Machine Learning (ML) – Fundamental Concepts

- Learn from a data set (DS).
- The ML system figures the patterns from data set.
- From DS, the ML system performs "fitting" and result is a Trained Model.

Types of ML:

- Supervised Learning (based on labeled data, both features (X) and expected output (Y) are present in data).
- Unsupervised Learning: No expected output (Y) given. Only features (X) are provided. The ML system finds patterns or does grouping/clustering.
- Reinforcement Learning: ML system learns from the previous experience.

Supervised Learning Techniques:

1. Linear Regression:

Output has continuous numeric values.

R2 score tells us how much variance (information) can be explained from the data. R2 score should be around 0.9 for good model performance.

Most common loss function is Square Error Loss.

Our loss function should be minimum, so do Regularization of data.

Regularization techniques are Lasso Regularization (L1), Ridge Regularization (L2) and Elastic Net Regularization (L1+L2).

Assumptions of Linear Regression:

1. There should be linear relation between data set and output.
2. There should be normalized distribution of error terms.
3. Multicollinearity (the inter-dependency among the features) should be minimum or none.

2. Logical Regression:

Output (Y) is non-linear and binary. Example: Classification.

Logistic Regression Function:

Applies to Linear Regression to bring its value between 0 and 1.

Gradient Descend: Gradient means gradual. We gradually bring the loss function to minimum by taking baby steps descend (decrease). It is an optimization algorithm. We do it iteratively.

Evaluation of Classification:

Evaluation functions are used.

1. Accuracy: To determine the accuracy of classification, the Confusion Matrix is used. The confusion matrix has four sections, true positive, true negative, false positive, false negative.

2. Recall: How many events were correctly recalled by the model.
3. Precision: How many correct events were recalled in how many chances.
4. F1-score: mean of recall and precision.

3. Decision Tree:

Tree-like structure is formed, asking questions from DS and split the data into parts.
Split by decision rules.
Two kind of decision trees:

1. Classification Tree: Data is split based on Y/N decisions.
2. Regression Tree: Data is split based on continuous real values.

How to decide decision predictor?
Choose the variable with minimum impurity.

1. Entropy (impurity): Measure of randomness. It should be close to 0.5
2. Gini Impurity (commonly used)

How well does attribute separates the data.
The information gain from the root to leaf node.

Hyper Parameters Optimization:

There are two kinds of parameters:

1. Parameters learned from the data set.
2. The parameters specific to the algorithm, not the model. They are hyper parameters.

To find the best values of parameters is called Tuning.

1. Manual Method: Manually set the param values and try them in model.
2. Grid Search CV: Test out all possible param values, one by one. Expensive option.
3. Random Search CV: Select a few random set of parameter values, try them out on model and select the best set of values.

Underfitting vs Overfitting:

Overfitting is when model performs well in training data, but not in real data.
Model learns noise and too many details from training data.
Overfitted model had higher variance and lower bias.
Possible reason: Model is too complex.
Overfitting solution: Regularization. And train with more data. And K-fold Cross Validation
Underfitting is when model didn't learn well from training data. Such model has high bias and low variance.
Possible reasons:

- Model is too simple.
- Model has high bias.

Solutions:

- Increase features in data.

- Increase model's complexity
- Increase training duration

K-fold Cross Validator:

Also called rotation estimation.

Validates whether model is overfitting or underfitting.

Split data into training data and test data. Then divide our target data set into K-folds or parts. Take each part (1,2,3...) one by one as test data. The other folds will be training data. Train model with training data and test it with the selected test data part.

KNN Algorithm:

Called K-nearest neighbors' algorithm. It is used for both classification ad regression. It is used for data set resampling or to find missing values in DS.

It is an instance-based learning technique. We choose K nearest neighbors. K is chosen to such value that the losses are minimum.

SVM (Support Vector Machine) Algorithm:

Used for both Regression and Classification. It's a popular algorithm.

Idea is to find hyperplane that separates classes. Support Vector are the data points closer to the hyperplane. Margin is the distance between the data points and the hyperplane.

Two types of SVM algorithm:

- Linear SVN
- Non-Linear SVN: Data points aren't binary.

Two classifiers in this algorithm:

- Hard Classifier: no data points are allowed in the hyperplane.
- Soft Classifier: Vice Versa

Ensemble Learning

This learning uses the wisdom of crowd.

Ensemble: list of predicates. So, the idea of this learning is to find the predicates through classifiers. There are various classifiers.

