protection measures discussed in the Privacy section, organizations can implement several other effective strategies. One important approach is **regular penetration testing**, where ethical hackers simulate attacks on the system to identify vulnerabilities before malicious actors can exploit them. Another approach is **secure model deployment**, which ensures that AI models are hosted in secure environments with restricted access, preventing unauthorized modifications or tampering. For instance, Google's AI models are hosted on the Google Cloud Platform (GCP), which employs a combination of secure virtual machines, encrypted storage, and access controls like **Identity and Access Management (IAM).** The secure hosting of Microsoft Azure is well-suited for enterprise environments and has a broader global infrastructure than the cloud-based GCP. Finally, **logging and monitoring** are essential for detecting suspicious activity or potential breaches in real-time. By continuously monitoring system performance and user interactions, organizations can quickly identify and respond to security threats, ensuring that generative AI systems remain protected against evolving risks.

References:

1. https://www.immuta.com/guides/data-security-101/retrieval-augmented-generation-rag
2. https://cloud.google.com/security/products/iam
3. https://cloud.google.com/iam/docs/overview

Other Considerations

Other considerations of LLM response generation may include scalability, interpretability, and energy efficiency. Addressing these considerations is also vital for the responsible and effective integration of LLM technologies across various industries.

Scalability

Scalability is the ability of a model to handle growing data volumes and user interactions without losing performance. It encompasses administrative scalability (across organizations), geographic scalability (across distances), and load scalability (computational capacity). Scalable systems must process diverse, high-volume queries efficiently. Key strategies include resource optimization (e.g., model quantization, model distillation, distributed computing), data management (e.g., compression, sharding), and efficient model training architectures (e.g., transfer learning, few-shot learning). Scalable algorithms like distributed machine learning process large datasets efficiently while addressing ethical concerns through bias detection, privacy-preserving technologies, and AI ethics guidelines. These approaches are vital for organizations to maintain consistent performance and meet increasing demands. For instance, Facebook's image recognition system integrates deep learning, distributed computing, smart data management, and privacy measures to handle vast visual data, enhancing user experience and content moderation.

Interpretability

Interpretability involves making the models' decision-making processes transparent and understandable to users. This transparency fosters trust and accountability by helping users comprehend why certain responses are generated. Key approaches to improving interpretability include visualizing data and model behavior, decomposition techniques that break larger models into smaller, more understandable components, and post-hoc methods such as feature attribution (identifying which input features most influenced the output) or attention mapping (highlighting regions of input that the model focused on). For example, in dialogue systems, developers can implement tools to provide users with explanations about how the system generates responses, such as highlighting key input factors or summarizing the rationale for an output. In high-stakes sectors like healthcare and finance, interpretability is important. For instance, generative AI used in clinical decision support should explain treatment recommendations to ensure they align with medical guidelines. The FDA underscores the importance of explainability in AI models to ensure models are accurate, fair, and transparent, adhering to regulatory and ethical standards. **[See chapter 11 for more information]**

Energy Efficiency

Training and deploying LLMs demand substantial computational resources, leading to high energy consumption and environmental impact. For example, training a single BERT model from scratch can produce a carbon footprint equivalent to a commercial trans-Atlantic flight. To make AI more sustainable, it is suggested to prioritize reusing existing models over training new ones from scratch and reserving the use of a large model for cases where they deliver substantial value. Optimizing energy usage while maintaining performance is essential for sustainable AI deployment. Techniques such as batching and parallelism can enhance throughput and scale deployment across applications, improving energy efficiency. Model distillation also helps by reducing model size and computational demands while maintaining performance. Additionally, innovations like TinyML (efficient machine learning for edge devices) and region-specific model deployments in areas with environmentally friendly energy sources (e.g., renewable energy regions like Quebec compared to the U.S.) are being explored to reduce carbon footprints. It is crucial to factor in the carbon footprint when selecting and using AI models. The developer community can play a key role by reporting performance metrics on energy usage, fostering transparency, and establishing benchmarks. Tools like CodeCarbon, Green Algorithms, and ML CO2 Impact are available to estimate and monitor the energy impact of AI algorithms, encouraging more environmentally conscious practices in the AI ecosystem.

References:

1. https://www.researchgate.net/publication/375370072_SCALABILITY_IN_ARTIFICIAL_INTELLIGENCE
2. https://www.ibm.com/think/topics/interpretability
3. https://www.xcally.com/news/interpretability-vs-explainability-understanding-the-importance-in-artificial-intelligence/
4. https://www.fda.gov/media/135748/download
5. Stojkovic, J., Choukse, E., Zhang, C., Goiri, I., & Torrellas, J. (2024). Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. arXiv preprint arXiv:2403.20306.
6. https://hbr.org/2023/07/how-to-make-generative-ai-greener

LLM Frameworks

LangChain

LangChain (https://www.langchain.com/) is a framework that facilitates the development of complex applications powered by language models. It allows developers to chain together multiple LLMs and tools to perform intricate tasks that require a combination of capabilities, such as understanding, reasoning, and generating language.

LangChain provides a modular architecture where each module represents a specific function or task. These modules can be combined in a pipeline or graph structure to process inputs and produce desired outputs. The framework supports prompt engineering, enabling fine-tuning of model interactions and improving performance on specific tasks.

In a pharmaceutical context, LangChain can be used to create an application that first extracts information from scientific literature, then summarizes findings, and finally generates hypotheses for new research directions.

Table 33: Pros and Cons of LangChain:

Pros	Cons
• **Modularity:** Enables the reuse of components across different applications. • **Scalability:** Supports scaling up to handle larger datasets and more complex tasks. • **Flexibility:** Allows for customization and integration with various LLMs and external tools.	• **Increased Latency:** Multiple model calls can lead to slower response times. • **Complex Debugging:** The interconnections between modules can make it challenging to identify and fix issues.

LangGraph

LangGraph (https://www.langchain.com/langgraph) extends the capabilities of LangChain by incorporating graph-based structures to represent and manage the relationships between tasks, data, and models. This approach enhances the ability to handle complex dependencies and interactions within language model applications.

LangGraph utilizes graph theory principles to model the flow of data and control between different components. Nodes represent tasks or data entities, while edges define relationships or dependencies. This structure allows for parallel processing and optimization of resource allocation.

In pharmaceutical research, LangGraph can manage complex workflows such as multi-omics data analysis, where different types of biological data need to be integrated and interpreted cohesively (see https://www.nature.com/articles/s41467-025-57430-4).

Table 34: Pros and Cons of LangGraph

Pros	**Cons**
• **Enhanced Data Management:** Improves the organization and accessibility of data and model interactions. • **Visualization:** Provides clear visual representations of processes, aiding in understanding and communication. • **Optimization:** Facilitates the identification of bottlenecks and opportunities for performance improvements.	• **Learning Curve:** Requires familiarity with graph-based concepts and tools. • **Setup Complexity:** Initial configuration can be time-consuming and may require specialized expertise.

LlamaIndex

LlamaIndex (https://www.llamaindex.ai/), formerly known as GPT Index, is a data framework that connects LLMs with external data sources. It allows for the indexing and efficient querying of large datasets, enhancing the ability of language models to access and utilize external information.

LlamaIndex constructs indices over datasets using embeddings and vector databases. It supports semantic search, traditional keyword-based retrieval methods, as well as hybrid search options (https://docs.llamaindex.ai/en/stable/examples/vector_stores/qdrant_hybrid/). The framework provides interfaces for LLMs to interact with these indices, enabling them to retrieve relevant information dynamically during inference.

For example, in drug discovery, LlamaIndex can index a database of chemical compounds and enable an LLM to retrieve information about compounds with specific properties or activities. (https://milvus.io/ai-quick-reference/can-llamaindex-be-used-for-entity-extraction-tasks?utm_source=chatgpt.com)

Table 35: Pros and Cons of LlamaIndex:

Pros	Cons
• **Efficient Data Retrieval:** Optimizes access to large datasets, reducing latency and improving performance. • **Scalability:** Capable of handling datasets ranging from small collections to massive repositories. • **Enhancement of LLM Capabilities:** Extends the knowledge and functionality of LLMs beyond their training data.	• **Index Maintenance:** Requires regular updates to indices to reflect new data or changes. • **Data Privacy Concerns:** Handling sensitive data necessitates robust security measures to ensure compliance with regulations.

Semantic Kernel

Semantic Kernel by Microsoft (https://learn.microsoft.com/en-us/semantic-kernel/overview/) is an open-source framework designed to integrate LLMs with traditional software development paradigms. It allows developers to create "semantic functions"—reusable components that combine natural language understanding and generation tasks with conventional code. This approach offers strong extensibility and can be integrated into broader enterprise systems, making it particularly appealing in regulated industries like pharmaceuticals.

Semantic Kernel also supports orchestrating prompts, chaining operations, and maintaining application-specific state. Its plugin-like architecture facilitates calling external services or domain-specific APIs, enabling a pharmaceutical team to blend LLM reasoning with, for example, chemical property calculators or EHR data retrieval services.

Pharmaceutical Use Case Example:
A clinical decision support system could use Semantic Kernel to interpret complex patient data, apply reasoning via an LLM to recommend treatment options, and then query a pharmacovigilance database to ensure no dangerous drug interactions exist.

Table 36: Pros and Cons of Semantic Kernel:

Pros	Cons
- **Enterprise Integration:** Designed to integrate LLM capabilities into existing codebases and workflows. - **Plugin Architecture:** Encourages easy addition of new tools and data sources. - **Robustness:** Maintained and supported by a major technology vendor, suggesting long-term reliability.	- **Code Complexity:** Requires software engineering knowledge to harness fully. - **Limited Maturity for Specialized Domains:** May need customization for highly specialized pharmaceutical tasks.

Guidance

Guidance (https://docs.llamaindex.ai/en/stable/community/integrations/guidance/) is a prompt orchestration framework that simplifies the process of writing, iterating, and maintaining complex prompt strategies for LLMs. It enables developers to create templates, conditionally execute steps, and chain together multiple prompt calls. This is particularly valuable in tasks where careful prompt design is crucial—such as translating regulatory guidelines into actionable research steps or guiding LLMs to extract specific types of biomedical information.

Pharmaceutical Use Case Example:
Guidance can help a pharmaceutical team craft a multi-step prompt workflow for analyzing a drug's mechanism of action. The workflow may follow ideas from Markhasin (https://arxiv.org/pdf/2505.03332) whereby the framework would extract key molecular interactions from the literature, interpret patient data, and then summarize the findings in an executive report—all orchestrated through well-structured, version-controlled prompt templates.

Table 37: Pros and Cons of Guidance

Pros	Cons
- **Prompt Engineering Control:** Facilitates consistent, high-quality prompts and reproducible results. - **Iterative Refinement:** Simplifies debugging and refining prompt strategies over time. - **Transparency:** Offers clear separation between prompt logic and model responses.	- **Limited Scope:** Focused primarily on prompt orchestration rather than full-stack application building. - **Requires Prompt Engineering Expertise:** Users need knowledge of effective prompt design patterns.

Chainlit

Chainlit is an open-source Python package to build one's own Conversational AI applications (e.g. AI chatbots) and further develop user-friendly interfaces by leveraging the power of ChatGPT-like widgets and Langchain methodologies. A Chainlit application can be built quickly by integrating seamlessly with an existing code base or starting from scratch in minutes. A Chainlit application can be consumed through multiple platforms: if user's assistant logic is written up once, with the Chainlit application, many platforms are available to be incorporated, such as Web App, Copilot, API, Custom React App, Teams, Slack, or Discord.

By default, a Chainlit application does not persist the chats and elements it generates. However, once enabled, data persistence will introduce new features such as chat history and human feedback, to the application. To enable data persistence in the Chainlit app, two options are available: custom data layer and Literal AI. After a user's custom data layer is imported in a chainlit app, then it should be assigned to a data layer variable.

Literal AI offers the simplest way to persist, analyze, and monitor data. Literal AI is a data persistence solution, allowing one to quickly enable data storage and analysis for a Chainlit app without setting up customized infrastructure.

Lastly, Chainlit apps can visualize multi-step reasoning to help the user understand the intermediate steps that produce an output. There are two ways to create steps, either by using the @cl.step decorator or by using the cl.step class.

The improvement of Chainlit over Streamlit is to demonstrate the intermediate steps and the model logics, allowing users to understand the thought process until the end of outputs by debugging the language model decision. Moreover, the "Sources" in Chatbot responses contains the reference IDs that navigate to the cited context from the referred original documents. Compared to Streamlit, Chainlit demands fewer lines of coding but utilizes richer chatbot functionalities.

REFERENCES

1. https://docs.chainlit.io/get-started/overview
2. https://medium.com/gitconnected/for-chatbot-development-streamlit-is-good-but-chainlit-is-better-4112f9473a69

General-Purpose vs. Medical/Clinical Large Language Models

General-Purpose Large Language Models

General-purpose LLMs, such as GPT-4, are trained on diverse datasets containing vast amounts of internet text. They excel in understanding and generating human-like language across a wide range of topics. Their capabilities include language translation, summarization, question-answering, and creative content generation.

Limitations

- **Lack of Domain-Specific Knowledge:** While versatile, they may not possess deep expertise in specialized fields like medicine or pharmacology.

- **Potential for Inaccuracies:** Without domain-specific training, they can produce incorrect or misleading information in technical contexts.
- **Compliance and Ethical Concerns:** May generate content that does not adhere to medical guidelines or ethical standards.
- **Hallucinations:** LLMs may produce incorrect information based on biased or low-quality training data; some hallucinations are caused by the probabilistic nature of next word predictions
- **Cutoff date:** Many LLMs have a training cutoff date, so the LLMs may not have knowledge of current events
- **Explainability**: Due to the black box nature of LLMs, it may be difficult to understand their responses
- **Bad actors:** bad actors may use LLMs to generate phishing messages

Medical/Clinical Large Language Models

Medical or clinical LLMs are fine-tuned on domain-specific corpora, including biomedical literature, clinical notes, and healthcare guidelines. Examples include BioBERT, PubMedGPT, and ClinicalBERT. These models are adept at understanding medical terminology, interpreting clinical data, and assisting with tasks such as diagnosis support and literature summarization.

The development of medical LLMs involves:

- **Domain-Specific Pre-Training:** Starting with general language models and further training on medical texts to capture specialized knowledge.
- **Fine-Tuning:** Adjusting the model on specific tasks or datasets, such as EHRs or clinical trial data.
- **Evaluation and Validation:** Rigorous testing against benchmarks and clinical cases to ensure accuracy and reliability.

Limitations:

- **Data Limitations:** Access to high-quality, annotated medical data can be restricted due to privacy concerns.
- **Overfitting Risks:** Models may become too specialized, reducing their ability to generalize to new or unexpected cases.
- **Regulatory Compliance:** Must adhere to strict regulations regarding patient data and medical advice dissemination.

References:

1. Gu, Y., et al. (2022). **Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing**. *ACM Transactions on Computing for Healthcare*, 3(1), 1–23.

Comparison of General Purpose LLMs and Specialized LLMs

Combining general purpose and medical LLMs can leverage the strengths of both. A hybrid approach allows for general linguistic capabilities while incorporating specialized medical knowledge. This can be achieved through model ensembles or by designing systems where tasks are routed to the most appropriate model.

Table 38: Comparison of General Purpose LLMs and Medical LLMs

General-Purpose LLMs	Medical/Clinical LLMs
• **Pros:** Broad knowledge base, adaptable to various tasks, and readily available. • **Cons:** Lack depth in specialized fields and potential for misinformation in critical areas.	• **Pros:** High accuracy in domain-specific tasks; understands medical terminology and context; and valuable for specialized applications. • **Cons:** Limited to medical content; may require more resources for training and validation; and potential ethical and legal considerations.

Ensuring compliance with regulations like HIPAA and GDPR is paramount. Strategies include:

- **Anonymization:** Removing personally identifiable information from datasets.
- **Secure Infrastructure:** Utilizing encrypted data storage and secure communication protocols.
- **Access Controls:** Implementing strict authentication and authorization mechanisms.

Models must undergo rigorous testing to validate their performance and ensure they meet regulatory standards. This includes:

- **Benchmarking:** Comparing model outputs against established standards or expert opinions.
- **Continuous Monitoring:** Implementing systems to detect and correct errors or drifts in model performance over time [**See chapter 17 for more information**].
- **Ethical Oversight:** Establishing committees or protocols to review and approve AI applications in sensitive areas.

References:

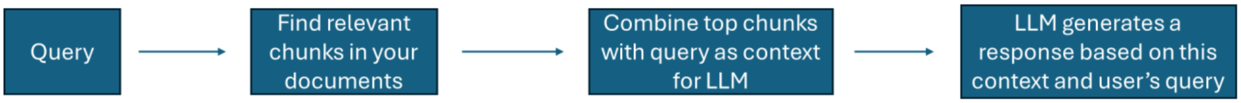
1. Lehman, E., Hernandez, E., Mahajan, D., Wulff, J., Smith, M. J., Ziegler, Z., Nadler, D., Szolovits, P., & Alsentzer, E. (2023). Do We Still Need Clinical Language Models? arXiv. Demonstrates that small clinical models outperform general LLMs on clinical tasks
2. Dorfner, F. J., Dada, A., Busch, F., Makowski, M. R., Han, T., Truhn, D., … & Sushil, M. (2024). Biomedical Large Language Models Seem Not to Be Superior to Generalist Models on Unseen Medical Data. arXiv. Reveals general-purpose LLMs sometimes outperform biomedical-tuned models

Chapter 15: Introduction to RAG (Retrieval Augmented Generation)

RAG, or retrieval augmented generation, is a popular pipeline to address some of the limitations of LLMs. These limitations include: 1) their training set is cut off at a specified date so the LLM does not have access to current information, 2) LLMs can hallucinate, 3) LLMs do not have access to specialized knowledge bases/databases, and 4) they have a very expensive training process. However, by pairing an LLM with a knowledge base, we can access current information, reduce hallucinations, access specialized corporate knowledge, and reduce training costs.

At a high level, the RAG pipeline consists of some key steps: the user's query, document chunking and indexing, retrieval, and response generation. First, the user specifies their query for the knowledge base. This knowledge base needs to be prepared/ingested by chunking each document into small sections, indexing these documents, and storing the semantic meaning of each chunk into a vector database. The user's query is executed against the vector store to find the most relevant chunks. Augmenting the user's query with these chunks as context, the LLM generates a response to the user's query.

