Part 1

Evaluation, Benchmarking, and Integration

Introduction

On Day 3, we focus on evaluating Large Language Models (LLMs) and integrating them into workflow automation. Evaluating LLMs is essential for ensuring their performance, reliability, and safety. We also explore practical applications by integrating LLMs into workflows using APIs and automation tools like n8n. By the end of this session, you will have a solid understanding of LLM evaluation techniques and practical implementation strategies.

🧑‍🏫 👩🏿‍🏫 What You'll learn

Lesson 1: Evaluating Large Language Models (LLMs) - part 1

- Why evaluating LLMs is critical for performance, reliability, and safety.
- Common evaluation metrics like BLEU, ROUGE, and Perplexity.
- Different types of text evaluation: Content Overlap, Model-based, and Human Evaluation.
- The concept of adversarial testing and model robustness.

Lesson 2 : Evaluating Language Models (LLMs) - part 2

- Understand traditional and model-based evaluation metrics.
- Learn how human evaluation methods provide more reliable assessments.
- Explore adversarial testing strategies to identify model vulnerabilities.
- Develop critical thinking in choosing the right evaluation approach for different tasks.

Lesson 3: LLM Integration and Workflow Automation

- The importance of integrating LLMs into workflows for practical applications.
- How to connect to LLMs via APIs.

Lesson 4: Advanced LLM Workflow Integration with n8n (bonus)

- How to build a conversational AI agent with n8n.
- Implementing memory and context in LLM workflows.
- Utilizing various nodes for enhanced workflow capabilities.
- Debugging and optimizing complex LLM integrations.

💼 Prerequisites

- Basic understanding of natural language processing (NLP) concepts.
- Familiarity with text generation tasks.
- A willingness to engage with mathematical concepts (don't worry, we'll make it intuitive!).
- Familiarity with machine learning models such as BERT.

📖 Useful Resources

- Understanding BLEU and ROUGE score for NLP evaluation
- Understanding BLEU and ROUGE score for NLP evaluation

- LLM Evaluation metrics explained
- How to Conduct Human Evaluation in the Field of NLP?
- Adversarial Training to Improve Model Robustness
- BERTScore Explained in 5 minutes
- OpenAI API documentation | (or other relevant API documentation)
- Access the Inference API
- How to Use Hugging Face Inference API
- n8n documentation
- N8N Tutorial: Building N8N Ai Agents

Conclusion

Today, we explored the critical aspects of evaluating Large Language Models (LLMs) and integrating them into automated workflows. Understanding evaluation metrics, adversarial testing, and human evaluation helps ensure that LLMs function reliably in real-world applications. Furthermore, we introduced API-based integration and n8n automation techniques for building efficient and scalable AI-driven workflows.

By applying these concepts, you will be able to:

- Evaluate and improve LLM performance effectively.
- Design automated workflows leveraging LLM capabilities.
- Optimize AI-driven processes for various business and research applications.

Part 2

Evaluating Large Language Models

Table of Contents

Last Updated: July 21st, 2025

Evaluating Large Language Models (LLMs) - part 1

Introduction

In this lesson, we will explore the critical aspects of evaluating Large Language Models (LLMs). We'll cover the importance of evaluation, the challenges involved, and introduce various evaluation metrics and methods.

🧑‍🏫 👩🏿‍🏫 What You'll learn

- The significance of evaluating LLMs for performance, reliability, and safety.
- The complexities and challenges associated with LLM evaluation.
- An overview of different evaluation methods, including content overlap metrics, model-based metrics, human evaluation, and adversarial testing.
- In-depth understanding of `BLEU`, `ROUGE`, and Perplexity metrics.

💼 Prerequisites

- Basic understanding of natural language processing (NLP) concepts.
- Familiarity with text generation tasks.

📖 Useful Resources

- Understanding BLEU and ROUGE score for NLP evaluation
- Understanding BLEU and ROUGE score for NLP evaluation
- LLM Evaluation metrics explained

---

1. Introduction to LLM Evaluation

Why Evaluate Large Language Models (LLMs)?