Voting Classifier: Select the model/algorithm that gets more votes.

Begging Classifier: Two step predicates selection. First is bootstrap in which each model selects samples and puts the replacement. And aggregation, which is the count or mean.

Random Forest: Applying begging classifier on decision tree.

Boosting: The idea is to train models in sequence, such that the incorrect data from model1 are fed to model2.

Two types of Boosting:

- Ada Boosting: Adjust weights to data points and add higher weights to the incorrect data points. Ada Boost makes N decision trees.
- Gradient Boosting: This boosting uses residuals.

The idea is, perform fit on model1, we get residuals. Add residuals to the loss function. Now perform fit on residuals and add the result to model1.

Unsupervised Learning Techniques:

The main idea is clustering, which is grouping (or relationship between) the data points. Between the clusters/groups, there should be internal cohesion and external separation. How to find internal similarities between data points within a group? One metric is the distance metric, e.g. Euclidean distance.

Types of Clustering:

- Exclusive: One data point should only exist in one group.
- Overlapping: One data point can be there in multiple groups.
- Hierarchical Clustering: Keep a parent-child relationship between the data points.

K-Mean Clustering:

It's a type of exclusive clustering. We divide data points into K groups. So first we need to decide the value of K. Once groups are formed, we find cluster centroid in each group. Then we calculate mean of distance of data points from centroids, and we keep adjusting the centroids.

Metrics used:

- Inertia (how much internal cohesion is there. We can use Sum of Square error). Sum of distances from data points to centroids. We balance out all the K clusters. The more the K, the less the inertia would be.
- Silhouette (means the external separation between the groups). It should be larger.

Hierarchical Clustering:

To show parent-child relationship, we use Dendrogram. The tree shows arrangement of clusters. There are two methods:

- Agglomerative Cluster (Bottom-up)
- Divisive Cluster (Top-down)

DB Scan Clustering:

Form clusters by looking at the local density. Here two parameters are used:

- Epsilon: The nearest data point. If epsilon is too small, then we'll have too many clusters. If epsilon is too large, then we may have one big cluster.
- Mini points: Another parameter.

Time-Series Analysis:

The data depends on time intervals.

Components of Time-Series Analysis:

Trend: is the variation as time passes. Trend may increase or decrease.

Seasonal: Trend changes in specific time (of month or year).

Cyclical: Cycles found in data points, in non-fixed time.

Irregularities: Nonrepeating fluctuations in data across time intervals.
Limitations of time-series analysis:

- Data points must be linear.
- Must perform transformation.

Time-Series Classification:
Stationary: No variations found as time passes. The variance is constant.
Non-Stationary: Variance is variable.

Deep Learning

Deep Learning (DL) is subset of ML. Mostly related to Artificial Nueral Networks (NN).
AI: Takes action based on data set provided.
ML: Observes patterns and predicts the outcomes.
DL: Using NN, tries to predict like humans do.

Deep NN

Used to find patterns from features and outcomes. E.g. finding complex patterns, classifying images.
There is an input layer, then there are deep or hidden layers of neurons and then output layer. Activation function is the part of the hidden layer that processes or transforms the data.
Training in DL is about adjusting the weights of model parameters to do the transformation using the activation function. Training happens in two passes:
Forward pass: finding output from input to processing.
Back pass or propagation: Updating the weights of model parameters. So, once we get the output from forward pass, we get the loss value. Using gradient descend, gradually decrease the parameter values unless we reach the minimum possible value for each parameter.
A neuron in NN is also called perception.

Activation Function

Sigmoid Function: Scales values between 0 and 1.
Tanh Function: It's a too complex function. Maps values between 1 and –1.
Relu Function: It's the commonly used one. If the value is less than 0 than it flats the output to 0.
Leaked Relu Function: Extension to Relu, it allows output below 0.
Adam Function: Commonly used function for optimization.

Learning Rate: Rate at which model parameters are changed during the training. We can increase or decrease it. If it's too low then the system is unable to move to the best parameter values, or the model will take a lot of time to reach the optimum parameter values.
If it's too high, then the training misses out the best parameter values.
Epoch: A cycle of one forward pass and one back propagation. More Epocs means a greater number of times parameters are changed. Hense, it results better performance.
Batch Size: No of instances used to complete one forward and back pass.
Example is, if there are 50 data rows and we process 10 rows in each iteration, batch size is 10. If batch size is too small, it induces noise during the loss calculation. If it is too large, then more memory is needed. But always go for the maximum batch size enough to fit into the memory.