Figure 27: RAG flow diagram


Query
Find relevant chunks in your documents
Combine top chunks with query as context for LLM
LLM generates a response based on this context and user's query

In more detail, the RAG pipeline uses the idea of an embedding model, which converts a chunk of text to a vector representation and "captures" the semantic meaning of the text. The user's query is converted to a vector representation through this embedding model. As for the corporate knowledge base, each document is chunked, and each chunk is converted to a vector representation via embedding. These embeddings are indexed at the chunk level and stored in a vector database. To find relevant chunks, there are two common techniques: 1) exhaustive search and 2) nearest neighbor (NN) search. In exhaustive search, the query's embedding is compared against all embeddings in the vector store using the cosine similarity metric to find the most relevant chunks. In the NN search, we find the chunks closest in latent space to the user's query and return those chunks. Finally, the retrieved chunks and the user's query are supplied to the LLM to generate a response to the original query.

The pipeline above is considered "vanilla" RAG or a basic implementation of RAG. Below we will discuss some advanced variations of RAG including: 1) query methods, 2) chunking strategies, 3) indexing methods, 4) retrieval approaches, 5) response generation considerations, and 6) pipeline evaluation. However, we will begin by discussing an introduction to embedding models.

References:

1. https://projectmanagers.net/top-10-disadvantages-of-large-language-models-llm/
2. https://aws.amazon.com/what-is/retrieval-augmented-generation/
3. https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/
4. https://arxiv.org/pdf/2407.01219

Embedding Approaches in Deep Learning[2]

Introduction

At the core of many successful deep learning applications lies the concept of embeddings - mathematical representations that capture semantic relationships within data by mapping discrete entities to continuous vector spaces.

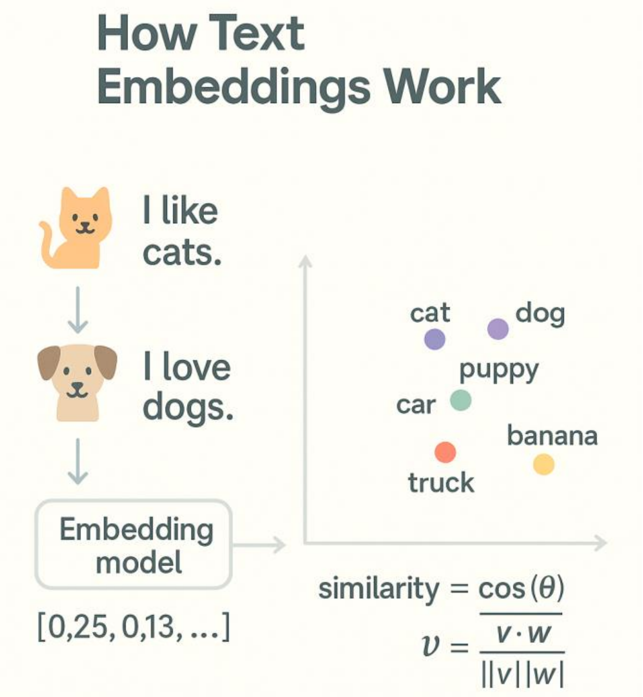


Figure 28: (Image created by ChatGPT)

These embeddings have become fundamental building blocks that enable machines to process and understand complex data, including pharmaceutical data ranging from molecular structures to clinical narratives.

Specifically, the ability to represent pharmaceutical concepts, compounds, proteins, and clinical information in a mathematically manipulable form has revolutionized multiple aspects of drug discovery, development, and pharmacovigilance. This section provides a comprehensive examination of embedding techniques, tracing their evolution from statistical foundations to contemporary neural network approaches, with specific attention to their applications and implications within pharmaceutical research and development.

[2] Embedding Model Leaderboard: https://huggingface.co/spaces/mteb/leaderboard

Evolution of Embedding Approaches

Classical Vector Representations

Before the deep learning revolution, embedding approaches relied on classical sparse representations such as one-hot encoding, bag-of-words models, and TF-IDF (Term Frequency-Inverse Document Frequency) weightings (Salton & Buckley, 1988). These techniques represented molecules and biomedical text as high-dimensional, sparse vectors where each dimension corresponded to a specific feature (e.g., presence of a functional group or occurrence of a biomedical term).

Pharmaceutical chemoinformatics traditionally employed fingerprints such as MACCS keys (Durant et al., 2002) and extended connectivity fingerprints (ECFPs) (Rogers & Hahn, 2010) to represent molecular structures. While effective for specific applications like similarity searches, these representations failed to capture complex biochemical relationships and semantic meaning inherent in pharmaceutical data.

Word2Vec and Distributional Semantics

A paradigm shift occurred with the introduction of Word2Vec by Mikolov et al. (2013), which leveraged neural networks to learn dense vector representations based on distributional semantics, the notion that words appearing in similar contexts share semantic relationships. The continuous bag-of-words (CBOW) and skip-gram architecture produced word embeddings that captured remarkable semantic and syntactic relationships.

In pharmaceutical contexts, Gómez-Bombarelli et al. (2018) pioneered the application of these techniques to create molecular embeddings by treating SMILES strings (linear text representations of molecules) as sentences, enabling the encoding of complex chemical structures into continuous vector spaces. This approach facilitated novel generative models for de novo drug design and property prediction.

GloVe and Global Matrix Factorization

Global Vectors for Word Representation (GloVe), developed by Pennington et al. (2014), combined the advantages of local context window methods with global matrix factorization. Unlike Word2Vec, GloVe explicitly factorized the logarithm of the co-occurrence matrix to derive word vectors that captured both local and global corpus statistics.

Within biomedical domains, GloVe embeddings trained on PubMed abstracts (Chiu et al., 2016) demonstrated enhanced performance in capturing relationships between biomedical concepts, enabling a more accurate classification of drug-drug interactions and adverse event detection in pharmacovigilance systems.

FastText and Subword Information

Bojanowski et al. (2017) extended Word2Vec with their FastText model, which incorporated subword information by representing words as bags of character n-grams. This approach addressed limitations in handling out-of-vocabulary words and morphologically rich languages.

This advancement proved particularly valuable in pharmaceutical contexts where technical terminology, chemical nomenclature, and protein sequences contain meaningful substructures. Schwaller et al. (2019) demonstrated that FastText embeddings could capture chemical reactivity patterns by encoding substructural features of molecules, improving reaction prediction models used in retrosynthesis planning.

Contextual Embeddings: ELMo and Beyond

A fundamental limitation of previous embedding techniques was their static nature—each word received the same vector regardless of context. This changed with the introduction of Embeddings from Language Models (ELMo) by Peters et al. (2018), which generated dynamic, context-sensitive word representations by extracting states from pre-trained bidirectional language models.

In pharmaceutical applications, contextual embeddings significantly improved named entity recognition for identifying drugs, diseases, and biological processes in clinical texts (Beltagy et al., 2019). These contextual representations captured polysemy, the phenomenon where the same term carries different meanings in different contexts—which is prevalent in biomedical literature (e.g., "receptor" as a protein or as a conceptual endpoint).

Transformer-Based Embeddings: BERT and Variants

The landscape of embedding technologies was revolutionized by the introduction of Bidirectional Encoder Representations from Transformers (BERT) by Devlin et al. (2019). BERT leveraged the transformer architecture's self-attention mechanisms to produce deeply contextual representations pre-trained on massive text corpora through masked language modeling and next sentence prediction objectives.

The pharmaceutical domain rapidly adopted and specialized this approach:

- **BioBERT**: Pre-trained specifically on biomedical literature, BioBERT (Lee et al., 2020) demonstrated superior performance in biomedical text mining tasks including relation extraction for drug-target interactions and biomedical question answering.
- **Clinical BERT**: Fine-tuned on clinical notes, this variant (Alsentzer et al., 2019) excelled at extracting medication information, adverse events, and patient outcomes from electronic health records.
- **ChemBERT**: Trained in chemical literature and SMILES representations, ChemBERT (Zhu et al., 2021) enabled improved molecular property prediction and chemical reaction outcome forecasting.
- **PubMedBERT**: Built from scratch on PubMed abstracts rather than adapted from general domain language models, PubMedBERT (Gu et al., 2021) established new state-of-the-art results on biomedical natural language processing benchmarks.

These domain-specific models addressed the vocabulary mismatch and contextual differences between general English and pharmaceutical technical language, significantly improving performance on specialized tasks.

Unified Multimodal Embeddings

Recent advances have focused on developing unified embedding spaces that simultaneously represent multiple modalities such as text with images, video, audio, or molecular data.

Vision–language encoders such as CLIP and ALIGN align an image encoder with a text encoder using hundreds of millions of alt-text pairs, so that a pathology slide embedding can sit next to the phrase “eosinophilic infiltration” in the shared space (Radford et al., 2021; Jia et al., 2021)

Adding the temporal axis, video–text models like VideoCLIP and the foundation-scale InternVideo embed snippets and captions jointly, enabling zero-shot transfer to tasks such as action recognition or cell-migration analysis (Xu et al., 2021; Wang et al., 2022)

In the audio domain, AudioCLIP and CLAP couple spectrogram encoders with language models, producing a geometry where a heartbeat waveform maps near the phrase "atrial fibrillation murmur" and supporting cross-modal retrieval and classification (Guzhov et al., 2021; Wu et al., 2024)

Molecular graph neural networks (GNNs) like Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017) and Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) generate embeddings directly from molecular graphs rather than linear representations, preserving structural information critical for pharmaceutical applications.

Multimodal approaches like Mol-BERT (Wang et al., 2022) and KG-BERT (Yasunaga et al., 2022) integrate knowledge graph information with molecular and textual data. Sicherman, A., & Radinsky, K. (2025) create embedding spaces where drugs, proteins, diseases, and biological pathways share a unified representational framework.

Foundation Model Embeddings

The most recent paradigm shift comes from large-scale foundation models that produce embeddings as a byproduct of self-supervised training on massive, diverse datasets. Models like OpenAI's text-embedding-ada-002 (OpenAI, 2022), GPT-4's embedding layer (Brown et al., 2020), and PaLM (Chowdhery et al., 2022) produce high-dimensional embeddings that capture semantic relationships across domains with unprecedented fidelity.

Pharmaceutical researchers have begun leveraging these embeddings for:

- Cross-modal retrieval between chemical structures and biomedical literature (Jin et al., 2020)
- Zero-shot classification of novel compounds into therapeutic categories (Huang et al., 2021)
- Semantic search across pharmaceutical knowledge bases (Szklarczyk et al., 2019)
- Representation alignment between protein sequences and their functional descriptions (Bepler & Berger, 2019)

The scale and generality of these models allow them to capture nuanced pharmaceutical concepts even without explicit domain specialization, though domain-adaptive pre-training continues to offer advantages for specific applications.

Applications for Pharmaceutical Research

Drug Discovery and Development

Embeddings have transformed multiple stages of the drug discovery pipeline:

Target Identification: Protein language models like ESM (Rives et al., 2021) and ProtBERT (Elnaggar et al., 2021) generate embeddings from amino acid sequences that predict protein structure and function, enabling the identification of novel druggable targets. These embeddings capture evolutionary information without requiring explicit multiple sequence alignments.

Virtual Screening: Molecular embeddings enable ultrafast screening of billions of compounds against target proteins. Lim et al. (2019) demonstrated that transformer-based molecular embeddings outperform traditional fingerprints in identifying active compounds across diverse targets, reducing false positives in high-throughput virtual screening campaigns.

De Novo Molecular Design: Generative models operating in learned embedding spaces can produce novel chemical entities with desired properties. Variational autoencoders and generative adversarial networks trained on molecular embeddings have successfully generated focused libraries for lead optimization (Jin et al., 2020).

ADMET Prediction: Embeddings of molecular structures enable more accurate prediction of absorption, distribution, metabolism, excretion, and toxicity profiles. Yang et al. (2019) showed that contextual molecular embeddings capture subtle structural features that influence cytochrome P450 metabolism and hepatotoxicity.

Clinical Development

Patient Stratification: Embeddings generated from electronic health records identify patient subpopulations likely to respond to specific treatments. Transformer-based models trained on longitudinal clinical data generate patient embeddings that capture complex comorbidity patterns and treatment response histories (Rajkomar et al., 2018).

Clinical Trial Optimization: Site selection and enrollment prediction models leverage embeddings of historical trial data, investigator performance, and demographic information to optimize clinical trial design and execution (Gao et al., 2020).

Regulatory Document Analysis: Specialized language models generate embeddings from regulatory submissions, enabling automated consistency checking, cross-reference validation, and identification of potential compliance issues (Liu et al., 2022).

Pharmacovigilance and Post-Market Surveillance

Adverse Event Detection: Social media monitoring systems employ embedding-based approaches to detect emerging safety signals. Contextual embeddings help distinguish between casual mentions and genuine adverse event reports in unstructured text (Klein et al., 2020).

Drug-Drug Interaction Prediction: Knowledge graph embeddings combined with molecular structure embeddings enable the prediction of previously unknown drug-drug interactions, helping to prevent potentially dangerous combinations (Nyamabo et al., 2021).

Signal Detection Enhancement: Embedding-based clustering of adverse event reports improves signal detection by grouping semantically related but textually diverse descriptions (Muñoz et al., 2021).

Challenges and Future Directions

Despite remarkable progress, several challenges and opportunities remain in the application of embeddings to pharmaceutical research:

Interpretability and Explainability

The black-box nature of neural embeddings presents challenges in regulatory contexts where model decisions must be explainable. Emerging techniques for embedding space visualization and feature attribution (Ying et al., 2019) are beginning to address this gap, but further work is needed to make embeddings fully interpretable to domain experts.

Data Scarcity and Privacy

While general domain embeddings benefit from virtually unlimited text, pharmaceutical applications often face limited data for rare diseases or novel compound classes. Few-shot learning approaches and transfer learning techniques are being developed to address this limitation (Altae-Tran et al., 2017).

Additionally, privacy concerns around patient data limit the sharing of embeddings derived from clinical text. Federated learning and differential privacy techniques offer promising solutions but introduce computational and performance challenges (Xu et al., 2021).

Multimodal Integration

The integration of diverse data types—genomic, proteomic, structural, clinical, and chemical—into unified embedding spaces remains challenging. Current approaches often rely on post-hoc alignment rather than truly joint learning. Multimodal transformers specifically designed for pharmaceutical data types may address this limitation (Huang et al., 2020).

Temporal Dynamics

Most current embedding approaches capture static relationships, yet biological systems and clinical trajectories are inherently dynamic. Temporal embedding techniques that capture disease progression, drug resistance evolution, and changing patient states represent an important frontier (Choi et al., 2016).

Domain Adaptation and Specificity

The optimal balance between general language knowledge and domain specificity remains an open question. While models like PubMedBERT demonstrate the value of domain-specific pre-training, they may miss broader linguistic patterns captured by general models. Efficient domain adaptation techniques that preserve general knowledge while incorporating specialized vocabulary and relationships are needed (Gururangan et al., 2020).

Conclusion

The evolution of embedding approaches—from sparse representations to contextual and multimodal neural embeddings—has fundamentally transformed how pharmaceutical researchers represent, analyze, and generate knowledge. These mathematical representations bridge the gap between symbolic biomedical knowledge and computational reasoning, enabling unprecedented advances in drug discovery, clinical development, and pharmacovigilance.

As embedding technologies continue to evolve, their integration into pharmaceutical workflows will deepen, enabling more personalized therapeutics, more efficient research processes, and safer medications. The convergence of domain expertise with these powerful representational tools promises to accelerate the development of life-saving treatments and enhance our understanding of human biology and disease. **Note that chapter continues after references.**

References

1. Altae-Tran, H., Ramsundar, B., Pappu, A. S., & Pande, V. (2017). Low data drug discovery with one-shot learning. ACS Central Science, 3(4), 283-293.
2. Alsentzer, E., Murphy, J. R., Boag, W., Weng, W. H., Jin, D., Naumann, T., & McDermott, M. B. A. (2019). Publicly available clinical BERT embeddings. In Proceedings of the 2nd Clinical Natural Language Processing Workshop (pp. 72-78).
3. Beltagy, I., Lo, K., & Cohan, A. (2019). SciBERT: A pretrained language model for scientific text. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 3615-3620).
4. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135-146.
5. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Amodei, D., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., ... Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.
6. Chiu, B., Crichton, G., Korhonen, A., & Pyysalo, S. (2016). How to train good word embeddings for biomedical NLP. In Proceedings of the 15th Workshop on Biomedical Natural Language Processing (pp. 166-174).
7. Choi, E., Bahadori, M. T., Schuetz, A., Stewart, W. F., & Sun, J. (2016). Doctor AI: Predicting clinical events via recurrent neural networks. In Machine Learning for Healthcare Conference (pp. 301-318).
8. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Fiedel, N., & et al. (2022). PaLM: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311.
9. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Vol. 1, pp. 4171-4186).
10. Durant, J. L., Leland, B. A., Henry, D. R., & Nourse, J. G. (2002). Reoptimization of MDL keys for use in drug discovery. Journal of Chemical Information and Computer Sciences, 42(6), 1273-1280.
11. Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., Bhowmik, D., & Rost, B. (2021). ProtTrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. IEEE Transactions on Pattern Analysis and Machine Intelligence.
12. Gao, J., Xiao, C., Wang, Y., Tang, W., Glass, L. M., & Sun, J. (2020). COMPOSE: Cross-modal pseudo-siamese network for patient trial matching. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 803-812).
13. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning (pp. 1263-1272).
14. Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., & Aspuru-Guzik, A. (2018).

Automatic chemical design using a data-driven continuous representation of molecules. ACS Central Science, 4(2), 268-276.

15. Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., Naumann, T., Gao, J., & Poon, H. (2021). Domain-specific language model pretraining for biomedical natural language processing. ACM Transactions on Computing for Healthcare, 3(1), 1-23.
16. Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., & Smith, N. A. (2020). Don't stop pretraining: Adapt language models to domains and tasks. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (pp. 8342-8360).
17. Huang, K., Xiao, C., Glass, L. M., & Sun, J. (2020). MolTrans: Molecular interaction transformer for drug-target interaction prediction. Bioinformatics, 37(6), 830-836.
18. Jin, W., Barzilay, R., & Jaakkola, T. (2020). Junction tree variational autoencoder for molecular graph generation. Journal of Chemical Information and Modeling, 60(9), 4055-4066.
19. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR).
20. Klein, A. Z., Sarker, A., Cai, H., Weissenbacher, D., & Gonzalez-Hernandez, G. (2020). Social media mining for birth defects research: A rule-based, bootstrapping approach to collecting data for rare health-related events on Twitter. Journal of Biomedical Informatics, 104, 103385.
21. Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: A pre-trained biomedical language representation model for biomedical text mining. Bioinformatics, 36(4), 1234-1240.
22. Lim, J., Ryu, S., Park, K., Choe, Y. J., Ham, J., & Kim, W. Y. (2019). Predicting drug-target interaction using a novel graph neural network with 3D structure-embedded graph representation. Journal of Chemical Information and Modeling, 59(9), 3981-3988.
23. Liu, X., Peng, Y., Zheng, S., Ngo, C., Wang, D., Naumann, T., & Liu, X. (2022). REGBERT: A pretrained language model for regulatory text mining in the pharmaceutical industry. Journal of the American Medical Informatics Association.
24. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
25. Muñoz, E., Godoy, J., Camino, R., & Barberan, T. (2021). Adverse drug events detection and clustering using deep language models. In Proceedings of the Thirteenth International Conference on Web Search and Data Mining (pp. 427-435).
26. Nyamabo, A. K., Yu, H., & Shi, J. Y. (2021). SSI-DDI: Substructure-substructure interactions for drug-drug interaction prediction. Briefings in Bioinformatics, 22(6), bbab133.
27. OpenAI. (2022). OpenAI API. https://openai.com/blog/new-and-improved-embedding-model
28. Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532-1543).
29. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Vol. 1, pp. 2227-2237).
30. Rajkomar, A., Oren, E., Chen, K., Dai, A. M., Hajaj, N., Hardt, M., Liu, P. J., Liu, X., Marcus, J., Sun, M., Sundberg, P., Yee, H., Zhang, K., Zhang, Y., Flores, G., Duggan, G. E., Irvine, J., Le, Q., Litsch, K.,

... Dean, J. (2018). Scalable and accurate deep learning with electronic health records. npj Digital Medicine, 1(1), 18.

31. Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., & Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. Proceedings of the National Academy of Sciences, 118(15), e2016239118.
32. Rogers, D., & Hahn, M. (2010). Extended-connectivity fingerprints. Journal of Chemical Information and Modeling, 50(5), 742-754.
33. Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24(5), 513-523.
34. Schwaller, P., Laino, T., Gaudin, T., Bolgar, P., Hunter, C. A., Bekas, C., & Lee, A. A. (2019). Molecular transformer: A model for uncertainty-calibrated chemical reaction prediction. ACS Central Science, 5(9), 1572-1583.
35. Wang, X., Li, Z., Jiang, M., Wang, S., Zhang, S., & Wei, Z. (2022). MolBERT: Molecular representation learning with language models and domain-relevant auxiliary tasks. In Proceedings of the AAAI Conference on Artificial Intelligence (pp. 4304-4312).
36. Xu, J., Glicksberg, B. S., Su, C., Walker, P., Bian, J., & Wang, F. (2021). Federated learning for healthcare informatics. Journal of Healthcare Informatics Research, 5(1), 1-19.
37. Yang, K., Swanson, K., Jin, W., Coley, C., Eiden, P., Gao, H., Guzman-Perez, A., Hopper, T., Kelley, B., Mathea, M., Palmer, A., Settels, V., Jaakkola, T., Jensen, K., & Barzilay, R. (2019). Analyzing learned molecular representations for property prediction. Journal of Chemical Information and Modeling, 59(8), 3370-3388.
38. Yasunaga, M., Leskovec, J., & Liang, P. (2022). LinkBERT: Pretraining language models with document links. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (pp. 2464-2477).
39. Ying, R., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019). GNNExplainer: Generating explanations for graph neural networks. Advances in Neural Information Processing Systems, 32, 9240-9251.
40. Zhu, Y., Meng, X., Chen, G., Zhu, C., & Lu, W. (2021). ChemBERT: A Pre-trained Language Model for Chemistry Domain. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (pp. 5612-5618).
41. Bepler, T., & Berger, B. (2019). Learning protein sequence embeddings using information from structure. Proceedings of the 7th International Conference on Learning Representations (ICLR).
42. Huang, K., Fu, T., Gao, W., Zhao, Y., Roohani, Y. N., Leskovec, J., & Zitnik, M. (2021). Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. arXiv preprint arXiv:2102.09548.
43. Jin, W., Barzilay, R., & Jaakkola, T. (2020). Hierarchical generation of molecular graphs using structural motifs. Proceedings of the 37th International Conference on Machine Learning, 80, 4839–4848.
44. Szklarczyk, D., Gable, A. L., Lyon, D., Junge, A., Wyder, S., Huerta-Cepas, J., ... & Morris, J. H. (2019). STRING v11: Protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. Nucleic Acids Research, 47(D1), D607–D613.

45. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... Sutskever, I. (2021). Learning transferable visual models from natural language supervision. arXiv preprint arXiv:2103.00020.
46. Jia, C., Yang, Y., Xia, Y., Chen, Y.-T., Parekh, Z., Pham, H., ... Duerig, T. (2021). Scaling up visual and vision-language representation learning with noisy text supervision (ICML 2021). arXiv preprint arXiv:2102.05918.
47. Xu, H., Ghosh, G., Huang, P.-Y., Okhonko, D., Aghajanyan, A., Metze, F., ... Feichtenhofer, C. (2021). VideoCLIP: Contrastive pre-training for zero-shot video-text understanding. arXiv preprint arXiv:2109.14084.
48. Wang, Y., Li, K., Li, Y., He, Y., Huang, B., Zhao, Z., ... Qiao, Y. (2022). InternVideo: General video foundation models via generative and discriminative learning. arXiv preprint arXiv:2212.03191.
49. Guzhov, A., Raue, F., Hees, J., & Dengel, A. (2021). AudioCLIP: Extending CLIP to image, text and audio. arXiv preprint arXiv:2106.13043.
50. Wu, Y., Chen, K., Zhang, T., Hui, Y., Nezhurina, M., Berg-Kirkpatrick, T., & Dubnov, S. (2024). Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation. arXiv preprint arXiv:2211.06687.
51. Sicherman, A., & Radinsky, K. (2025). *ReactEmbed: A cross-domain framework for protein–molecule representation learning via biochemical reaction networks*. *arXiv*.

Queries on your documents

The first step of an RAG pipeline is to formulate your query. However, to improve your RAG pipeline, various techniques have been introduced in the literature to modify your original query. This includes methods such as augmentation of your query, rewriting your query, or generating subqueries from your query. If your organization wants the query to refer to information contained in multiple databases or knowledge graphs, your query may be routed to the appropriate data store or converted to a SQL/Cypher command to retrieve the appropriate information. Below we review the literature for query modification algorithms.

Query re-writing

Query rephrasing/rewriting is a method to rephrase a user's query to better align the user's question with the knowledge base's document chunks to improve RAG performance. This approach is called rewrite-retrieve-read rather than the standard RAG, which is retrieve then read. To implement query rewriting, an LLM can be used to rewrite the user's query prior to retrieval. This may include: zero-shot query rewriting, few-shot prompting query rewriting, fine tuning of a pretrained model for the query rewriting task, or finding a pre-existing model already trained on the query rewriting task. These rewriting methods can improve RAG performance by removing irrelevant context, by introducing keywords to better align the query and the document chunks, or by rephrasing irregular queries. Other methods of rewriting include sub-question generation, query decomposition, step-back, and HyDE. Below we will summarize each of these approaches.

Sub-queries

In the sub-query generation approach, the initial query generates multiple subqueries to approach your question from multiple perspectives. The result from each query is retrieved and the results are merged to form the response. The goal of this method is to generate a more comprehensive response compared to the original query. Related to sub-query generation is the query decomposition method where a complex query is converted into a series of simpler queries and the step-back query method where the original query is combined with a more general query ("step back").

HyDE

HyDE or hypothetical document embeddings is another approach to query rewriting. The main insight of this method is to generate a hypothetical document/answer document to a user's query and then find relevant information similar to the hypothetical document. This is useful since documents typically contain answers to questions not the questions themselves. Rather than compare the latent spaces (embeddings) between a query and your documents, the latent spaces are better aligned by comparing embeddings of the hypothetical document to the embeddings of your documents. After combining the user's query with the retrieved documents, the LLM responds to the query. This methodology can improve RAG performance, is more efficient than fine-tuning, and can be applied to a variety of tasks.

Text to SQL, Text to Cypher

The Text to SQL and Text to Cypher approaches can be used when your organization wants to interface a natural language query with a database or knowledge graph. In the text to SQL approach, your initial query is converted to a SQL command which is then executed on a SQL database. The LLM then uses your initial query, query prompt, and the returned records to form a response. The flow diagram of this approach is given by:

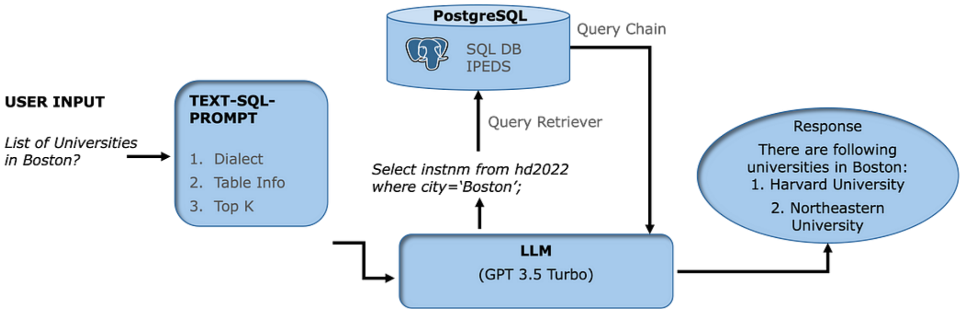


Figure 29: Example of Text to SQL [https://medium.com/@OmkarSadekar/text-to-sql-using-llm-and-context-injection-with-rag-for-large-databases-8a2ae4f171ee]

A similar approach is followed for querying knowledge graphs. Your initial query is converted to a cypher command which is executed on a knowledge graph to retrieve relevant information. The LLM will then utilize the initial query, your prompt, and retrieved information to generate a final response.

Self-query retriever

The idea of the self-query retriever is to understand the user's query as both a meta filter query and a semantic similarity search. More precisely, meta data constraints are extracted from the user's query and executed to filter your records. This meta data filtering is made possible since you need to describe each meta data field to the model. The filtered records are then compared semantically to the remainder of the user's query. As an example, consider: "What are some animated movies about insects between 1990 and 2010?". The self-query retriever would extract meta data filters: genre = animated, release date between 1990 and 2010. After filtering records on these constraints, the remaining records would be compared semantically to the insect concept to generate the final response.

Note that only some vector stores/databases support a self-query retriever, so please consult: https://python.langchain.com/docs/integrations/retrievers/self_query for a list of supported databases. For further information on the self-query retriever and code examples, please visit: https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/self_query/ and https://towardsdatascience.com/how-to-build-a-rag-system-with-a-self-querying-retriever-in-langchain-16b4fa23e9ad

Query routing: Logical and Semantic Routing

Query routing is an important step in developing an RAG pipeline. The two main types of query routing are: logical routing and semantic routing. In logical routing, an RAG pipeline can interface with multiple databases or knowledge graphs. Then, based on routing rules or query structure, the router will decide which database or knowledge graph is the most appropriate given the user's query. The query is then routed to that database to retrieve the most relevant context, and a response is generated. Similar to logical routing is the idea of semantic routing. In this approach, the semantic meaning of the query is used to find the appropriate database rather than using rules or query structure.

Multi-step transformations

In the multi-step transformation method, a complex query is iteratively modified by the model. Subcomponents of the original query are queried against a knowledge base and its response helps simplify the user's original query. As an example, "Who was the director for the film debut of actor, Liam Neeson?" The model would first query the database for the film debut of Liam Neeson which is "Pilgrim's progress". Then a second query would find the director of "Pilgrim's progress" which is Ken Anderson.

Query references:

1. https://medium.com/@samarrana407/mastering-rag-advanced-methods-to-enhance-retrieval-augmented-generation-4b611f6ca99a
2. https://medium.com/@OmkarSadekar/text-to-sql-using-llm-and-context-injection-with-rag-for-large-databases-8a2ae4f171ee
3. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/self_query/
4. https://towardsdatascience.com/how-to-build-a-rag-system-with-a-self-querying-retriever-in-langchain-16b4fa23e9ad
5. https://medium.com/@samarrana407/mastering-rag-advanced-methods-to-enhance-retrieval-augmented-generation-4b611f6ca99a
6. https://python.langchain.com/v0.1/docs/use_cases/query_analysis/techniques/step_back/
7. https://akash-mathur.medium.com/advanced-rag-query-augmentation-for-next-level-search-using-llamaindex-d362fed7ecc3

8. https://dev.to/rogiia/build-an-advanced-rag-app-query-rewriting-h3p
9. https://haystack.deepset.ai/blog/optimizing-retrieval-with-hyde
10. https://zilliz.com/learn/improve-rag-and-information-retrieval-with-hyde-hypothetical-document-embeddings
11. https://towardsdatascience.com/advanced-query-transformations-to-improve-rag-11adca9b19d1/
12. https://docs.llamaindex.ai/en/stable/optimizing/advanced_retrieval/query_transformations/#multi-step-query-transformations

Chunking Strategies

Chunking your documents is an important first step in any RAG pipeline. This step divides each document into small pieces or chunks. The chunks' token count across all retrieved chunks needs to fit within the constraints of a LLM context window (token limit). Chunking also aids in retrieval efficiency, improved retrieval relevance, and pipeline scalability. Some common chunking approaches are illustrated by the ChunkViz webapp: https://chunkviz.up.railway.app/. Below we will describe various chunking strategies and the advantages/disadvantages of each.

Fixed size chunking

The simplest chunking strategy is called fixed size chunking. In this approach, your document is divided into chunks which are defined as every k characters, words, or tokens. However, to have some semantic continuity, an overlap of m characters is recommended. Fixed size chunking can be implemented using python string functions, nltk, or spacy. While this approach is easy to implement and is computational efficient, it ignores semantic and contextual boundaries. This can result in sentences/paragraphs being broken at abrupt locations and can lose semantic meaning. Dynamic chunking such as semantic chunking, recursive chunking, agentic chunking, and other methods automatically adjusts chunk boundaries and chunk size based on linguistic context.

Recursive chunking

An alternative to fixed size chunking is called recursive chunking. One such implementation is RecursiveCharacterTextSplitter from LangChain in python. This algorithm attempts to preserve context by progressively splitting into smaller grammatical units (paragraphs, sentence, word, character) until we reach a desired chunk size.

Semantic chunking

Another improvement compared to fixed size chunking is called semantic chunking. This approach considers natural grammatical units such as sentences, paragraphs, sections, or topics to divide your documents into chunks. Some python packages such as spacy or nltk can aid in the implementation of this approach. Additionally, semantic chunking has better retrieval performance compared to fixed size chunking, since the surrounding context will be retrieved. However, semantic chunking is more complex to implement compared to fixed size chunking.

A second semantic chunking approach incorporates semantic meaning from embeddings to determine chunk breakpoints. This approach is implemented in llambdaindex's SemanticSplitterNodeParse class.

This python class embeds each document chunk and assesses chunk to chunk similarity. Groups of sequential chunks which are semantically similar are kept together which establishes a “natural” break in the document. Unfortunately, this method is much slower than previous approaches due to the computational cost to embed and semantically compare chunks.

Document structure chunking

In document structure chunking (also known as context aware chunking), we consider the document’s structure and type (Markdown, python, JS, multimodal, pdf, or HTML) during the chunking process. Some python functions include MarkdownTextSplitter for Markdown documents, PythonCodeTextSplitter for splitting on typical code structures (by function, by class), or GPT4-vision to chunk multi-modal documents. The advantage of this approach is to preserve the document’s inherent organization after chunking which can improve retrieval performance.

Agent chunking

In agent chunking, an LLM agent “reads” the document from top to bottom to decide the breakpoints in the document. This approach can be costly due to the inherent monetary cost of each LLM call. Also, the accuracy of this approach has not been assessed. One such implementation is given here:
https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/agentic_chunker.py

A second implementation is given in langchain:
https://templates.langchain.com/new?integration_name=propositional-retrieval
which combines the idea of propositions (https://arxiv.org/pdf/2312.06648.pdf) and decides how to group propositions into chunks.

Hybrid chunking

A hybrid chunking approach will combine multiple chunking approaches in your chunking strategy to realize the advantages of each approach. Perhaps one chunking strategy is used for one part of your RAG pipeline and another strategy for a different portion of your pipeline. Another example is to chunk headlines of a document differently than the main body of a document.

Chunking comparison (Table 39)

Chunking Comparison Table	Recommendations
Fixed size chunking	Not recommended
Recursive chunking	Easy to implement, may split sentences mid-way, suitable when similar chunk sizes are needed
Semantic chunking	Computationally expensive due to DL embedding models, has a better extracted context than recursive chunking
Document aware chunking	Helpful to preserve document organization. Useful for markdown, python, html, and pdf
Agent chunking	Computationally expensive, try if semantic/recursive chunking is not satisfactory
Hybrid chunking	Useful for complex document types

Choosing chunking parameters (chunk size/chunk overlap)

Some considerations in deciding your chunk size include: 1) the context window of the LLM, 2) your content type, and 3) retrieval considerations. For the first consideration, your chunk size must be less than the model's context window. Additionally, the chunk size cannot be too small or too large. If the chunk is too small, then the information context may be lost. On the other hand, a chunk that is too large may capture multiple semantic concepts. The second consideration is content type. Some content types like technical documentation may require smaller chunk sizes. Lastly, there are retrieval considerations. Typically, a RAG pipeline may retrieve the top-k chunks as context for the LLM; this will constraint the maximum size of a chunk to be the context window size divided by k. After considering these guidelines, various chunk sizes can be compared using an evaluation framework such as RAGAS (see the "RAG evaluation" section below). As for chunk overlap, users can try 10-20% of the chunk size.