Evaluating an LLM is essential to determine how well it performs and whether it meets the intended goals. Imagine you've developed a chatbot—how do you assess if it provides useful responses? Evaluation helps answer critical questions such as:

- **Performance:** Does the model generate accurate, relevant, and coherent responses? A well-performing LLM should provide factually correct answers, maintain contextual relevance, and follow logical reasoning.
- **Reliability:** Is the model consistent? A reliable LLM should provide stable responses to the same or similar prompts instead of fluctuating unpredictably.
- **Safety:** Does the model avoid generating harmful, misleading, or biased content? Evaluating safety ensures that the model does not reinforce stereotypes, spread misinformation, or produce toxic language.

Challenges in Evaluating LLMs

Unlike traditional software where performance is measured with fixed metrics like accuracy or speed, LLMs produce natural language output, making their evaluation more complex and subjective. Some key challenges include:

- **Variability in Output:** The same input prompt can generate multiple different yet valid responses. Unlike deterministic models, LLMs are probabilistic and can phrase responses differently while still being correct.
- **Subjectivity in Assessment:** The quality of a response depends on the application and the user's expectations. What might be a "good" answer for one use case (e.g., technical support) might not work well for another (e.g., creative writing).
- **Difficulty in Evaluating Complex Tasks:** Tasks like summarization, reasoning, or creative text generation do not have a single "correct" output. Evaluating them requires more sophisticated methods beyond simple correctness checks.

Overview of Evaluation Methods

To systematically assess LLM performance, various evaluation techniques are used, each addressing different aspects of model quality:

1. Content Overlap Metrics

These metrics compare the generated text against a reference text (e.g., a ground-truth answer).

- Common metrics include **BLEU** (for translation), **ROUGE** (for summarization), and **METEOR** (for alignment with human references).
- While useful for structured tasks, they struggle with assessing creativity, reasoning, and conversational responses.

2. Model-based Metrics

Instead of relying on exact word overlap, these methods use another AI model to judge the quality of the generated text.

- **BERTScore** computes similarity based on embeddings rather than word-matching.
- **GPTScore** and **Reward Models** are emerging approaches where a separate model ranks the generated text based on relevance or coherence.

3. Human Evaluation

Involves human annotators reviewing LLM outputs based on predefined criteria.

- Humans can assess aspects like **fluency, coherence, factual accuracy, and bias**, which automated metrics struggle with.
- While the most reliable, this method is costly and time-consuming.

4. Adversarial Testing

Actively probing the model to identify vulnerabilities, such as bias, factual errors, or susceptibility to misleading prompts.

- **Example:** Prompting an LLM with tricky or harmful questions to see if it generates misleading or toxic outputs.
- This helps in improving robustness by identifying weak spots that require further fine-tuning or safety measures.

By combining these evaluation techniques, we can form a comprehensive understanding of an LLM's strengths and weaknesses, ultimately leading to improvements in its performance and usability.

🚀 Challenge

Answer the following questions

- Why is evaluating LLMs more complex than evaluating traditional software?
- What are the key reasons for evaluating an LLM's safety?
- How does adversarial testing contribute to LLM improvement?

---

2. Content Overlap Metrics

Introduction to Content Overlap Metrics

Content Overlap Metrics evaluate the quality of machine-generated text by comparing it to a human-generated reference. These metrics are based on **n-gram overlap** — sequences of 1 or more consecutive words.

Why Use Content Overlap Metrics?

- Offer an **objective** way to assess similarity.
- Widely used in **translation, summarization, and generation** tasks.
- Fast and **computationally efficient.**

🔍 However, they **do not evaluate meaning**, synonyms, or flexible rephrasings — so they may misjudge quality in more creative or semantic tasks.

BLEU (Bilingual Evaluation Understudy)

What BLEU Measures

BLEU measures how many **n-grams** (word sequences) in the candidate text also appear in the reference. It's a **precision-based** metric, meaning it checks how much of the candidate is correct — not whether all reference content is captured.