Hyper Parameter Tuning

Process of setting the parameters on activation function and Data Set. These parameters aren't learnt through training.
Early stopping: We monitor training and when the performance improvement stops, we also stop the training. Epocs use the early stop technique.
If we see dead neurons in our NN, then use Leaky Relu function.
The relu function is used only in the hidden layers of the NN.

Problems during the back propagation:

- Vanishing Gradient

Gradient approaches zero and the model will not be improved further.

- Exploding Gradient

Gradient is too large, causing it to diverge.
Solution (to both problems):

1- Proper weight Initialization
2- Using non saturated function such as Relu.
3- Batch Normalization
4- Gradient Clipping

<u>Common Neural Network Architecture</u>

Binary Classification:

Input Neurons: 1 per feature, Output Neurons: 1
Normally used Activation function: Relu
Output Activation Function: Sigmoid
Loss Function: Binary Entropy

Multiple Classification:

Input Neurons: 1 per feature, Output Neurons: 1 per label
Normally used Activation function: Relu

Output Activation Function: Softmax
Loss Function: Categorical Entropy

Regression:

Input Neurons: 1 per feature, Output Neurons: 1
Hidden Layers Activation function: Relu
Loss Function: Mean Square Error

Convolution NN

Such NN is related to computer vision. Through Computer Vision, computer can see the outside world, captures images and manipulates the pixels or processes it.
How? On image pixel matrix, select a kernel, which is a sub-matrix. Move kernel step by step and manipulate the pixels using dot products. The step size of kernel is called stride.

Transfer Learning

A method of using parameter learnt from one model into another model. Such models have the similar tasks.
Three ways to do transfer learning:

- Train model on one data set to reuse it into another similar model.
- Use an already pre-trained model. E.g. BERT and fine-tune it for new model.
- Use pre-trained model without further training, for feature extraction only.

Feed Forward NN:

This NN moves only in one direction, input to output. It can't find relationship between words in a sentence. Features are independent. Good for image classification but bad for NLP.

Recurrent NN:

Processes new word by taking output from the previous word. There are three architectures:

- 1:1 Architecture: one input gives one output
- 1:M Architecture: e.g. image processing
- M:1 Architecture: e.g. NLP sentiment analysis
- M:M Architecture: e.g. translation. The encoder decoder architecture is an example

Faces issues such as vanishing gradient and exploding gradient.
Solution: **LSTM NN**, it's a NN that can learn the long-term dependencies.

Natural Language Processing (NLP)

From raw text, computer transforms and understands the patterns and generates output.

- Language Modeling: Predicting the next word based on sequence of words in a language. E.g. Translation.
- Text Classification: Categorization, sentiment analysis.
- Information Extraction: Extract important information, e.g. from emails.
- Information Retrieval: Finding text based on queries, e.g. Search Engine.
- Conversational Agent: Talks like human, e.g. Siri, Alexa.
- Text Summarization: e.g. Report Generation
- Question Answering: e.g. Chatbot
- Topic Modeling: Extract topics from very large text, e.g. Data Mining

Text extraction: vectorization, which means converting text into numeric format.

Vectorization Methods:

- One Hot Encoding (forms binary vector)
- Bag of words – BoW - (occurrence of words in a text, tells us words in a sentence, but doesn't tell us order of words.)
- Count of Vectorizer (no of times word occurs in text)
- TF/IDF: This formula tells us about the importance of a word w.r.t. other words. This value is very low for very common words. It's an improvement over BoW.

Word Embedding:

Numeric formatting of words in such a way that the words with the same meanings will have the same numeric representation. Two kinds:

- Skip-gram Modeler: Predicting context words from the given word. Works well with rare words and small DS.
- CBoW (Continuous Bag of Words): Predicting next word from the given context words. Works well with large DS and is faster.

Large Language Model (LLM)

LLM is a model trained on huge data. It is unsupervised learning. It works like a giant auto-complete for text. It's like a language modeling task.

Once LLM is trained from huge data, we get a **Foundation Model**.

Zero-Shot Learning: LLM can generate data for any task without learning particulars for that task. Like a multi-talented writer.

Types of LLM:

- Encoder Only – e.g. Classification
- Decoder Only – e.g. Text generation
- Encoder/Decoder - e.g. Translation, Q&A

Challenges of LLM:

- Computational cost, time cost.
- Needs huge data to train.
- Needs complex training, infrastructure and deployments.