Graph based chunking

A graph based chunking approach extracts entities and their relationship from your documents to build a knowledge graph. Some common options include: 1) instagraph (https://github.com/yoheinakajima/instagraph/tree/main) which uses gpt-3.5-turbo-16k, 2) DiffbotGraphTransformer() from langchain, and 3) LLMGraphTransformer() from langchain which uses a user provided LLM. During extraction, you may set requirements for the types of nodes or relationships you wish to extract and whether you want extraction of node properties (ie birth year). After construction of the knowledge graph, the graph can be visualized and stored into a graph database.

Chunking reference list:

1. https://medium.com/@anuragmishra_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d
2. https://www.sagacify.com/news/a-guide-to-chunking-strategies-for-retrieval-augmented-generation-rag
3. https://antematter.io/blogs/optimizing-rag-advanced-chunking-techniques-study
4. https://blog.gopenai.com/indexing-and-routing-strategies-in-retrieval-augmented-generation-rag-chatbots-06271908e9f6
5. https://howaibuildthis.substack.com/p/rag-in-action-beyond-basics-to-advanced
6. https://www.datacamp.com/tutorial/how-to-improve-rag-performance-5-key-techniques-with-examples
7. https://zilliz.com/learn/guide-to-chunking-strategies-for-rag
8. https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/
9. https://medium.com/@fcatser/chunking-strategies-for-llm-applications-dfd44e17b163
10. https://div.beehiiv.com/p/advanced-rag-series-indexing
11. https://medium.com/@glorat/context-aware-chunking-for-enhanced-retrieval-augmented-generation-oct23-9dcd435d9cf1
12. https://medium.com/stream-zero/chunking-strategies-and-retrieval-augmented-generation-5e684b1c6203
13. https://www.linkedin.com/pulse/advance-your-rag-dynamic-chunking-hybrid-t79zf/
14. https://www.projectpro.io/article/chunking-in-rag/1024
15. https://gleen.ai/blog/agentic-chunking-enhancing-rag-answers-for-completeness-and-accuracy/
16. https://www.mongodb.com/developer/products/atlas/choosing-chunking-strategy-rag/

17. https://antematter.io/blogs/optimizing-rag-advanced-chunking-techniques-study
18. https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb
19. https://blog.gopenai.com/indexing-and-routing-strategies-in-retrieval-augmented-generation-rag-chatbots-06271908e9f6
20. https://python.langchain.com/v0.1/docs/use_cases/graph/constructing/
21. https://docs.unstract.com/unstract/unstract_platform/user_guides/chunking/
22. https://www.mongodb.com/developer/products/atlas/choosing-chunking-strategy-rag/

Indexing

An indexing method is a technique used to organize and store data (e.g., the content and metadata of documents) in a way that allows for efficient searching and retrieval, often by creating a structured representation of the data called an index that enables fast access to relevant information. In the context of modern applications such as LLMs, search engines, and recommendation systems, indexing is crucial for handling vast amounts of unstructured or semi-structured data.

In this section, we will explore various facets of indexing methods, focusing on the key challenges and innovations that have emerged in recent years. The following subsections will introduce general techniques and concepts related to the granularity of indexing, the use of sub-chunk indexing, and comparisons between dense and sparse indexing approaches. On a more advanced level, we will delve into the emerging multi-vector representations used in DL as well as cutting-edge methods like RAPTOR and ColBERT, which offer improved performance by considering semantic similarity and hierarchical structures. Additionally, we will examine advanced indexing strategies such as knowledge graph indexing and variable-dimension indexing, which aim to enhance the contextual relevance and adaptability of retrieval systems.

A critical aspect of any indexing method is performance, such latency—the speed at which data can be retrieved, the accuracy of results, and computational efficiency. The cost of implementing these indexing solutions, in terms of computational resources and financial investment, will be considered to provide a complete view of the trade-offs involved. Finally, as privacy concerns continue to grow, especially in sectors dealing with sensitive data, we will also touch on how indexing methods can be designed to preserve user privacy.

This section aims to provide an overview of the evolving landscape of indexing methods and their role in improving the efficiency, accuracy, and scalability of modern retrieval systems. By addressing these various subtopics, we hope to equip researchers and practitioners with the tools and knowledge necessary to navigate the complexities of indexing in deep learning and related fields.

References:

1. https://web.cs.hacettepe.edu.tr/~pinar/courses/VBM681/lectures/Shutze-19web.pdf
2. https://www.coveo.com/blog/top-information-retrieval-techniques-and-algorithms/
3. https://medium.com/@j13mehul/rag-part-4-indexing-1985f4000f72
4. https://howaibuildthis.substack.com/p/rag-in-action-beyond-basics-to-advanced
5. https://www.datacamp.com/tutorial/how-to-improve-rag-performance-5-key-techniques-with-examples

General Indexing Considerations

Granularity

Indexing documents allows quick retrieval of relevant information from a user's query, with granularity referring to the level at which this indexing occurs. In RAG models, the standard approach is chunk-level indexing, where each chunk of a document is embedded as a vector and stored in a vector database, allowing the system to retrieve the most relevant chunks for a query. For instance, Jeong et al. (2024) introduced Self-BioRAG, a framework for biomedical text, where documents were segmented into 128-word chunks with 32-word overlaps to ensure comprehensive evidence coverage. Indexing can also be performed at different levels such as the sentence level, document level, or by utilizing a hierarchical structure of chunks. Document-level indexing suits large, long-form content like books or research articles, whereas sentence- or paragraph-level indexing is more common in Q&A systems to extract precise information. The granularity level influences retrieval performance: finer granularity provides more accurate results, while coarser granularity can speed up retrieval but may yield broader or less focused results.

Reference:

1. Jeong, M., Sohn, J., Sung, M., & Kang, J. (2024). Improving medical reasoning through retrieval and self-reflection with retrieval-augmented large language models. *Bioinformatics*, *40*(Supplement_1), i119-i129.

Sub-chunk Indexing

Besides indexing at the chunk level, documents can be indexed at the sub-chunk level (e.g., sentences) or at the chunk summary level. In sub-chunk indexing, individual sentences or small segments of a document are indexed and stored separately, allowing a user's query to find the most relevant sentences and retrieve their surrounding context (the chunk containing the sentence). Sub-chunk indexing is particularly useful when: 1) precision is critical, especially in cases where an exact phrase or sentence fragment is needed to answer queries accurately, and 2) the queries are complex, requiring nuanced or detailed answers. This method is commonly applied in fields like legal document retrieval, medical literature, or scientific research, where precise details are crucial.

Reference:

1. https://thetechbuffet.substack.com/p/rag-indexing-methods

Dense vs Sparse Indexing

In addition to deciding the granularity level of your indexing approach, a practitioner should decide between dense and sparse indexing. Dense indexing stores documents as high-dimensional vector embeddings, enabling a more precise semantic search but requiring significant storage and computational power. Sparse indexing, in contrast, represents documents in a more lightweight format, reducing memory consumption but potentially sacrificing some retrieval accuracy. The choice between dense and sparse indexing depends on the application's need for precision versus resource constraints. A common example of sparse indexing is Term Frequency-Inverse Document Frequency (TF-IDF), a numerical statistic used to evaluate how important a word is to a document in a collection (or corpus). In traditional TF-IDF indexing, each document is represented as a vector, where each dimension corresponds to a unique term in the corpus. Since most documents contain only a small subset of terms, many vector values are zero, resulting in a sparse representation. Unlike TF-IDF, which focuses on the frequency or importance of words, NLP techniques like Word2Vec and GloVe capture the semantic meaning of words by learning from their contexts in large text corpora, producing dense vector embeddings where similar words are located close to each other in high-dimensional space.

References:

- TF-IDF: Salton, G. (1983). Modern information retrieval. *(No Title)*. https://sigir.org/files/museum/introduction_to_modern_information_retrieval/frontmatter.pdf
- Word2Vec: Mikolov, T. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.Multi-vector Representation Indexing
- GloVe: Pennington, J., Socher, R., & Manning, C. D. (2014). *GloVe: Global Vectors for Word Representation*. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532-1543).

Latency

Some additional indexing considerations include latency, performance, cost, and privacy. Latency measures the time it takes to retrieve relevant documents and is crucial for real-time applications like chatbots or search engines, where users expect instant responses. Dense vector search, which involves navigating high-dimensional spaces, can result in higher latency. However, techniques like Approximate Nearest Neighbors (ANN) search reduce retrieval time by trading off a small amount of accuracy for much faster lookups. Spotify's open-source library, ANNOY (Approximate Nearest Neighbors Oh Yeah), is designed for fast, efficient retrieval in large-scale data. By building a forest of binary trees trained on randomized data subsets, ANNOY can quickly approximate nearest neighbors in high-dimensional spaces like song or image embeddings. Its static indexes can be shared across processes, minimizing memory usage and enabling quick distribution and lookups in production environments, making it ideal for real-time applications like Spotify's recommendation system.

Reference:

1. ANNOY: Spotify Engineering. (2016). Annoy: Approximate Nearest Neighbors in C++/Python.

Performance

In addition to latency (retrieval speed) discussed above, the performance of indexing methods is primarily determined by accuracy of retrieved results, memory efficiency, and scalability. The first metric is accuracy, where accuracy is evaluated by how well the retrieved documents align with the user query, typically using metrics like precision, recall, F1 score, and mean reciprocal rank (MRR). Precision is defined as the proportion of retrieved documents that are relevant, while recall is defined as the proportion of relevant documents that have been retrieved out of all relevant documents available in the dataset. MRR is defined in terms of the position of the first relevant result in the list, with 1 indicating that relevant results are appearing higher in the ranking and 0 meaning that no relevant documents were retrieved for any of these queries.

The second metric is memory efficiency. Efficient indexing methods should ideally occupy minimal memory while maintaining retrieval quality. For example, inverted indexes are compact for sparse data but may not be as memory efficient for dense embeddings, which are better suited to vector quantization techniques or other compression methods.

Lastly, scalability is essential for handling large datasets; effective indexing methods should maintain consistent performance and accuracy as the dataset size grows. ANN-based methods often support scalable retrieval for dense embeddings, making them more suitable for large-scale RAG systems compared to some traditional indexing structures. Overall, performance tuning involves balancing trade-offs between retrieval accuracy, speed, and resource consumption. Additionally, performance can be improved through techniques such as caching frequently accessed data or using hybrid dense-sparse models.

References:

1. Prabhune, S., & Berndt, D. J. (2024). Deploying Large Language Models With Retrieval Augmented Generation. *arXiv preprint arXiv:2411.11895*.
2. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.
3. Abbasi, M., Bernardo, M. V., Váz, P., Silva, J., & Martins, P. (2024). Revisiting Database Indexing for Parallel and Accelerated Computing: A Comprehensive Study and Novel Approaches. *Information*, *15*(8), 429.
4. Han, Y., Liu, C., & Wang, P. (2023). A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*.

Cost

The cost of indexing methods is influenced by computational requirements for retrieval, infrastructure needs for real-time applications, and ongoing expenses related to storage, memory, and maintenance. During retrieval, efficient mechanisms like approximate nearest neighbor (ANN) search can be computationally demanding, particularly with large-scale, high-dimensional datasets that require maintaining and querying embeddings. Beyond computation, the cost of storing large collections of documents or embeddings can be significant, especially if frequent updates or changes are required. The retrieval engine and the generative model often run multiple components simultaneously in parallel, which increases memory and bandwidth demands. Regular maintenance is also required to keep the knowledge base and retrieval mechanisms up-to-date, particularly for dynamic datasets. Typically, there’s a trade-off between cost and performance, with more accurate systems requiring deeper, more complex processes and larger models, driving up costs, while optimizing for cost may reduce accuracy or response quality.

References:

1. Prabhune, S., & Berndt, D. J. (2024). Deploying Large Language Models With Retrieval Augmented Generation. *arXiv preprint arXiv:2411.11895*.
2. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.
3. https://www.index.dev/blog/estimating-optimizing-cost-developing-ai-application#cost-components

Privacy

Indexing methods must account for privacy, especially when handling sensitive information. There are two main sources of potential risks in RAG in terms of data privacy: 1) source data and embeddings and 2) prompt data and LLMs. One way to mitigate this is to redact the information (for more details, see the Data Governance chapter), masking out personal identifiable data in the source document or prompt before sending it to the LLM or API provider. Advanced techniques like privacy-preserving indexing enable secure data retrieval without revealing sensitive details about the indexed data. These methods are often used when users need to query a database while protecting both the content and the queries themselves, especially important for cloud-based data services (i.e., sensitive information saved on remote servers). Cryptographic techniques, such as homomorphic encryption or secure multi-party computation, allow for secure queries and indexing without exposing raw data or results. Additionally, differential privacy can be implemented during data analysis to safeguard individual anonymity, even when auxiliary information is accessible. Differential privacy is achieved by introducing statistical noise to query results, which protects individual contributions while preserving the accuracy of aggregate insights. In Google's BigQuery, this approach allows for the analysis of sensitive datasets while ensuring compliance with privacy regulations. Additional privacy measures include deploying a local LLM or hosting an LLM within a private cloud environment to maintain data confidentiality.

References:

1. Dwork, C., & Roth, A. (2014). The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science, 9(3–4), 211-407.
2. http://infolab.stanford.edu/~bawa/Pub/ppi.pdf
3. https://www.private-ai.com/en/2024/05/23/rag-privacy-guide/
4. https://cloud.google.com/bigquery/docs/differential-privacy

Advanced Indexing Approaches

Multi-vector representation

Multi-vector representation indexing expands on the idea of traditional indexing by allowing multiple embeddings per document, including title, content, and summary. Document summaries are embedded and stored in a vector store, whereas document content is maintained in a docstore that accommodates semi-structured or unstructured data, typically in formats like JSON, BSON, or XML. These summaries and document content are linked with a common ID, with the summaries designed to capture the essence of the documents, filtering out noise, irrelevant details, and redundant information. This enables quick retrieval of relevant document summaries in response to user's queries, along with the associated documents. Facebook's Deep Learning Recommendation Model (DLRM) employs multi-vector representations to enhance model performance in recommendation scenarios by effectively managing diverse feature types and leveraging the interactions between embeddings. Additionally, multi-representation indexing is used in the parent document retriever, where each document is represented at different levels: the whole document, large chunks, and small chunks. When a user submits a query, the system will find similar small chunks or return the entire parent document if it fits in the LLM's context window; otherwise, the parent large chunk is returned. This method proves particularly valuable in applications like chatbots, search engines, and recommendation systems, where users often seek specific answers but may also benefit from additional contextual information.

References:

1. DLRM: https://ai.meta.com/blog/dlrm-an-advanced-open-source-deep-learning-recommendation-model/
2. https://blog.gopenai.com/indexing-and-routing-strategies-in-retrieval-augmented-generation-rag-chatbots-06271908e9f6
3. https://div.beehiiv.com/p/advanced-rag-series-indexing
4. https://pixion.co/blog/rag-strategies-hierarchical-index-retrieval
5. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/multi_vector/
6. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/parent_document_retriever/

ColBERT

BERT (Bidirectional Encoder Representations from Transformers) is a powerful transformer-based model designed to understand the context of words in a sentence by analyzing them in both directions, which significantly improves natural language understanding tasks. ColBERT (Contextualized Late Interaction over BERT) is an advanced retrieval model that builds on BERT's deep contextualized embeddings by offering a scalable, efficient search mechanism. Instead of computing full interactions between queries and documents at once, ColBERT splits the process into two steps: first, it encodes the query and document into token-level embeddings using BERT, and then performs a step called "late interaction", where similarity scores are computed by comparing each query token to the most similar token in the document. This step is more efficient than full cross attention by ensuring that only the most relevant parts of the document are compared with the query, thus reducing the computational load while still leveraging token-level relevance. ColBERT achieved higher precision on benchmarks like TREC and MS MARCO, while being faster and more memory-efficient than direct pairwise interactions, making it suitable for large-scale search applications. However, despite being faster than exhaustive BERT models, ColBERT can still be computationally expensive compared to traditional sparse retrieval models like BM25, with its performance hinging on the efficiency of managing and searching large embeddings.

References:

1. Khattab, O., & Zaharia, M. (2020, July). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval* (pp. 39-48).
2. Lee, J. D. M. C. K., & Toutanova, K. (2018). Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, *3*(8).

RAPTOR

Similar to the parent document retriever with multiple levels of a document representation, Stanford's RAPTOR (Recursive Abstractive Processing for Tree Organized Retrieval) enhances the precision and efficiency of retrieval by employing a hierarchical approach to chunk summaries. This method involves several key steps. First, each document is divided into chunks, and queries related to each chunk are indexed for target retrieval. Then, RAPTOR clusters semantically similar chunks and generates summaries for these clusters using an LLM. Each document has multiple summaries to capture different perspectives to increase retrieval accuracy. This summarization process is recursive and produces a tree of chunk summaries at various abstraction levels, with higher-level summaries representing broader overviews of the content. In their evaluation on benchmark datasets such as NarrativeQA, QASPER, and QuALITY, the authors demonstrated that RAPTOR's hierarchical chunk summaries and multi-summary indexing showed significant performance improvements, especially in handling complex, multi-part questions. This method proves useful in applications that require navigating large, complex documents or content, such as legal texts, research articles, or technical manuals.

Reference:

1. Sarthi, P., Abdullah, S., Tuli, A., Khanna, S., Goldie, A., & Manning, C. D. (2024). Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*.

Variable Dimension Indexing

In variable dimension indexing, the input data may differ in the number of features or attributes used to describe each data point. For instance, in content-based image retrieval, different images might have varying feature sets (such as color, texture, or shape), and variable dimension indexing allows the system to accommodate these differences. This approach can also apply to the dimension of the embeddings, where embeddings of different sizes or structures are generated based on the input data's complexity or nature. For example, if embeddings are generated from text or images, the embedding dimension could vary if different strategies or neural network layers are used to encode different types of data (e.g., varying document lengths, image sizes). By handling the variability in the input data, variable dimension indexing can adapt to datasets with heterogeneous features, thereby improving search performance and enabling more accurate retrieval across diverse datasets.