BLEU Score Formula

$$\mathrm{BLEU} = \mathrm{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where:

- BP: **Brevity Penalty**
- w_n: weight for n-gram precision (typically equal, e.g. $w_n = \frac{1}{N}$)
- p_n: **Modified precision** for n-grams (explained below)
- N: Maximum n-gram length (commonly 4)

Step-by-Step BLEU Calculation Example

Reference: `"The cat sat on the mat"`
Candidate: `"The cat is on the mat"`

Let's calculate BLEU-2 (unigram and bigram):

1. Extract N-grams

Unigrams:

- Reference: the, cat, sat, on, the, mat
- Candidate: the, cat, is, on, the, mat

Bigrams:

- Reference: the cat, cat sat, sat on, on the, the mat
- Candidate: the cat, cat is, is on, on the, the mat

2. Count Matching N-grams

Unigram Precision ((p_1)):

- Matching unigrams: the, cat, on, the, mat → 5 matches
- Total unigrams in candidate: 6

$$p_1 = \frac{5}{6}$$

Bigram Precision ((p_2)):

- Matching bigrams: the cat, on the, the mat → 3 matches
- Total bigrams in candidate: 5

$$p_2 = \frac{3}{5}$$

3. Apply Modified Precision

Modified precision caps the count of each matching n-gram to the maximum number of times it appears in any reference sentence. In this example, we already applied this during matching.

4. Compute Brevity Penalty (BP)

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp\left(1 - \frac{r}{c}\right) & \text{if } c \leq r \end{cases}$$

Where:

- c = candidate length = 6
- r = reference length = 6
  So,

$$\text{BP} = 1$$

5. Combine with Weights

Assume equal weights: $w_1 = w_2 = 0.5$

$$\text{BLEU} = 1 \cdot \exp\left(0.5 \cdot \log\left(\frac{5}{6}\right) + 0.5 \cdot \log\left(\frac{3}{5}\right)\right)$$

$$\text{BLEU} \approx \exp\left(0.5 \cdot (-0.182) + 0.5 \cdot (-0.511)\right) = \exp(-0.347) \approx 0.707$$

> ✅ Final BLEU-2 Score ≈ **0.707** (or **70.7%** similarity)

How to implement it in Python ?

You can follow this colab

Begin by importing the required modules:

We import sqrt, log, and exp to perform fundamental mathematical operations—square-root extraction, natural logarithms, and exponentiation—and Counter to efficiently count and tally elements in iterable collections.

```
from math import sqrt, log, exp
from collections import Counter
```

We define the hypothesis and reference texts to establish the candidate output and its ground-truth comparisons, using examples drawn from David Bamman's NLP23 repository.

```
hypothesis="Abandon all hope , ye who enter here"
references=["All hope abandon , ye who enter here",
"All hope abandon , ye who enter in !",
"Leave every hope, ye that enter", "Leave all hope , ye that enter"]
```

Then, we have to get the n-grams from the given text. We'll define a function to get the frequency of n-grams from the given text of size "order".

```
#Getting the n-grams from the given text
def get_ngrams(text, order):
    """
    Given a string `text` and an integer `order`, returns a Counter object containing
    the frequency counts of all ngrams of size `order` in the string.
    """
    ngrams = Counter()

    words = text.split()
    for i in range(len(words)- order+1):
      ngram = " ". join(words[i: i + order])
      ngrams[ngram] += 1

    return ngrams
```

🚀 Challenge

Print the n-grams!