Customize LLM for specific task:

- Prompt Tuning: Adjusting the input parameters.
- Fine Tuning: Train model on specific DS, changes model's parameters.
- Adapter: Adding light weight layer over existing model layer. The original model remains unchanged.

Transformer Architecture

RNN has limitation for parallel processing of input. Answer? Transformer.
Uses Word Embedding (from NLP) to get vector representation of text.
But long-range dependencies among words are hard to capture.

Positional Encoding: Used to know the words order in a sequence, since transformer doesn't know about words order. It injects word order into the architecture. The word order is its location/position in its sequence. A unique vector is formed called positional embedding.

Self-Attention: It is the core of the transformer architecture. It allows a word to weight its attention with other words to determine relevance.
For each word, three vectors are created; key, query and value. Key and query vectors are compared to compute the attention score. This score is used to weigh the value vector.

Multi-head Attention: Multiple self-attention layers in parallel. Each layer focuses on different word and results are combined. Each layer works independently.

Masked Multi-head Attention: It resides on the decoder layer. While generating the next word, it only looks at the previous word, not the next one, it is called auto-regressive property. It ensures causality, so the output every time depends solely on previous input. It ensures training stability by avoiding cheating.

Feed-Forward Network: Additional layers that apply transformation on each embedding to capture complex relationships.

Cross Attention: Enables decoder to use information/output from encoder to generate output sequence.

PROMPTS

Prompt Engineering: No change is made into the model. Examples: zero shot, few shots and chain of thoughts (break reasoning into steps to solve problem).

Prefix Tuning: Small set of trainable parameters are added to the prefix of the prompt.

Prompt Template: Most common way.

Contextual Prompting: Give background or context of the task with the prompt.

Prompt Learning: Use prompt and completion pair with virtual tokens or learnable prompts. Gives better performance than Prompt Engineering.

- Prompt Tuning: Soft Embeddings (optimize small set of input embeddings) for specific task are added to prompt. Adopts large pre-trained model for specific task without modifying the model parameters.
- P-Tuning: Uses prompt encoder to generate dynamic prompts by preserving the past tokens or previously generated words. It only optimizes small set of task-specific parameters. It's a prompt tuning with continuous prompt embedding. Embedding can be added anywhere with the prompt.

Pre-trained LLM -> Train on task specific Data Set -> inference by soft prompts and original input.

PEFT (Parameter Efficient Fine Tuning): Train model to downstream task with minimum modification on model parameters.

- Adapter: Small NN module is added to each layer of pre-trained model. During fine-tuning, only adapters are trained, the model remains unchanged.
- LoRA (Low Rank Adaptation): Only low rank parameters are modified.
- BitFit (Bias Term Fine-Tuning): Only bias terms are updated.
- Layer-wise freezing: Freeze majority of model, only fine-tune the top few layers.
- Prefix Tuning: Append the learnable vector.

LLM can be trained from books, web crawlers, code repos, newspapers, conversations, research papers etc.

Data Cleaning: Sub-word tokenization is tokenization based on LLM aware dictionary.

Loss Function of LLM

Aka Cost function. It’s the difference between model predictions and the truth labels.

Cross Entropy Loss: Widely used. It measures probability distribution over dictionary vs the true probability of next word in sequence. It encourages model to assign higher probability to the correct words.

Two variants:

- Negative Log Likelihood
- Perplexity

Contrastive loss: Encourages model to bring the similar sentences together dissimilar sentences further apart.

Reinforcement Learning Loss: Combines other loss functions to fine tune model based on human feedback. It prefers model that aligns well to human preferences and penalizes output that deviates from it.

Minimize Bias in LLM

Data preprocessing: Tokenization and anonymization (removal of sensitive attributes)

Statistical Tests: Conducted for Bias Detection.

Topics in Advanced Prompt Engineering

Temperature Control:

Controls the randomness of the response. Higher temperature means more diverse and creative response, while lower temperature means more focused and conservative response.

Task Framing: Framing of prompt to our task.

Open vs Close Ended Prompt:

- Open ended prompt allows wide range of creative responses.
- Closed ended prompt has clear directions to get specific and relevant information.

Data Visualization and Explanatory

To Analyse text generation:

- Word Clouds: Visualizes the most frequently used words to understand topics of focus.
- N-gram Analysis: Analyze sequence of words to identify common patterns.
- Sentiment Analysis: Plots the sentiment distribution

Evaluation of Model Performance:

- Perplexity Plot: Visualizes perplexity of LLM to track progress and potential issues.
- Confusion Matrix: For classification task.
- PCA/t-SNE Plot: Visualizes high dimensional embeddings of words and sentences to identify patterns and relationships.