References:

1. Weber, R., Schek, H. J., & Blott, S. (1998, August). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB* (Vol. 98, pp. 194-205).
2. Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018, May). The case for learned index structures. In *Proceedings of the 2018 international conference on management of data* (pp. 489-504).
3. https://www.galileo.ai/blog/mastering-rag-how-to-select-an-embedding-model

Knowledge Graph Indexing

Knowledge graph indexing links entities (nodes) and their relationships (edges) within documents to form a graph structure. An entity is a distinct item, concept, or object --- such as a person, place, event, or idea --- that a system categorizes based on its significance. This approach enables retrieval based on both the content of the documents and the underlying relationships between entities, providing more contextually rich and accurate responses. It's especially useful for domains where understanding connections between concepts is crucial, and this structure mirrors how humans naturally comprehend relationships. For instance, Wise et al. (2020) from AWS AI proposed a COVID-19 Knowledge Graph (CKG) to extract the relationships between scientific articles on COVID-19. This domain-specific graph aims at improving information access for COVID-19 literature using primarily single-type nodes with semantic relationships extracted from texts. More recently, Chandak et al. (2023) proposed a precision medicine knowledge graph, PrimeKG, that incorporates over 20 biomedical data sources and supports heterogeneous node types, including biological processes, proteins, diseases, and phenotypes. Unlike traditional single-type knowledge graphs, this multi-typed structure allows for a more nuanced representation of biomedical knowledge, enabling complex queries that span diverse biological entities. By combining the semantic content with structural insights learned from the knowledge graph (e.g., how the entities are arranged and connected in the graph), they improved the quality of recommendations, showing how understanding document relationships can enhance retrieval.

References:

1. https://blog.google/products/search/introducing-knowledge-graph-things-not/
2. Wise, C., Ioannidis, V. N., Calvo, M. R., Song, X., Price, G., Kulkarni, N., ... & Karypis, G. (2020). COVID-19 knowledge graph: accelerating information retrieval and discovery for scientific literature. *arXiv preprint arXiv:2007.12731*.
3. Chandak, P., Huang, K., & Zitnik, M. (2023). Building a knowledge graph to enable precision medicine. *Scientific Data*, *10*(1), 67.

Real-World Applications

Indexing methods have extensive use across a wide range of real-world applications, tailored to the specific needs of different industries and systems. Below, we will discuss four examples in enterprise systems, educational tools, e-commerce platforms, and healthcare to explain the diverse applications of indexing methods.

For enterprise document management systems, such as Microsoft's 365 Copilot productivity tool, large volumes of documents (reports, manuals, guidelines) need to be indexed and searched. For instance, a person might need to search for a particular guideline on safety protocols within a large operations manual. By indexing documents at the chunk level, these systems enable faster searches by allowing the retrieval of specific sections rather than whole documents, improving both search speed and relevance.

For online learning platforms like Coursera or Udemy, educational content such as videos, transcripts, and text-based reading materials is often broken into chunks or modules. These platforms use indexing to facilitate targeted searches, helping learners find specific topics, lessons, or answers within larger courses. This indexing also enhances recommendations by allowing the platform to suggest smaller, more relevant portions of content based on the learner's search query, rather than showing entire courses.

For e-commerce platforms, multi-vector representation is used to capture diverse product attributes such as the brand, color, price, and customer reviews, allowing for personalized and highly relevant recommendations based on multiple aspects of a query. Google’s Multi-Aspect Dense Retrieval model (MADRM) can effectively capture these multiple aspects to improve retrieval quality, from general language semantics to domain knowledge, while enhancing interpretability via the proposed aspect embeddings. These systems rely on structured relationships between entities to offer insights and connections beyond keyword matching, recommending products based on individual purchase behavior and/or demographic purchase trends.

For the pharmaceutical industry, knowledge graphs offer a unified framework to organize and integrate heterogeneous data sources, aligning with the Findable, Accessible, Interoperable, and Reusable (FAIR) data principles. This approach facilitates efficient access and analysis for researchers and stakeholders, supporting the validation of medical diagnosis, the optimization of treatment strategies, and the improvement of outcomes. Knowledge graphs also enhance interoperability by establishing standardized data models and semantic relationships, enabling seamless integration and exchange of data across different systems and platforms.

As data volumes continue to grow and the need for real-time, personalized, and precise information intensifies, indexing methods will remain critical in advancing data-driven solutions across sectors, ultimately driving innovation and improving user experiences. However, it is important to keep up with emerging methods, as the field of data indexing is constantly evolving, and newer tools and techniques will continue to emerge.

References:

1. https://learn.microsoft.com/en-us/microsoftsearch/semantic-index-for-copilot
2. https://elearningindustry.com/content-chunking-engaging-course
3. https://bloomfire.com/resources/confluence-vs-sharepoint/
4. https://showcase.ems.psu.edu/node/131
5. https://research.google/pubs/multi-aspect-dense-retrieval/
6. https://www.wisecube.ai/blog/how-knowledge-graphs-are-shaping-the-future-for-the-pharmaceutical-industry/
7. https://www.ibm.com/think/topics/knowledge-graph

Summary

The following table summarizes key indexing methods discussed in this section, highlighting their pros, cons, and recommended use cases. We hope this comparison table can provide a quick reference to help readers choose the most appropriate strategy based on the nature of the data and the system's requirements.

Table 40: Summary of the pros and cons of various indexing methods

Indexing methods	Pros	Cons	Recommendations/when to use
Chunk level	• Simple and efficient to manage and query • Reduced storage requirements • Works well for moderately sized texts	• May overlook finer-grained matches • Limited recall for long or dense documents	For general-purpose retrieval tasks where moderate granularity is enough (e.g., simple FAQ, summary searches, short documents)
Sub-chunk level	• Captures finer-grained details • Higher accuracy for precise queries than chunk level	• More storage and computation costs • Increased query complexity	When fine-grained retrieval is essential (e.g., technical documentation or dense research)
Multi-vector representation	• Models multiple aspects of data • Improved query relevance and personalization	• More complex to implement than the previous two methods • High storage and computation costs	For diverse datasets with multi-faceted queries (e.g., e-commerce, multimedia)
ColBERT	• High accuracy via late interaction • Balances efficiency and effectiveness	• More complex to deploy • High storage and computation costs	For high-accuracy retrieval in competitive applications like search engines or research tools
RAPTOR	• Improves precision and efficiency of retrieval • Documents have summaries with different perspectives and abstraction levels	• Relatively new, less established • May require specialized knowledge to implement	For queries with deep context (e.g., legal, academic) and multiple retrieval steps (e.g., multi-hop)
Variable dimension	• Optimizes space and relevance • Adapts to content complexity	• Requires careful tuning • Implementation complexity	For heterogeneous datasets where certain dimensions need prioritization
Knowledge graphs	• Rich semantic relationships • Supports reasoning and complex queries	• High setup and maintenance cost	For structured and interconnected data (e.g.,

		- Not natively designed for purely textual or unstructured data	medical databases, enterprise knowledge systems)

Retrieval methods

In this section we build upon the vanilla RAG retrieval methods with more advanced retrieval methods to improve RAG performance. Some advanced retrieval methods include: reranking methods, prompt compression, and hybrid retrieval approaches.

Reranking introduction

Reranking is an advanced RAG technique which takes the initial list of retrieved documents and re-ranks them to be more relevant to the user's query. The traditional method of comparing embeddings of the user's query and document chunks is very quick with the cosine similarity metric. However, classical embeddings may lose information when compressing semantic meaning into a vector representation. To address this concern, a more complex model, such as an ANN, re-ranks the retrieved chunks to improve relevance to the user's query. These approaches are more accurate than simple cosine similarity but are much more computationally expensive.

Reranking methods

Some current approaches for reranking include: rankgpt, diversityreranker, cohere rerank, and bge-rerank. In the rankgpt approach, an LLM such as chatgpt or gpt-4 assesses the relevance between the retrieved results and the user's query to rerank the retrieved results. Additionally, distillation can be applied to GPT reranking to develop a smaller reranking model. Besides rankgpt, a user can use diversity reranking to rank retrieved results. In this method, the MMR metric (maximal marginal relevance) aims for documents to be relevant to the user's query while having novelty (ie selected documents should not be similar to previously selected documents).

Additionally, reranking methods include cohere and bge-rerank. The cohere reranker is a paid offering from cohere utilizing a cross-encoder model to assign a relevancy score between a user's query and a list of retrieved documents. From this relevancy metric, retrieved documents are reranked. As for bge-rerank, this series of models are built upon other pretrained models such as xlm-roberta, bge-m3, gemma-2b, or miniCPM. After installing their models from github (https://github.com/FlagOpen/FlagEmbedding/tree/master/examples/inference/reranker), bge models score the relevancy between each query/document pair, allowing reranking of retrieved results.

Prompt compression

In a typical RAG pipeline, your query and retrieved results as well as the query prompt are used as input to the LLM. Depending on the size of the input, this may create additional inference costs. To minimize costs while still retaining input quality, prompt compression can be utilized. This method reduces the input size (in terms of input tokens) and attempts to preserve response quality. One such framework is LLMLingua from Microsoft (https://github.com/microsoft/LLMLingua) which performs prompt compression via the PromptCompressor class and integrates with LLM frameworks such as LangChain and LlamaIndex. Similarly, compression can be applied after retrieval [ie ContextualCompressionRetriever] to compress retrieved documents or utilized within AI agent systems to reduce token counts of conversational histories.

Hybrid retrieval

In hybrid retrieval, a combination of retrieval approaches is combined into a production workflow. Based on the user's query type, we may want to route the query to one vector database or another. For other queries that don't require a semantic search, perhaps a simple keyword search is implemented. Lastly, for complex queries that want to understand the relationships between entities, perhaps the query is executed against a knowledge graph. After retrieving results from each of these data sources, results are combined and passed as context to an LLM to answer a user's original query.

RAG-fusion

RAG-fusion is a popular advanced RAG retrieval technique which gives users a more comprehensive response compared to a single query. However, this approach is much more computationally expensive compared to a single query. The RAG-fusion method begins by converting the user's initial query into multiple sub-queries to provide multiple perspectives of the original query. For each sub-query, multiple relevant chunks are retrieved. These results are merged and reranked based on the reciprocal rank score. The top chunks are then used as context to answer the user's original query.

CRAG

See LLM Generation section below for details.

Table 41: Comparison of Retrieval Approaches

Retrieval comparison table	Recommendations
Standard	Default approach
Reranking	Try if standard retrieval has poor performance; more computationally expensive
Prompt compression	May save costs for LLMs that charge by token and faster inference time. May lose meaning if you compress too much.
Hybrid	For complex workflows requiring routing to multiple databases or knowledge graphs
RAG-fusion	Obtains a more comprehensive response to a query by retrieving multiple aspects/perspectives of the original query.
CRAG	Try this approach if you are retrieving irrelevant documents or have a large number of hallucinations; CRAG will evaluate the retrieved documents and potentially combine with external data prior to response generation

RAG retrieval references:

1. https://div.beehiiv.com/p/advanced-rag-series-retrieval
2. https://www.pinecone.io/learn/series/rag/rerankers/
3. https://www.pinecone.io/learn/refine-with-rerank/
4. https://www.datacamp.com/tutorial/rankgpt-rag-reranking-agent
5. https://www.cs.cmu.edu/~jgc/publication/The_Use_MMR_Diversity_Based_LTMIR_1998.pdf
6. https://opensearch.org/docs/latest/ml-commons-plugin/tutorials/reranking-cohere/
7. https://www.mongodb.com/developer/products/atlas/prompt_compression/
8. https://neo4j.com/developer-blog/enhance-rag-knowledge-graph/
9. https://medium.com/@kbdhunga/advanced-rag-rag-fusion-using-langchain-772733da00b7
10. https://medium.com/@samarrana407/mastering-rag-advanced-methods-to-enhance-retrieval-augmented-generation-4b611f6ca99a
11. https://div.beehiiv.com/p/rag-say
12. https://github.com/langchain-ai/langchain/blob/master/cookbook/rag_fusion.ipynb
13. https://medium.com/@kiran.phd.0102/rag-fusion-revolution-a-paradigm-shift-in-generative-ai-2349b9f81c66
14. https://arxiv.org/pdf/2401.15884
15. https://www.datacamp.com/tutorial/prompt-compression
16. https://medium.com/@sahin.samia/crag-corrective-retrieval-augmented-generation-in-llm-what-it-is-and-how-it-works-ce24db3343a7

Advancements in LLM Response Generation

A key challenge in LLM response generation is balancing creativity with factual accuracy. Traditional autoregressive models, which generate responses word by word, can sometimes hallucinate information or produce misleading outputs when trained on incomplete or biased data. Retrieval-Augmented Generation (RAG) frameworks have emerged as a solution by integrating external knowledge retrieval into the generation process. However, RAG-based systems may retrieve irrelevant or inconsistent information. We now discuss some of the newer advancements in LLM response generation that enhance reliability and coherence.

Self-RAG (Self-Retrieval-Augmented Generation)

The Self-Retrieval-Augmented Generation (Self-RAG) is a framework that enhances traditional Retrieval-Augmented Generation (RAG) models by introducing a self-reflective mechanism. It addresses the limitations of current RAG systems, which often generate text without adequately considering the relevance or necessity of retrieved data. The main innovation in Self-RAG lies in its on-demand retrieval mechanism and the introduction of reflection tokens that enable the model to self-evaluate and adapt its responses. Empirical evaluations show that Self-RAG significantly outperforms state-of-the-art LLMs (e.g., ChatGPT, retrieval-augmented Llama2 Chat) and other RAG-based methods across a variety of tasks.

One of the key components of Self-RAG is the on-demand retrieval mechanism, which allows the model to retrieve information selectively based on the task's contextual requirement. On-demand systems are contrasted with tasks handled with pre-generated content ahead of time without user interaction. The ***retrieval tokens*** signal when and what to retrieve, ensuring that only necessary and relevant information is fetched. Unlike traditional RAG models that always rely on pre-retrieved data, this approach minimizes the inclusion of irrelevant or off-topic content, making the generated responses more precise and aligned with user needs.

Self-RAG also incorporates ***critique tokens***, one type of reflection token, which enables the model to perform an introspective assessment of its outputs at each stage of RAG process. These critique tokens help evaluate the factual accuracy and overall quality of the generated text. For example,

1. Are the retrieved documents relevant to the user's query? If not, rephrase the query and retrieve it again.
2. Are the statements produced by the LLM supported by the retrieved information?
3. Does the LLM's response effectively address the user's query?

By analyzing its own responses, the model can adapt its future outputs and provide explicit guidance for fact-checking, such as suggesting citations or highlighting areas that require validation. This self-evaluation capability not only improves the reliability of the model's output but also makes fact verification more efficient.

The architecture of Self-RAG is trained end-to-end, allowing seamless integration of retrieval and generation components within a single, cohesive model. It employs an arbitrary LLM for text generation, making the framework highly flexible and adaptable. Thanks to the reflection tokens, Self-RAG can be tailored for various downstream applications, from question-answering systems to complex content generation tasks.

While Self-RAG offers innovative features that improve accuracy, relevance, and adaptability, these benefits come with trade-offs in complexity and computational costs. Additionally, Self-RAG's independent processing of retrieved passages prevents synthesis across documents, limiting the generation of cohesive responses when information is distributed. It may also perform unnecessary retrievals, introducing inefficiencies and potentially irrelevant information.

Table 42: Comparison between Traditional RAG and Self-RAG Models

Feature	**Standard RAG**	**Self-RAG**
Retrieval	Fixed number of retrievals	Adaptive retrieval, can skip or retrieve multiple times
Task Flexibility	Less suitable for diverse tasks	Suitable for diverse tasks (e.g., instruction-following)
Critique & Relevance	No relevance assessment	Critiques passages and output with reflection tokens
Decoding Constraints	No constraints	Applies hard or soft constraints during decoding

References:

1. Asai, A., Wu, Z., Wang, Y., Sil, A., & Hajishirzi, H. (2023). Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.
2. https://selfrag.github.io/
3. https://openreview.net/forum?id=hSyW5go0v8&

Rewrite, Retrieve, Read (RRR) Models

The Rewrite, Retrieve, and Read (RRR) model enhances traditional RAG content by introducing a query rewriting step before retrieval, focusing on adapting the search query itself rather than the retriever or reader. This step bridges the gap between vague user queries and the knowledge required for accurate retrieval. The pipeline consists of three steps:

1. Rewrite: Refine the original query to better align with the required knowledge.
2. Retrieve: Search for related context based on the rewritten query.
3. Read: Process the input and retrieved context to generate the output

The query rewriter in Step 1 enhances information relevance, improves downstream generation tasks, and adapts dynamically to various knowledge-intensive tasks through feedback adjustments. This framework supports using a frozen LLM as the reader and a real-time web search engine as the retriever, catered with a tunable small language model as the rewriter. Analyses on open-domain and multiple-choice question answering show the effectiveness of query rewriting in improving retrieval and generation outcomes, as measured by improvements in exact match, F1 scores, and the accuracy of retrieved content in hitting or missing the correct answer.

References:

1. https://arxiv.org/pdf/2305.14283
2. https://ai.meta.com/research/publications/retrieve-and-refine-improved-sequence-generation-models-for-dialogue
3. https://medium.com/lime-eng/refine-retrieve-rebuttal-limes-rag-flow-90fd4d13c1a4

Corrective RAG (CRAG)

Corrective RAG (CRAG) is an advanced retrieval method with the aim of improving retrieval performance. This method starts by retrieving some documents potentially relevant to the user’s query through some retriever method. Next, a retrieval evaluator assesses the relevancy of each retrieved document and classifies the relevancy as correct, ambiguous, or incorrect. If the documents are deemed relevant, then the documents are chunked, and any irrelevant chunks are removed. The remaining chunks are merged. For retrieved documents that are deemed ambiguous, we combine the results of the above procedure with an external search procedure, whereby the initial query is rewritten and then executed as a web search or Wikipedia search. The resulting webpages are chunked, and irrelevant chunks are removed. Lastly, for documents classified as incorrect, we follow the external search procedure. After implementing this procedure and applying it to four datasets, CRAG’s authors concluded improved performance compared to traditional RAG approaches.

References:

1. https://arxiv.org/pdf/2401.15884
2. https://medium.com/@sahin.samia/crag-corrective-retrieval-augmented-generation-in-llm-what-it-is-and-how-it-works-ce24db3343a7

RAG evaluation

One framework to evaluate the performance of RAG pipelines is RAGAS (Retrieval Augmented Generation Assessment). This framework can be utilized by installing its python package, ragas. To begin with you need a RAG pipeline, a set of questions with their retrieved chunks, and the pipeline's outputs. Additionally, you need a ground truth dataset consisting of your questions with the ideal retrieved chunks, and the ideal answers. From the ground truth dataset, we can calculate metrics for your pipeline's retrieval and generation performance. This includes context recall and precision to evaluate retrieval, whereas faithfulness and answer relevancy are used to evaluate your generation performance. In detail, the performance metrics are:

- **retrieval recall:** the proportion of relevant contexts actually retrieved by the pipeline
- **retrieval precision:** proportion of contexts retrieved that are actually relevant to the query
- **faithfulness:** proportion of claims in the response that are also found in the retrieved chunks
- **relevancy:** looks at the relevancy of the pipeline's output to the user's query

With these performance metrics, your RAG pipeline can be evaluated, and it can be determined if your pipeline performance is satisfactory for your use-case or if the pipeline's performance needs improvement. For a code example of the ragas framework, view the link below.

Reference:

1. https://www.pinecone.io/learn/series/rag/ragas/

GraphRAG

GraphRAG was developed by Microsoft Research to address some limitations of LLMs by combining the ideas of LLMs with knowledge graphs. Specifically, GraphRAG outperforms vanilla RAG in two key areas: 1) queries that need understanding of entity relationships and 2) queries that relate to holistic understanding of documents such as document themes, summarizes of semantic concepts, or queries that require aggregation across documents. Additionally, GraphRAG provides users with increased model trust and transparency by citing information as it generates the response. This allows users to easily verify the LLM's response against the source documents.

At a technical level, GraphRAG begins by extracting entities and relationships from your document collection using an LLM. This information is used to create a knowledge graph of your private data collection. Next, a bottom-up clustering is applied to find semantic clusters and summaries are generated for each cluster. The model then retrieves relevant information from the knowledge graph and semantic cluster summaries based on the user's query. Finally, the LLM generates a response to the user's query based on the provided context. Benchmark datasets from the original paper consisting of a podcast dataset and a news dataset showed GraphRAG having improved performance compared to naïve RAG in metrics such as comprehensiveness and diversity, while naïve RAG had a higher directness metric compared to other models.

References:

1. https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/
2. https://arxiv.org/pdf/2404.16130

Guidance on deciding between long context LLMs and RAG pipelines

Advantages of long context (LC) LLMs

1. Reduced complexity/ease of use: with long context LLMs you don't need to optimize your RAG pipeline components such as embedding model, chunking methodology, or retriever strategy.
2. Inference speed: KV caching can improve the inference speed of long context LLMs
3. Entity relationships: long context LLMs may have improved understanding of information relationships in their context window compared to naïve RAG. However, newer methods such as GraphRAG can understand entity relationships.
4. Accuracy: Zhuowan Li et al's manuscript illustrated improved performance for LC models compared to naïve RAG for long context understanding on datasets from LongBench and InfinityBench. While Xinze Li et al demonstrated LC models outperform RAG for QA benchmarks, RAG outperformed LC for dialogue sources, and RAG had comparable performance for sources with papers/reports. However, recent advanced RAG techniques (reranking, query re-writing, subqueries, etc) were not considered in these papers and may increase the performance of RAG.
5. Use-cases: LC models are most suitable for tasks like multi-document comparison or summarization
6. Coherence: LC models can analyze long documents without the need for document segmentation

Advantages of RAG pipelines

1. Latency/Cost: Processing a million tokens in a long context window LLM can result in higher latency and higher cost compared to a RAG pipeline
2. Debugging: it is easier to debug hallucinations with RAG pipelines compared to long context window LLMs
3. Current information: RAG can easily interface with current information in corporate vector databases or knowledge graphs
4. Very large datasets: very large knowledge bases or corporate databases may not fit in the long context window token limit
5. Information location: LC model's performance on extracting information from their context may depend on the location of the relevant information (beginning vs middle vs end of context window).
6. Context relevancy: RAG will select chunks relevant to the user's query as the context, whereas a LC model's context may include irrelevant information
7. Use-cases: RAG is most suitable for systems such as customer support or real-time applications
8. Volume: If you have a high volume of queries, RAG may be preferred over LC models.

References:

1. https://www.vellum.ai/blog/rag-vs-long-context
2. https://arxiv.org/pdf/2407.16833
3. https://arxiv.org/pdf/2501.01880
4. https://www.superannotate.com/blog/rag-vs-long-context-llms
5. https://www.louisbouchard.ai/long-context-vs-rag/
6. https://masteringllm.medium.com/will-long-context-llms-make-rag-obsolete-17ddbc6f6412

Chapter 16: Introduction to AI agents

AI agents such as ReACT and XML Agents, coupled with sophisticated tools like LangChain, LangGraph, and LlamaIndex, are revolutionizing drug discovery, clinical research, and patient care. This section provides a comprehensive overview of these AI agents and tools, delving into their implementation details, advantages, limitations, and real-world applications within the pharmaceutical industry. Through detailed analysis and comparative studies, we aim to guide pharmaceutical stakeholders in leveraging AI technologies effectively to enhance innovation and improve healthcare outcomes.

AI Agents in Pharmaceutical Applications

ReACT Agent

ReACT (Reasoning and Acting) is an AI framework that synergizes logical reasoning and actionable responses within language models. It enables AI systems to not only process and understand complex information but also to interact with their environment by executing actions based on their reasoning. This dual capability is particularly beneficial in pharmaceutical contexts where decision-making processes often involve interpreting vast amounts of data and performing subsequent actions.

ReACT integrates chain-of-thought prompting with action sequences in LLMs. The model is trained using a combination of supervised fine-tuning and reinforcement learning from human feedback (RLHF). The training data includes annotated reasoning paths and corresponding actions, enabling the model to learn how to generate coherent reasoning steps and execute appropriate actions.

The model's architecture allows it to maintain an internal state that keeps track of the reasoning process, ensuring consistency and logical coherence in its outputs.

Table 43: Pros and Cons of the ReAct Agent

Pros	Cons
• **Autonomous Decision-Making:** ReACT empowers AI systems to make independent decisions based on complex reasoning, reducing the need for human intervention. • **Adaptability:** The model can handle multifaceted tasks by dynamically adjusting its reasoning and actions. • **Learning Capability:** Through continuous training, ReACT can improve its performance over time, adapting to new data and scenarios.	• **Data Requirements:** The model requires extensive and high-quality training data, which can be challenging to obtain in specialized domains. • **Computational Complexity:** The integration of reasoning and action planning increases computational demands, potentially affecting scalability. • **Error Propagation:** Mistakes in early reasoning steps can propagate through the action sequence, leading to compounded errors. • **Lack of Robustness:** The agent-based pipeline is overly sensitive to prompts, and it can be broken easily as the process contains various moving pieces.

In the pharmaceutical industry, ReACT can be utilized for:

- **Automated Hypothesis Generation:** Analyzing genomic data to propose new drug targets. (Song et al. ,2025)
- **Clinical Decision Support:** Assisting clinicians in treatment planning by evaluating patient data and suggesting interventions. (Sutton et al., 2020)
- **Supply Chain Optimization:** Managing logistics by predicting demand and adjusting inventory levels accordingly. (Pawar, 2024)

References:

1. Yao, S., et al. (2022). **ReAct: Synergizing Reasoning and Acting in Language Models**. *Advances in Neural Information Processing Systems*, 35, 27718–27730.
2. Song, K., Trotter, A., & Chen, J. Y. (2025). LLM Agent Swarm for Hypothesis-Driven Drug Discovery. arXiv preprint arXiv:2504.17967.
3. Sutton, R. T., Pincock, D., Baumgart, D. C., Sadowski, D. C., Fedorak, R. N., & Kroeker, K. I. (2020). An overview of clinical decision support systems: benefits, risks, and strategies for success. *npj Digital Medicine, 3*(1), 17.
4. Pawar, B. N. (2024). The role of predictive analytics in supply chain optimization. *Journal of Recent Trends in Computer Science and Engineering, 12*(1), 16–26.

XML Agent

An XML Agent is designed to process, interpret, and manipulate data in XML (Extensible Markup Language) format. XML is widely used for representing structured information in various domains, including healthcare. An XML Agent excels in parsing complex hierarchical data structures, making it invaluable for handling electronic health records (EHRs), clinical trial data, and regulatory documents.

The agent employs advanced natural language processing techniques tailored to the syntax and semantics of XML. It uses tree traversal algorithms to navigate hierarchical structures and pattern recognition methods to extract and interpret data. Machine learning models are trained on annotated XML datasets to improve the agent's ability to handle diverse and complex XML schemas.

For example, in processing EHRs, an XML Agent can extract patient information, medical history, lab results, and medication records, converting them into actionable insights for clinicians or researchers.

Table 44: Pros and Cons of an XML Agent

Pros	Cons
• **Efficiency in Structured Data Handling:** Optimized for processing XML, allowing rapid and accurate data extraction. • **Interoperability:** Facilitates integration with systems that use XML, enhancing data exchange and communication. • **Scalability:** Capable of handling large volumes of data without significant loss in performance.	• **Format Dependency:** Limited to XML data; less effective when dealing with unstructured or alternative data formats like JSON or plain text. • **Complexity of XML Schemas:** Variations in XML implementations can pose challenges, requiring additional customization.

In the pharmaceutical industry, an XML Agent can be utilized for:

- **EHR Management:** Streamlining the extraction and analysis of patient records for clinical decision support.
- **Regulatory Compliance:** Assisting in the preparation and submission of regulatory documents formatted in XML to authorities like the FDA.
- **Clinical Data Integration:** Merging data from different clinical trials or studies for meta-analyses.

Reference:

1. Zeng, X., & Parmanto, B. (2004). **XML-based Web content adaptation framework**. *IEEE Transactions on Knowledge and Data Engineering*, 16(10), 1231–1245.

AutoGPT Agent

AutoGPT is an autonomous AI agent framework that leverages Large Language Models (LLMs) to plan, reason, and act with minimal human intervention. It decomposes complex tasks into achievable subtasks, iteratively refining its approach based on intermediate results. In pharmaceutical applications, AutoGPT can assist in exploratory research, literature reviews, and data analysis, helping teams rapidly identify promising drug candidates or refine clinical strategies.

AutoGPT's workflow typically involves a "controller" LLM that outlines a series of goals and then orchestrates their completion. The model leverages chain-of-thought reasoning to generate step-by-step plans and can integrate a variety of tools—such as web search engines, databases, or domain-specific calculators—to gather information and solve problems. Through iterative feedback loops, AutoGPT identifies when additional data or analysis is required and adjusts its strategy accordingly.

For instance, in a drug discovery project, AutoGPT might start by reviewing recent scientific literature, proposing a set of molecular targets based on known pathways, and then querying chemoinformatics databases to retrieve candidate compounds. It could subsequently simulate drug-likeness scores and generate prioritized lists for experimental validation. (AutoGPT, 2024)

Table 45: Pros and Cons of AutoGPT agent

Pros	Cons
• **Autonomous Research Exploration:** Reduces manual workload by continuously self-directing research activities. • **Dynamic Planning and Iteration:** Adjusts strategies in response to new data, ensuring more agile decision-making. • **Tool Integration:** Can incorporate a range of external tools, from literature search APIs to bioinformatics calculators, improving task completion accuracy.	• **High Computational Demand:** Running continuous iterative cycles can be resource intensive. • **Data Quality Sensitivity:** Relies on the quality and availability of external information sources. • **Complex Setup:** Requires careful configuration of goals, prompts, and tool interfaces.

Pharmaceutical Industry Use Cases:

- **Lead Compound Discovery:** Automatically screen large compound libraries using chemoinformatics APIs and select promising candidates.
- **Systematic Literature Review:** Continuously scan new publications via web search tools and summarize emerging therapeutic targets.
- **Clinical Protocol Optimization:** Use external planning and simulation tools to propose and refine clinical study designs.

References:

1. Birr, Timo, et al. AutoGPT+P: Affordance-Based Task Planning Using Large Language Models. 15 July 2024, arxiv.org/abs/2402.10778, https://doi.org/10.15607/rss.2024.xx.112. Accessed 13 Dec. 2024.
2. AutoGPT. (2024, June 14). *How AI is shaping the future of drug discovery*. https://autogpt.net/how-ai-is-shaping-the-future-of-drug-discovery/

HuggingGPT Agent

HuggingGPT is an orchestration framework that coordinates multiple specialized models via a central LLM “manager.” This architecture allows the agent to tackle complex pharmaceutical tasks by leveraging domain-specific tools.

HuggingGPT works by receiving a high-level goal and decomposing it into subtasks. Each subtask is assigned to the most appropriate model or tool from a curated repository. After each specialized model provides its output, the central LLM aggregates and interprets the results, generating coherent and actionable insights. This multi-model synergy is especially valuable in pharmaceutical R&D, where diverse data types - ranging from textual regulatory guidelines to structural proteomics information - must be integrated.

Table 46: Pros and Cons of HuggingGPT Agent

Pros	Cons
• **Modular Flexibility:** Leverages specialized models and tools, ensuring optimal performance across a broad range of tasks. • **Scalability:** Can easily incorporate new domain-specific tools or updated models. • **Holistic Insights:** Integrates diverse data sources into a single, coherent analysis.	• **Complex Orchestration:** Requires careful setup and maintenance of multiple tool and model interfaces. • **Latency and Computation:** Multiple model calls can increase inference time and resource requirements. • **Dependency on External Tools:** Performance depends on the availability and quality of integrated models.

Pharmaceutical Industry Use Cases:

- **Protein Structure and Binding Analysis:** Use a protein-folding model, a docking simulation tool, and a text-mining model to identify new drug targets.
- **Regulatory Compliance Checks:** Employ regulatory guidelines parsing tools to ensure proposed trial protocols meet FDA or EMA requirements.
- **Pharmacovigilance:** Integrate adverse event detection models and literature mining services to monitor and report potential side effects.

Reference:

1. Shen, Yongliang, et al. "HuggingGPT: Solving AI Tasks with ChatGPT and Its Friends in HuggingFace." ArXiv:2303.17580 [Cs], 2 Apr. 2023, arxiv.org/abs/2303.17580.
2. Ghafarollahi, A., & Buehler, M. J. (2024). *ProtAgents: Protein discovery via large language model multi-agent collaborations combining physics and machine learning*. *Digital Discovery.*

Generative Agents with Tool Integration (e.g., Calculator, Web Search)

Generative agents that integrate tool usage combine the creative, generative capabilities of LLMs with the precision and reliability of external utilities. These agents can perform tasks like complex statistical analyses, database queries, chemical property calculations (https://arxiv.org/pdf/2410.03963), and web-based literature searches. In pharmaceutical settings, this combination streamlines decision-making and reduces human labor in knowledge-intensive tasks.

Such agents are trained or fine-tuned to recognize when external tools are needed. They use "chain-of-thought" prompting to decide what actions to take, then call appropriate APIs—such as a chemical database for structural similarity searches, a clinical trials registry for ongoing study information, or a calculator for precise dosage computations. The output is a refined solution that benefits from both the LLM's reasoning and the tool's accuracy.

For instance, when evaluating a candidate drug's ADME (Absorption, Distribution, Metabolism, and Excretion) properties, the agent might first suggest a hypothesis (e.g., "This molecule's logP suggests good oral bioavailability"). It can then call an online pharmacokinetic calculator tool to confirm this hypothesis quantitatively and query recent publications from a web-based database to validate its predictions against empirical data.

Table 47: Pros and Cons of GenAI Agents with Tools

Pros	Cons
• **Enhanced Accuracy:** Reliance on external tools reduces the risk of factual errors. • **Expanded Capabilities:** Can handle arithmetic, data retrieval, and domain-specific queries that exceed the LLM's internal knowledge. • **Iterative Improvement:** Continual refinement of prompts and tool usage leads to better outcomes over time.	• **Integration Complexity:** Requires stable APIs and careful prompt engineering to ensure the agent calls tools effectively. • **Tool Availability and Access:** Dependence on external resources may be limited by subscription fees, licensing, or connectivity. • **Latency Considerations:** Invoking multiple tools may increase overall response time.

Pharmaceutical Industry Use Cases:

- **Dose Calculation and Simulation:** Integrate a pharmacokinetic calculator to determine optimal dosing regimens.
- **Regulatory Guidance Retrieval:** Utilize a web search tool to pull up the latest regulatory guidelines and incorporate them into trial protocols.
- **Chemical Structure Analysis:** Query chemoinformatics databases and run similarity searches to identify structural analogs of a promising compound.

References:

1. Bran, A. M., Cox, S., Schilter, O., Baldassari, C., White, A. D., & Schwaller, P. (2024). Augmenting large language models with chemistry tools. Nature Machine Intelligence, 6(5), 525–535. https://doi.org/10.1038/s42256-024-00832-8
2. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. NeurIPS 2023. https://doi.org/10.48550/arXiv.2302.04761
3. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. Proceedings of ICLR 2023. https://doi.org/10.48550/arXiv.2210.03629
4. Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., & Schulman, J. (2022). WebGPT: Browser-assisted question-answering with human feedback. arXiv preprint. https://doi.org/10.48550/arXiv.2112.09332
5. Kim, S., Thiessen, P. A., Cheng, T., Zhang, J., Gindulyte, A., & Bolton, E. E. (2018). An update on PUG-REST: RESTful interface for programmatic access to PubChem. Nucleic Acids Research, 46(W1), W563–W570. https://doi.org/10.1093/nar/gky294
6. Ansari, M., Watchorn, J., Brown, C. E., & Brown, J. S. (2024). dZiner: Rational inverse design of materials with AI agents. arXiv preprint. https://doi.org/10.48550/arXiv.2410.03963

7. Lipinski, C. A., Lombardo, F., Dominy, B. W., & Feeney, P. J. (2001). Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. Advanced Drug Delivery Reviews, 46, 3–26. https://doi.org/10.1016/S0169-409X(00)00129-0
8. Daina, A., Michielin, O., & Zoete, V. (2017). *SwissADME: A free web tool to evaluate pharmacokinetics, drug-likeness and medicinal chemistry friendliness of small molecules. Scientific Reports, 7*, 42717. https://doi.org/10.1038/srep42717.
9. ChEMBL Team. (n.d.). *ChEMBL Data Web Services: Chemical searching (Substructure and Similarity)*. EMBL-EBI. Retrieved August 5, 2025, from https://chembl.gitbook.io/chembl-interface-documentation/web-services/chembl-data-web-services.