Python Libraries:

- Panda: For data manipulation and analysis
- NumPy: For numerical operations
- MatPlots and Seaburn: For creating static plots and charts.
- Plotly: For interactive visualization.

NVIDIA Tools and Software

NeMo Megatron: Supports very huge models with billions of parameters. It uses model parallelism techniques.

NVIDIA Enterprise: Deploying and managing AI workloads, LLMs in production environments. To manage models across the industries.

NVIDIA Clara: Healthcare specific AI framework.

NVIDIA Metropolis: Video Analytical framework for traffic management and city services and safety of public.

NVIDIA Drive: Vehicle platform solution for self-driving vehicles.

NVIDIA Jarvis: Framework to build and deploy conversational applications.

A100 Tensor Core GPU: High performance GPU to accelerate training. Offers great performance.
NVIDIA AGX System: They are super computers, pre integrated with GPUs, networking and software tools. Offers powerful training and deploys LLM on scale.
CUDA: Parallel computing platform using GPU.
GPU = Graphical Processing Unit. Has 0ver 16K cores. Very fast.
CUDA = Compute Unified Device Architecture. Used to tap into GPU power.
CPU tells GPU to execute a code n times, very fast.
CudNN = CUDA Deep NN. Framework for Deep Learning. Uses CUDA. Building blocks for developing DL. Versions problem? For particular models you need to get particular cudNN version.
TensorRT = ML framework/SDK used to run inference on NVidia hardware. Highly optimized. Way to run the fastest inference. It reduces latency while doing inference by optimizing the computational graph. It uses INT8 precision, giving high performance.
Convert your model into TensorRT: Use tools loke TensorFlow or PyTorch.
Torch-TensorRT integration: PyTorch has eager mode and script mode (optimized for prod) Script mode generates Torch script for our model. It's an intermediate representation of our model. Torch-TensorRT automatically generates the Py script. Optimized torch script module.
Triton Inference Server:
Open source inference serving software that helps standardize model deployment and execution. Delivers fast & scalable AI to production. Supports trained model from any framework, so gives freedom to AI teams to use any framework for model. Reliable environment for LLMs.
We can query Triton Inference Server vis HTTP or gRPC. Models are loaded from cloud storage or File System. Then the Trinton server batch the model requests to serve them sequentially. Then scheduler sends request to the appropriate framework to perform computation. It supports Kubernetes.
NVIDIA NeMo:
Open-source framework that contains prebuilt models, tools for optimization, training and deployment, and prompt templates. End to end cloud native framework. Also used to run the model at scale. Supports multi-modality, like text to text, text to image, text to 3D model and image to image. Also provides modules for NLP. NeMo also provides data visualization tools.

For data curation, it does extraction, deduplication and filtration of unstructured data at scale. Uses distributed training using 3D parallelism techniques to efficiently use multiple GPUs across multiple nodes by distributing the model on scale.
Model customization: Once foundation model is trained, it can be customized using adapters or p-tuning. Continuously improve the model using enforcement learning by human responses. We can use skills and add focus on user's task.
NeMo uses Triton Inference Server to deliver latency and through put in single, multiple GPU or nodes configuration. To Accelerate the inference.
NeMo Guardrails: To monitor the model performance, Guardrails can be used to monitor response. Puts safety measures and constraints to make the conversational systems safe and reliable. Can filter out toxic or offensive language and bias.
NVIDIA Riva: Deploying real-time conversational AI systems, customizing and optimizing the conversational experiences.
NVIDIA RAPIDS: Libraries to accelerate pipelines on GPU.
NVIDIA Nim: Set of inference microservices for deploying AI models. It is a ready to serve microservice. It is kind of serverless as there is no need to manage infrastructure while deploying our LLM. The model is nim is pre-tuned and optimized.
TAO Toolkit: Toolkit built on PyTorch/Tensorflow . A low code, CLI base solution allows you to adapt, fine-tune and optimize pre-train model with your own data. It is compatible with TensorRT and uses FP16, INT8 for optimization. It automates hyper parameter tuning. Can be integrated with Triton Inference Server. It supports transfer learning unlike TensorRT. The models are deployed into DGX or cloud as well.
TrensorRT-LLM: Uses TrensorRT to deploy LLMs on GUPs. It's a python API.
NVIDIA DGX Cloud: Nvidia's cloud platform for LLM deployment.
NVIDIA NGC: GUP optimized AI solution that gives built-in security and performance optimization.
NVIDIA NGC Catalog: Contains pre-trained models, containers and helm charts.