Chapter 17: MLOps Best Practices

Background

Deep learning (DL) has emerged as a powerful tool in pharmaceutical research and development. Deploying these models efficiently and reliably is crucial for maximizing their impact. The deployment process involves transitioning models from development environments like HPC to production-ready applications, such as APIs or dashboards. This section will discuss the usage of high-performance computing (HPC), Posit Connect, database systems (e.g., Oracle, Postgre, Snowflake), data storage solutions (e.g., AWS S3, HPC), and some potential challenges like GPU and memory limitations.

At Biogen R&D, our utilization of High-Performance Computing (HPC) environments is integral to providing the computational horsepower required to train sophisticated deep learning (DL) models. We have developed a set of best practices to ensure the efficacy of this process. It is suggested to establish a virtual environment specifically tailored to organizational needs, incorporating all requisite deep learning frameworks and GPU drivers, ensuring readiness for use.

For the acceleration of neural network training, it is recommended to employ GPU acceleration, taking advantage of GPUs within the HPC infrastructure. These GPUs are equipped with Tensor Cores optimized for tasks such as deep learning training and inference, enabling mixed-precision matrix multiplications at a faster rate than traditional CUDA cores.

It is suggested to manage large datasets and complex architectures through distributed training across multiple GPUs or nodes, coupled with the implementation of model checkpointing. This allows for the resumption of training from the last saved state in the event of interruptions. The recommendation extends to facilitating collaboration via version control systems like Git for effective code version management.

Transitioning DL models from development to production is recommended to follow a meticulous and detailed approach, including encapsulating models within Docker containers or creating shared virtual environments to ensure consistent behavior across environments. Robust pipelines for data preprocessing, model inference, and result post-processing are recommended. Additionally, incorporating error handling and logging mechanisms is suggested for troubleshooting, along with comprehensive testing through unit, regression, load testing, and user acceptance testing (UAT) to identify potential bottlenecks. It is crucial to document all test cases and validation steps clearly to maintain the integrity of the deployment process. This documentation should align with the standards of the software development lifecycle to ensure traceability, reproducibility, and compliance with regulatory requirements.

The deployment strategy is recommended to enhance model availability via APIs or dashboards for user-friendly access to predictions and insights, utilizing frameworks like Flask or FastAPI for APIs and tools like RShiny, Streamlit, Chainlit, or Dash for dashboards. Both APIs and Dashboards can be seamlessly deployed on Posit Connect server. Security measures such as token-based authentication, user access control, and encryption are recommended to protect sensitive data.

In real-world ML systems, the actual ML code represents only a small portion of the overall system. The surrounding infrastructure, including data pipelines, monitoring, deployment, and scalability requirements, is vast and highly complex (Sculley et al. 2015). Successfully bringing ML models into production requires a structured approach that spans the entire model lifecycle. The lifecycle of an ML project typically involves the following steps:

1. **Scoping** – define project and align with business goals
2. **Data Preparation** – define data and establish baseline; label and organize data
3. **Modeling** – select and train a model; perform error analysis
4. **Deployment** – deploy in production; monitor and maintain the system

A real-world example is Owkin’s MSIntuit CRC tool, an AI-driven diagnostic system for colorectal cancer (CRC). In the **scoping** phase, the goal was to improve CRC diagnosis using histopathological images. **Data preparation** involved collecting and labeling high-quality images with expert pathologists. For **modeling**, deep learning architectures, including U-Net and ResNet, were trained to detect microsatellite instability (MSI), incorporating federated learning to preserve privacy. Finally, the validated model was **deployed** into clinical workflows, with continuous monitoring ensuring its reliability and regulatory compliance.

Additionally, deploying large language models (LLMs) like GPT-3.5, GPT-4, and open-source alternatives such as LLaMA2 and WizardLM presents unique strategic considerations and challenges. A critical concern is data privacy, as LLMs process vast amounts of potentially sensitive information, necessitating stringent measures to ensure compliance with data protection regulations. Inference speed is another critical issue, as the real-time application of these models requires rapid response times not easily achieved with complex LLMs. For commercial models such as GPT-3.5 and GPT-4, deployment on cloud platforms like Azure provides a scalable and secure environment, benefiting from Microsoft's infrastructure and expertise in managing data privacy and computational efficiency. Open-source models, such as LLaMA and WizardLM, offer flexibility in deployment options, including Databricks and High-Performance Computing (HPC) environments, utilizing frameworks such as vLLM to optimize performance and manage resource allocation effectively. This approach allows for leveraging the specific advantages of each platform to address the challenges associated with deploying LLMs, including balancing inference speed with data privacy and computational resource requirements.

Database integration is a key component of the model deployment workflow, enabling effective data storage, retrieval, and management in a production environment. This includes connecting ML/DL pipelines to various databases for real-time data access and ensuring data consistency and integrity through well-defined schemas and appropriate data validation processes.

For unstructured data storage, cloud-based solutions like AWS S3 or HPC are recommended for scalable and cost-effective storage options. Data partitioning and indexing strategies are suggested to improve data retrieval speeds, with regular data archiving and backups to prevent data loss.

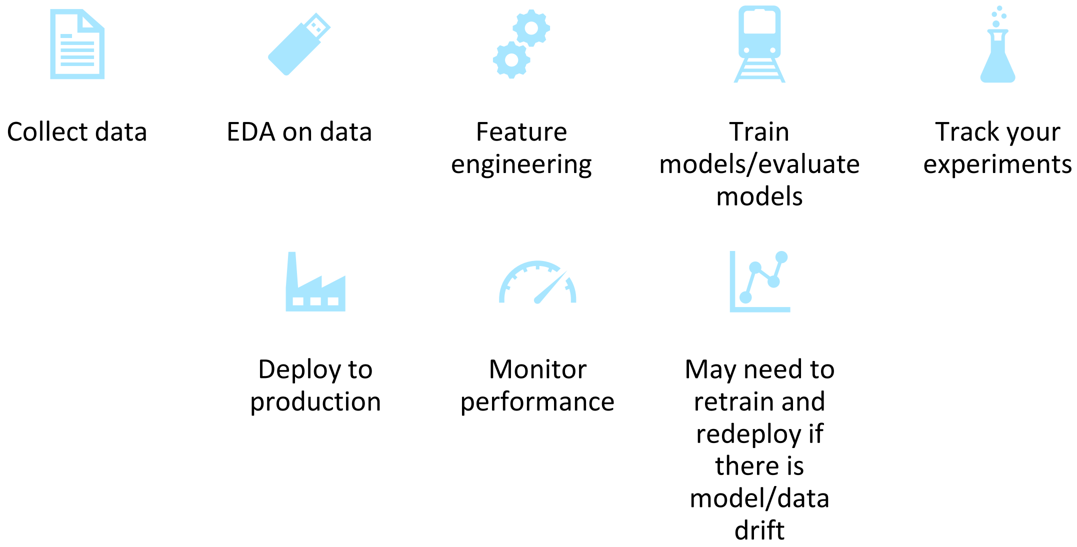
Addressing challenges such as GPU limits and memory constraints, particularly for large-scale model training, is recommended through strategies like multi-GPU training, distributed training across multiple nodes, and leveraging cloud-based GPU resources as needed.

Monitoring for scalability and performance issues is crucial as user numbers and requests grow. Implementing monitoring mechanisms to detect and address any degradation in model performance over time is recommended to maintain efficiency and overall performance of deployed models.

Introduction to MLops

MLOps (Machine Learning Operations) is the discipline of streamlining the lifecycle of machine learning models, from development to deployment and ongoing maintenance. By integrating principles from software engineering, DevOps, and data science, MLOps enables teams to automate workflows, accelerate model development, improve reproducibility, maintain model reliability at scale, and improve collaboration between data science and engineering teams. Without a structured MLOps strategy, ML models risk becoming outdated, difficult to maintain, and prone to performance degradation due to evolving data distributions. At a high level, a MLOps workflow is:

Figure 30: A sample MLOps workflow:



This chapter explores the best practices in MLOps, covering its key components, core principles, tooling comparisons, as well as introducing **LLMOps** (Large Language Model Operations).

Key Components of MLOps

MLOps builds on DevOps principles to address the unique challenges of developing, deploying, and maintaining ML models at scale. At its core, MLOps aims to unify ML system development and operational processes to streamline workflows, improve reproducibility, and ensure continuous delivery of high-performing and reliable models. The following core principles form the foundation of effective MLOps practices.

Scoping

Scoping includes identifying a business problem, defining the business objectives, aligning with the business portfolio, brainstorming and assessing the feasibility of potential AI solutions, determining milestones and success criteria, evaluating the expected benefits and risks, and budgeting for resources. You should also consider ethical questions such as algorithm fairness and biasness. Furthermore, it is essential to select an appropriate development infrastructure (e.g., Databricks, high-performance computing (HPC) clusters, cloud-based environments, or local Jupyter notebooks) to support the computational demand of the upcoming workloads. It is also essential to select an appropriate production infrastructure (e.g., cloud-based platforms, on-premises servers, hybrid environments) for seamless deployment and scaling of the AI solutions.

Data extraction

The data extraction step includes gathering and integrating relevant data from various data sources for the ML task. We also recommend keeping track of data provenance (where the data came from) and lineage (the historical record of data). To ensure consistency and comparability in results and analysis, standardization ensures that all collected data is compatible, whether the data is centralized or decentralized. This process includes setting uniform data preprocessing rules, modeling protocols, evaluation metrics, and validation procedures to maintain accuracy and reliability across different data modalities.

Exploratory data analysis (EDA)

Explore, share, and prep data for building the ML model. You should understand the data schema, distributions, and key characteristics. Also, create reproducible, editable, and shareable datasets, tables, and visualizations. Getting to know your data is essential for selecting an appropriate model, identifying biases and limitations, and ensuring data quality. Initial cleaning will be needed in this step to ensure valid insights (e.g., handling missing data, correcting any obvious data errors, standardizing formats, and applying basic filtering). Additionally, it helps refine feature engineering strategies, detect potential data drift, and establish baseline expectations for model performance. Tools such as TensorFlow Data Validation (TFDV) can help detect data issues and provide insights into a better understanding of the ML data at scale. [See chapter 1 for additional information on EDA]

Data Preparation

Data preparation includes cleaning, aggregating, and transforming data [See chapter 2 for more details]. Apply data transformation and feature engineering, if necessary. Refined processing in this step tailors the data for modeling based on what was learned from the EDA. Split the data into training, validation, and test sets [See Chapter 4 for additional information]. This step is crucial as a key principle in ML modeling is that a well-prepared dataset with a reasonable algorithm often outperforms a sophisticated algorithm applied to poor-quality data. *What defines a well-prepared dataset? Sufficient coverage of covariate distributions; consistent target variable definition; robustness to data and concept drift; and appropriately sized training and validation sets to prevent overfitting or underfitting.*

Model training and tuning

Train models using libraries like scikit-learn to use the prepared data. Optimize hyperparameters with validation data to get the best performing ML model [See Parameter Tuning chapter for more details]. When selecting a model, consider deployment constraints early if a benchmark is already established and the goal is to build and deploy. If no benchmark exists, prioritize establishing one to assess feasibility and guide model selection. For a more automated level, leverage tools such as AutoML to automatically perform trial runs and create reviewable and deployable code.

Model evaluation

Assess the model on a holdout test set with a set of evaluation metrics to quantify the performance of the trained model and ensure that the model is adequate for deployment. Note that there are ML evaluation metrics (such as accuracy) and business evaluation metrics (such as revenue or user engagement). It is important to have both technical and business teams to agree on metrics that both teams are comfortable with [See Model Evaluation Chapter for more details].

Model tracking

Systematically log model training runs, including datasets, processing steps, model versions, feature sets, hyperparameters, and evaluation metrics to ensure reproducibility and traceability. Tracking this lineage supports model comparison, model selection, auditing, and reliable deployment.

Model deployment and serving

Seamlessly deploy validated models into a target production environment, ensuring they are accessible to applications, APIs, or end-users while maintaining performance and reliability. Deployment can vary in automation levels, ranging from manual processes (hand-deployed and monitored), shadow mode (safe parallel evaluation alongside the production model without impact), and human-in-the-loop systems (human approval of outputs before actions are taken), to fully automated deployment processes with AI assistance (minimal human intervention).

Model monitoring

Continuously track model performance in production to detect issues such as concept drift, bias, or performance degradation, enabling timely intervention. A monitoring dashboard is often created, with key metrics and thresholds for alarms. These metrics and thresholds may need to be adjusted over time. Key metrics to monitor include:

- Software metrics (e.g., memory usage, latency, throughput, computation load, server load, number of users, and GPU memory usage)
- Input metrics (e.g., average input length, missing values)
- Output metrics (e.g., number of null responses, frequency of repeated user actions)

Automated model retraining

Model performance declines over time so it is important to retrain models with new data regularly and update pipelines to improve accuracy and adaptability, ensuring models remain relevant as data distributions evolve. A retraining trigger can be set based on a decay threshold, which is a predefined performance drop (e.g., accuracy falling below a certain level). When this threshold is reached, it indicates that the model's predictions are becoming less reliable, prompting the need for retraining with fresh data.

Iterative Workflows

ML is inherently experimental and requires rapid iteration across data preprocessing, feature engineering, model selection, and evaluation. MLOps supports iterative development by enabling version control of datasets, models, and configurations; continuous training; and efficient rollback or comparison of past runs. This agility is essential for improving model performance and adapting to data drift over time.

- Why it matters: without tooling to support iteration, teams lose reproducibility, slow down delivery, and make it harder to collaborate or learn from past results
- Specific tools: Airflow, Kedro, Prefect, or MLflow

Automation

Automation streamlines workflows across the machine learning lifecycle, from data ingestion and preprocessing to training, testing, and deployment. By reducing manual intervention, automation minimizes errors and accelerates iteration cycles, improving overall efficiency, consistency, and scalability.

- Why it matters: Increases efficiency, consistency, and scalability
- Specific tools: Kubeflow Pipelines, Airflow, Azure ML Pipelines, or GitHub Actions

CI/CD

CI/CD (Continuous Integration and Continuous Delivery) extends software development practices to MLOps by automating the integration, testing, and deployment of both code and models. CI ensures that the new code is regularly updated and merged into a shared repository and tested to detect issues early. CD automates the deployment of tested code to production or staging environments, ensuring faster and more reliable software releases. In MLOps, CI/CD extends to ML models, enabling automated retraining, validation, and deployment of models as data and algorithms evolve.

- Why it matters: Supports frequent updates and faster time-to-value while ensuring reliability
- Specific tools: Jenkins, GitLab CI/CD, AWS CodePipeline, or CircleCI

Reproducibility & Versioning

Reproducibility ensures that ML experiments can be consistently repeated by different team members over time. This involves version-controlling not only code but also datasets, configurations, features, model weights, and environments.

- Why it matters: Enables debugging, compliance, and collaboration
- Specific tools: DVC, MLflow, Git, Docker, or conda environments

Scalability

ML systems should scale seamlessly from development to production and be portable across environments (e.g., on-premises or multi-cloud). For extremely large datasets that might not fit into memory, distributed computing frameworks can be used to efficiently process and analyze data in parallel, ensuring scalability without performance bottlenecks.

- Why it matters: Ensures flexibility and cost-efficiency
- Specific tools: Docker, Kubernetes, MLflow, Ray, PySpark, Dask, RAPIDS, or ONNX

Monitoring & Governance

Post-deployment model monitoring tracks model performance, drift, latency, and failures in real-time. Observability includes logging, metrics, and alerts across the ML lifecycle.

- Why it matters: Detects performance degradation and operational issues early
- Specific tools: Prometheus, Grafana, Evidently, or WhyLabs

Security & Compliance

Ensuring data privacy, securing model pipelines, and complying with regulatory standards are essential in production ML systems [See additional privacy information in Chapter 3]. MLOps platforms should support identity and access management (IAM), data encryption, audit logging, and compliance with frameworks like HIPAA, GDPR, or ISO/IEC standards. Embedding security throughout the ML lifecycle reduces risk and builds trust in AI systems.

- Why it matters: Mishandling sensitive data (especially in sectors like healthcare, finance, or government) can result in legal penalties, reputational damage, or user distrust
- Specific tools: Azure ML, AWS SageMaker, MLflow, or Databricks

References:

1. Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. *IEEE access*, *11*, 31866-31879.
2. https://ml-ops.org/content/mlops-principles
3. https://neptune.ai/blog/mlops-principles
4. https://mlops-guide.github.io/Versionamento/

Three Levels of Automation

The level of automation of the key steps determines the maturity of the ML process, influencing how quickly new models can be trained and deployed in response to evolving data or updated implementations. Reference: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Footnotes:

- ***Concept drift and data drift.*** *Concept drift happens when there are changes in underlying data distribution or relationships over time that can affect model performance. For example, a fraud detection model that performs well initially might start missing fraudulent transactions because of the emergence of new types of scams. Data drift evolves with changes in the input data itself, such as feature distributions, without necessarily impacting the relationship between features and target variables. For example, a medical model designed to predict disease outcomes might see a shift in the distribution of patient data, such as the average age of patients increasing. While the relationships between features and outcomes stay the same, the changes in the data's distribution could reduce the model's accuracy.*

Table 48: Comparison of Automation Levels

MLOps Level	Characteristics	Components	Useful When	Challenges
0: Manual Process (most common)	Minimal automation Scripts and notebooks used ad hoc Disconnection between ML (statisticians, data scientists) and operation (engineers) Infrequent release iterations No active performance monitoring *Deployment of a trained model as a prediction service to production*	Basic experimentation Manual and script-driven model development and deployment	Small-scale models with infrequent updates Early-stage ML projects (e.g., get a working prototype first)	Lack of reproducibility Lack of adaptability to real-world dynamics of the environment or data High operational effort Difficult scaling
1: Continuous Training	Continuous training and delivery with new data by automating the ML pipeline Experimental-operational symmetry (same pipeline in development and production) Modularized code for components and pipelines Continuous delivery of models as prediction service on new data *Deployment of a whole training pipeline (running recurrently to serve the trained model*	Orchestrated ML experiments Automated pipeline for data validation and (online) model validation Feature store (standardizing features for training and serving) Metadata management (recording each execution of ML pipeline) ML pipeline triggers (e.g., on demand, on a schedule, on data availability, on model performance	Regular deployment of model requiring reproducibility Moderate data changes or infrequent new implementations of the pipeline Fewer pipelines to manage	Data and concept drift management Ensuring pipeline reliability Debugging automated workflows

	as the prediction service)	degradation, or on data drift)		
2: Full Automation	Robust, automated, rapid, and reliable A CI/CD system to automatically test and deploy new pipeline implementations Scalable ML systems with automated model retraining *Deployment of an ML pipeline that can automate the retraining and deployment of new models*	Source control End-to-end CI/CD (pipeline continuous integration, delivery, retraining, and monitoring)	Large-scale ML applications Frequent and rapid changes in data and business environment Real-time inference and robust monitoring	Complex system integration Managing governance and compliance Optimizing retraining cycles

Tooling Comparisons

Choosing the right tools is critical for building scalable, maintainable, and efficient machine learning systems. This section compares commonly used tools across the MLOps lifecycle. By understanding the strengths and trade-offs of each tool, teams can make informed decisions that align with their workflows, infrastructure, and governance needs.