Simplified Summary of RAG (Retrieval-Augmented Generation)

RAG consists of **two main processes**:

1. **Document Ingestion (Offline)** – Prepares and stores information for retrieval.
2. **User Query, Retrieval, and Response Generation (Online)** – Answers user queries by retrieving relevant information and generating responses.

Document Ingestion (Offline Process)

This phase processes and indexes documents so they can be used later for answering queries.

Steps:

1. **Ingest Data** – Documents from databases, PDFs, or live feeds are loaded using tools like LangChain or LlamaIndex.
2. **Split Text** – Large documents are broken into smaller chunks so they fit into embedding models.
3. **Generate Embeddings** – A deep learning model (e.g., E5, word2vec, OpenAI embeddings) converts text into high-dimensional vectors.
4. **Store in Vector Database** – These embeddings, along with their documents, are stored for fast search and retrieval.

NVIDIA Tools for This Phase:

- **Data Curator** – Removes duplicate data efficiently.
- **RAPIDS RAFT & cuDF** – Accelerate data processing on GPUs.

User Query, Retrieval, and Response Generation (Online Process)

When a user submits a question, the system retrieves relevant information and generates a response.

Steps:

1. **Convert Query into an Embedding** – The user's question is converted into a vector using an embedding model.
2. **Retrieve Relevant Documents** – The system searches for similar embeddings in the vector database (using cosine similarity or other distance measures).
3. **Pass to LLM** – The query and retrieved documents are sent to a Large Language Model (LLM) to generate a response.

NVIDIA Tools for This Phase:

- **TensorRT-LLM & Triton Inference Server** – Optimizes LLM performance for fast inference.
- **NeMo Retriever** – Provides low-latency, high-accuracy retrieval.

Challenges in RAG

- **Finding a good retriever** – Many datasets have access restrictions.
- **Vague user queries** – Users don't always phrase questions clearly.
- **Contextual understanding** – Combining information from multiple sources is complex.
- **Computational cost** – Large models require significant resources.

How to Improve RAG Accuracy?

Best Practices:

- Evaluate using **Ragas, ARES, and Bench** frameworks.
- Test different **text chunking** strategies (e.g., sentence splitting, recursive chunking).
- Try **different retrieval techniques** (e.g., BM25, knowledge graphs).

- **Re-rank retrieved results** for better accuracy.
- Modify the **LLM's system prompt** to fine-tune responses.

How to Reduce Latency?

- Reduce **Time to First Token (TTFT)** and **Time Per Output Token (TPOT)**.
- Use **smaller LLMs** to lower response times.

LangChain Components in RAG

1. **Data Preparation:**
    a. **Ingestion** – WebBaseLoader, PyPDFLoader.
    b. **Processing** – RecursiveCharacterTextSplitter.
2. **Embedding:** OpenAIEmbeddings or NVIDIA's **Text QA Embedding Model** (optimized for QA tasks).
3. **Retrieval:** Searching in the vector database using .as_retriever().

Miscellaneous notes

Principal Component Analysis (PCA) reduces dimensionality of dataset.
Multi-instance GPU (MIG) is used to deploy multiple instances into single GPU, enhances scalability and reduces latency.
Differential privacy adds noise to the data set. It stops data leakage.
Elastic Weight Consolidation (EWC) is used to mitigate the occasional forgetting by constraining the weights important for previous task.
Perplexity measures how well the model predicts the next word in text generation. It addresses relevance. Higher perplexity can result from underfitting due to insufficient capacity to capture data patterns, while too low perplexity might indicate overfitting to the training data.
Iterative Model Refinement: Uses human judgement and automated tools like Perplexity and BLEU to assure quality, relevance and safety.
BERT score is used to check if words have similar meanings, even if different words are used. Semantic similarity between reference & generated text.
Byte pair encoding helps in efficiently tokenizing text into subword units, which balances the vocabulary size and retains semantic information
Utilizing a replay buffer helps mitigate catastrophic forgetting by interleaving new data with original examples
Word Sense Disambiguation is designed to resolve ambiguities by determining the correct meaning of a word in each context
RMSprop maintains a moving average of the squares of gradients, allowing it to adjust the learning rate according to gradient variance.

Dynamic Masking Mechanism: Selectively masks or hides input data to improve model generalization.