Note that the tools highlighted here represent commonly used examples rather than an exhaustive list. Users are encouraged to consult the latest resources (e.g., Neptune.ai's platform landscape) to stay current with emerging options.

Amazon SageMaker

Amazon SageMaker is a robust full-managed cloud service for the entire ML lifecycle, with flexible services for building, training, tuning, deploying, and monitoring models. SageMaker is ideal for AWS-centric organizations needing an end-to-end managed solution. It offers a secure environment for handling enterprise and sensitive data, provided it is configured correctly and follows best practices such as using encryption, network isolation, and strict access controls.

Table 49: Pros and Cons of Amazon SageMaker

Pros	Cons
• Powerful for scaling training jobs with managed infrastructure • SageMaker pipelines provides native support for CI/CD workflows • Includes built-in algorithms, labeling tools, and explainability features	• Complex pricing and service structure; costs can increase quickly • Requires deep knowledge of AWS ecosystem • Limited built-in GUI compared to Azure ML Studio

- Strong ecosystem for large-scale and production-ready ML workloads	

Azure Machine Learning

Azure ML is a fully managed MLOps platform integrated with Azure cloud services for the entire ML lifecycle. Supports the entire ML lifecycle, from development to deployment and monitoring. Azure ML is a strong choice for enterprises already using Microsoft services with robust compliance and governance features.

Table 50: Pros and Cons of Azure Machine Learning

Pros	Cons
- Enterprise-ready with built-in governance, compliance, and RBAC (role-based access control) - Deep integration with Azure services (e.g., Azure DevOps, Azure Data Lake, and Azure OpenAI) - Easy to use; excellent support for automated ML (AutoML) and responsible AI dashboards - Offers both code-first (SDK) and low-code UI (Studio) interfaces	- Steep learning curve for complex projects - May feel overly complex or fragmented for smaller teams - Cost can escalate with large workloads or many services

Google Vertex AI

Google Vertex AI is a fully managed cloud platform for the end-to-end ML lifecycle, designed to streamline the building, training, deployment, and monitoring of ML models. It integrates with other Google Cloud services and is well-suited for organizations that rely on Google's ecosystem for scalable and flexible ML solutions.

Table 51: Pros and Cons of Google Vertex AI

Pros	Cons
- Powerful for scaling model training and deployment with Google Cloud infrastructure - Native support for CI/CD workflows through Vertex AI pipelines - Built-in tools for model explainability, feature engineering, and AutoML capabilities	- Pricing can be complex, and costs can increase quickly based on resource consumption - Steeper learning curve for users not familiar with Google Cloud - Limited ecosystem and fewer pre-built algorithms compared to AWS or Azure

• Deep integration with Google Cloud services, enabling seamless data storage and processing at scale	

MLflow

MLflow is a self-managed, open-source platform that provides flexible components for experiment tracking, model packaging, deployment, and registry. It can be self-hosted or integrated with managed platforms. MLflow provides four components:

1. *The MLflow Tracking* component provides both an API and a user interface (UI) for logging and viewing parameters, code versions, metrics, and artifacts from your machine learning runs. You can use it in scripts (e.g., Python, REST, R, JAVA), notebooks, or any environment to track experiments locally or on a server, then compare multiple runs (including across different users).
2. *The MLflow Projects* component provides a standard way to package and share ML code. A project is a folder or Git repo with your code and a descriptor (e.g., a conda.yaml file) that defines the environment and how to run the code. When used with *MLflow Tracking*, it automatically logs details like the Git commit and parameters. You can run *Projects* directly from GitHub or your own repo and combine them into workflows.
3. *The MLflow Models* component defines a format for saving and serving models in different "flavors" (e.g., TensorFlow, PyTorch, or a Python function). Models are stored in a folder with metadata and can be deployed to various platforms, like REST APIs in Docker, cloud services like AWS SageMaker and Azure ML, or Spark for batch and streaming. When logged via *MLflow Tracking*, the model is automatically linked to the run and project that created it.
4. *The MLflow Model Registry* component is a central place to manage the full lifecycle of ML models. It offers versioning, stage transitions (e.g., staging → production), annotations, and tracks which run created each model, supporting collaboration and lifecycle management.

MLflow is recommended for teams wanting full control and flexibility, especially for hybrid or multi-cloud setups.

Table 52: Pros and Cons of MLflow

Pros	Cons
• Lightweight and easy to adopt incrementally • Framework-agnostic and cloud-agnostic; works with anything from scikit-learn to PyTorch • Popular for experiment tracking and model versioning • Vendor-neutral; can be run anywhere (local environment, on-premises clusters,	• Not a fully managed service by itself; it needs hosting and integration with external tools • UI and features are relatively basic compared to cloud-native platforms

cloud platforms, and managed services including SageMaker and Azure) • Open source and free to use; developer-friendly but setup needed	

Data Version Control (DVC)

Data Version Control (DVC) is an open-source versioning tool tailored for ML workflows. Just like git is for code and metadata versioning, DVC is for data and model versioning. It extends Git to handle large datasets, models, and intermediate files that are impractical to store in a Git repository directly. It also facilitates experiment tracking, pipeline management, and collaboration across teams, making it a key component in maintaining reliable and traceable ML systems. DVC achieves this by using lightweight metafiles to track data locations and changes, while the actual data can be stored in remote storage solutions such as AWS S3, Google Drive, or Azure Blob Storage. This separation of code and data storage allows teams to work efficiently without compromising on reproducibility or traceability. DVC integrates seamlessly with Git, so teams can use familiar workflows while adding robust version control for their datasets and models.

Table 53: Pros and Cons of DVC

Pros	Cons
• Lightweight and efficient at versioning large data files and pipelines • Reproducible experiments with code, data, and models in-sync • Cloud-based; easier to share and version data and models across teams	• Not a full lifecycle tool; DVC lacks features like a built-in model registry and focuses mainly on data and model versioning, so it must be used alongside other tools for complete ML lifecycle management • Might need to manually develop extra features for full functionality • Checking for missing dependencies in DVC is quite challenging

Is there a best MLOps platform?

When choosing an ML platform, consider the following factors:

- Data Compatibility – Can it store, clean, and manage the types of data you'll use?
- Security – Does it meet your data protection and compliance needs?
- Integration – Does it work well with your current tools and systems?
- Scalability – Can it grow with your needs without major limitations or high costs?
- Ease of Setup – Is it easy to configure and get started with?
- Pricing – How is it priced (e.g., compute, data usage) and does it fit your budget?

There is no one-size-fits-all solution. The best MLOps platform is the one that fits your specific use case, team capabilities, and infrastructure. Focus on the core functionality you need and assess whether buying or building a platform offers the best value for your organization.

Summary of Tooling

In general, fully managed providers (AWS SageMaker, Azure ML, Google Vertex AI) support various frameworks for model training and deployment, provide pre-built models through model catalogs, offer strong integration with their respective cloud ecosystems and services, follow a pay-as-you-go pricing model, and allow for considerable customization.

- Azure offers robust security features and seamless integration with Microsoft services, making a strong choice for enterprise users. However, it can be expensive and often requires familiarity with the Azure ecosystem to maximize its capabilities.
- SageMaker is highly flexible and scalable with deep integration into the broader AWS ecosystem, ideal for large-scale and production-grade ML applications. That said, it has a steep learning curve and costs can escalate quickly, especially for long running or resource intensive workloads.
- Google Vertex AI stands out for its user-friendly interface and powerful AutoML capabilities, making it especially suitable for teams working with large datasets. Its limitations include a smaller selection of pre-built models compared to AWS, and potential high costs as data size and usage increase.

On the other hand, open-source platforms (such as MLflow, Kubeflow, Snowflake, and Databricks that are not connected to a specific public cloud) offer great flexibility and cloud-agnostic deployment, making them ideal for teams seeking control over their infrastructure or working in hybrid environments. However, they often require a more manual setup, maintenance, and integration effort compared to managed platforms.

Still interested? Here are some other popular ML platforms you can explore on your own: Weights & Biases (W&B), Databricks, Kubeflow, Hugging Face Inference Endpoints, Comet ML, DataRobot, Neptune.ai, Metaflow, BentoML, Valohai, Domino Data Lab, TrueFoundry, and Modelbit.

LLMOps: MLOps for Large Language Models

Introduction

MLOps has revolutionized the deployment and management of ML models by streamlining processes such as data preprocessing, model training, deployment, and monitoring. Building upon the foundations of MLOps, LLMOps emerges as a specialized discipline focusing on the operationalization of LLMs like GPT-4, LLaMA, and others. In the healthcare context, LLMOps addresses the unique challenges and opportunities presented by integrating LLMs into clinical workflows, research, and patient engagement.

LLMs have demonstrated potential in various healthcare applications, including:

- **Real-time Patient Monitoring and Clinical Decision Support**: Assisting healthcare professionals by analyzing patient data in real-time, providing immediate insights for critical care, and suggesting potential diagnoses or long-term treatment plans.

- **Administrative Automation**: Streamlining tasks such as medical coding, billing, and documentation, thereby reducing administrative burdens.
- **Patient Engagement**: Enhancing patient interactions through LLM-powered chatbots and virtual assistants that provide 24/7 support for patients.
- **Clinical Research**: Accelerating the pace of clinical research by swiftly analyzing vast datasets to uncover insights and patterns.

However, deploying LLMs in healthcare settings introduces challenges related to data privacy, model interpretability, and compliance with regulations like HIPAA. LLMOps provides a structured approach to address these challenges by:

- **Ensuring Data Security**: Implementing protocols to protect sensitive patient information during model training and inference.
- **Monitoring Model Performance**: Continuously evaluating LLM outputs to maintain accuracy and relevance in clinical contexts.
- **Facilitating Compliance**: Aligning model deployment and usage with healthcare regulations and ethical standards.

Key Differences in LLMOps vs. Traditional MLOps

Training and deploying LLMs can be different from traditional MLOps due to their scale and complexity.

- **Computational Resources**: LLM training requires massive computation, often leveraging specialized hardware like GPUs or TPUs for parallel processing. Model compression (reducing overall model size), distillation (training a smaller model to mimic a larger one), and quantization (converting model weights from high-precision to low-precision formats, e.g., 32-bit float to 8-bit integers) can help manage inference costs.
- **Transfer Learning**: Instead of training from scratch, pre-trained LLMs can be used as a starting point. Fine-tuning can then be applied on domain-specific data to adapt the model and enhance performance efficiency and save computing resources.
- **Human Feedback**: Reinforcement learning from human feedback (RLHF) and continuous user input refine model responses. Since LLM tasks are often open-ended, human feedback from end users is important for increasing LLM performance.
- **Hyperparameter Tuning**: Optimizing parameters like batch size and learning rates not only improves performance but also increases computational efficiency during training and inference for LLMs.
- **Performance Metrics:** Unlike traditional models (with metrics such as accuracy, AUC, or F1 score), LLM evaluation requires specialized assessment techniques and uses NLP-specific metrics such as Bilingual Evaluation Understudy (BLEU) and Recall-Oriented Understudy for Gisting Evaluation (ROGUE). Newer benchmarks and evaluation datasets have emerged to better capture aspects like reasoning, factual accuracy, and instruction following. See more benchmarks here.

Reference:

1. https://www.linkedin.com/pulse/dawn-llmops-healthcare-navigating-future-large-models-choudhari-curbf/

Conclusion

MLOps best practices ensure efficient, scalable, and responsible ML model deployment. Incorporating these practices helps organizations manage models more effectively, from training to production. The growing field of LLMOps introduces additional challenges, which must be addressed for successful large-scale AI deployments.

In conclusion, the deployment of deep learning models and large language models at Biogen incorporates a comprehensive set of best practices designed to optimize computational resources, ensure data privacy, and maintain high inference speeds. The strategic deployment of both commercial and open-source models across suitable platforms, from Azure to Databricks and HPC environments, underscores a commitment to innovation while addressing the inherent challenges of working with sophisticated AI models. These practices not only facilitate the seamless transition of models from development to production but also ensure that Biogen remains at the forefront of leveraging AI to drive advancements in the biotechnology sector. **Mathematical appendix follows references.**

Additional References:

1. Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. *IEEE access*, *11*, 31866-31879.
2. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, *28*.
3. Saillard, C., Dubois, R., Tchita, O., Loiseau, N., Garcia, T., Adriansen, A., ... & Svrcek, M. (2023). Validation of MSIntuit as an AI-based pre-screening tool for MSI detection from colorectal cancer histology slides. *Nature Communications*, *14*(1), 6695.
4. Deeplearning.ai MLOps specialization. Coursera.
5. https://aws.amazon.com/what-is/mlops/
6. https://ml-ops.org/
7. https://censius.ai/blogs/mlops-use-cases
8. https://chisw.com/blog/mlops-use-cases/#8_mlops_examples_that_drastically_change_the_game_for_businesses
9. https://cloud.google.com/discover/what-is-mlops?hl=en
10. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning
11. https://dlab.berkeley.edu/news/what-mlops-introduction-world-machine-learning-operations
12. https://learn.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment?view=azureml-api-2
13. https://dev.to/polarsquad/aiml-platforms-pros-and-cons-28ka
14. https://www.mlflow.org/docs/2.7.1/what-is-mlflow.html#
15. https://mlflow.org/docs/2.3.0/concepts.html

16. https://www.techtarget.com/searchenterpriseai/tip/Compare-Google-Vertex-AI-vs-Amazon-SageMaker-vs-Azure-ML
17. https://www.devopsschool.com/blog/mlflow-using-laptop-vs-databricks-vs-azure-vs-sagemaker-vs-kubernetes/
18. https://dvc.org/doc/use-cases/versioning-data-and-models
19. https://censius.ai/blogs/dvc-vs-mlflow
20. https://medium.com/walmartglobaltech/model-and-data-versioning-an-introduction-to-mlflow-and-dvc-260347cd0f6e

Chapter 18: Mathematical Appendix

1. The core formula for **variational autoencoders (VAEs)** revolves around the evidence lower bound (ELBO), which is a lower bound to the marginal log-likelihood of the observed data. The ELBO is expressed as follows:

   $$\log p(x) \ \geq E[q(z|x)]\{\log\ p(x|z) - KL(q(z|x)||p(z))\}$$

   where $p(x)$ is the marginal log-likelihood of data; $q(z|x)$ is the variational distribution of the latent variables given the data; $p(x|z)$ is the likelihood of data given a latent variable z; $p(z)$ is the prior distribution of the latent variables; $KL(q(z|x)||p(z))$ is the Kullback-Leibler divergence between the variational posterior $q(z|x)$ and the prior $p(z)$; $E[q(z|x)]$ denotes the expected value with respect to the variational posterior $q(z|x)$. The ELBO formula aims to maximize the likelihood of data while simultaneously enforcing the variational posterior $q(z|x)$ to be close to the prior $p(z)$.

2. **QLoRA** has one storage data type (e.g. NF4, W^{NF4}) and a computation data type (e.g. 16-bit BrainFloat, W^{BF16}). The quantized base model with a single LoRA adapter, to define QLoRA, can be expressed as for a single linear layer:

   $$Y^{BF16} = \ X^{BF16}W^{BF16}(W^{NF4}) + X^{BF16}L_1^{BF16}L_2^{BF16},$$

   W^{NF4} is dequantized to W^{BF16} to perform the forward and backward pass, but only the weight gradients for the LoRA parameters ($\frac{\partial X^{BF16}}{\partial W^{BF16}}$), which use 16-bit BrainFloat, are computed.

3. For image analysis, given an input image of *I* for which the network predicts class *c*, **counterfactual explanation** is produced to identify how I could change to $I^* = T(I, I')$ via a transformation (T) to find the minimum number of region replacements from I' to I to produce I^*such that the trained model classifies I^* as an instance of a specified distractor class c'. Through the optimization process in the following equation, the vector $\boldsymbol{a}$ results in the discriminative attention map on image I, representing which features (pixels) in I were edited with features copied from the distractor of I' to generate I^* such that the trained model classifies I^* to the class of c'.

   $$\min_{P,\boldsymbol{a}} ||\boldsymbol{a}||_1 \ \ s.t.\ c' = argmax\left((\mathbf{1} - \boldsymbol{a})f(I) + \boldsymbol{a}Pf(I')\right),$$

   where P is a permutation matrix which rearranges the spatial cells of $f(I')$ to align with spatial cells of $f(I)$ and where $f(\cdot)$ is a spatial feature extractor, mapping the image to a dimensional spatial feature.

4. The precision credibility of **anchors** is assessed by the conditional probability where two predictions of an original instance (x) and a permuted instance (z) are the same given a subset of features (A). Thereby, A turns out to be an anchor when the following expression holds,

   $$\mathbb{E}_{\mathcal{D}(z|A)}\left[1_{f(x)=f(z)}\right] \ \geq \ \tau, \ \ A(x) = 1,$$

where $\mathcal{D}(z|A)$ is a perturbation distribution conditioned on A and $f(x)$ is a black box model to make the individual prediction for instance x. $A(x) = 1$ when all feature predicates defined by A, are true for instance x.

References:

1. https://medium.com/@ml_dl_explained/understanding-the-evidence-lower-bound-elbo-loss-in-variational-autoencoders-423fcf19d318
2. https://doi.org/10.48550/arXiv.2305.14314
3. Goyal Y, Wu Z, Ernst J, Batra D, Parikh D, Lee S (2019) Counterfactual visual explanations. In: Proceedings of the 36th international conference on machine learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, PMLR, Proceedings of machine learning research, vol 97, pp 2376–2384
4. Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. “Anchors: high-precision model-agnostic explanations”. AAAI Conference on Artificial Intelligence (AAAI), 2018