Now, let's calculate bleu, which is done in four steps as clearly indicated in the code block below:

```
def calculate_bleu(hypothesis, references):
    bleu=0
    p1=0
    p2=0
    p3=0
    p4=0
    bp=1

    # 1. Find the closest reference to the hypothesis
    closest_size=100000
    closest_ref=[]

    for ref in references:
      ref_size = len(ref)
      if abs(len(hypothesis) - ref_size) < closest_size:
        closest_size = abs(len(hypothesis) - ref_size)
        closest_ref = ref
        pass

    # 2. Calculating pn
    pns=[]
    for order in range(1,5):
      # calculate intersection and union of n-grams
      # hint: use the get_ngrams function you implemented
      # calculate pn for each order
        hyp_ngrams = get_ngrams(hypothesis, order)
        hyp_count = Counter(hyp_ngrams)
        closest_ref_ngrams = get_ngrams(closest_ref, order)
        closest_ref_count = Counter(closest_ref_ngrams)
        intersection_count = dict(hyp_count & closest_ref_count)
        intersection_size = sum(intersection_count.values())
        hyp_size = max(len(hyp_ngrams), 1)
        p_n = intersection_size / hyp_size
        pns.append(p_n)
        pass

    # 3. Calculating the brevity penalty
    bp=1
    c=len(hypothesis)
    r=min(abs(len(ref) - c) for ref in references)
    if c > r:
      bp = 1.0
    else:
      bp = exp(1 - r / c)

    # 4. Calculating the BLEU score
    weights = [0.25] * 4
    bleu=bp * exp(sum(w * log(p_n) for w, p_n in zip(weights, pns)))

    # Assigning values to p1, p2, p3, p4!
    p1, p2, p3, p4 = pns


    # Do not change the variable name
    return bleu, p1, p2, p3, p4, bp
```

Final step? Calculate and print the bleu score. The value should be `0.5<bleu<1`.

```
bleu, p1, p2, p3, p4, bp=calculate_bleu(hypothesis, references)
print("BLEU: %.3f" % bleu)
```

🚀 Challenge

What was the bleu value ? What does it indicate?

Summary: Strengths vs. Limitations

Strengths 💪	Limitations ⚠
Widely adopted in NLP	Ignores **meaning and synonyms**
Objective and reproducible	Penalizes **minor rephrasings**
Fast, simple to implement	Not ideal for **open-ended tasks** like summarization or dialogue

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE is a **recall-based** metric designed primarily for **summarization** tasks. Unlike BLEU, which measures **how much of the generated text matches the reference**, ROUGE measures **how much of the reference text is captured by the generated text**.

How ROUGE Works ?

1. It **counts matching n-grams** between the generated and reference texts.
2. It calculates **recall**—how much of the reference text is retained in the generated output.
3. It has multiple variants that capture different aspects of text similarity.

Example 1: Summarization Case

- **Reference Sentence:**

`"The quick brown fox jumps over the lazy dog."`

- **Generated Sentence:**

`"The brown fox jumps."`

Bigram Matching (2-grams):

Bigram	In Reference?	In Generated?	Match?
The quick	✅	❌	❌
Quick brown	✅	❌	❌
Brown fox	✅	✅	✅
Fox jumps	✅	✅	✅
Jumps over	✅	❌	❌
Over the	✅	❌	❌
The lazy	✅	❌	❌
Lazy dog	✅	❌	❌

- **Matched bigrams:** 2
- **Total bigrams in reference:** 8
- **ROUGE-2 recall:** 2 / 8 = 25%

Example 2: Article Summarization

Reference Text:
"The COVID-19 pandemic led to an economic recession and widespread lockdowns across the world."
Generated Summary:
"COVID-19 caused economic recession and lockdowns worldwide."

Here, ROUGE would measure how much of the **meaningful content** was retained.

Variants of ROUGE

- 🔹 **ROUGE-N:** Measures n-gram recall (similar to BLEU but recall-based).
- 🔹 **ROUGE-L:** Uses the **Longest Common Subsequence (LCS)** to compare sentence structures.

- 🔹 **ROUGE-W:** Weighted version of LCS that prioritizes consecutive matches.

Strengths of ROUGE

✓ **Great for summarization** tasks.
✓ **Considers recall**, making it useful for determining how much of the reference text is preserved.
✓ **More flexible than BLEU** since it is not overly sensitive to minor word changes.

Limitations of ROUGE

✕ **Still relies on word overlap**—it does not understand meaning.
✕ **Does not work well for very short or very long texts** (high recall may not mean good quality).
✕ **Does not consider synonyms or paraphrasing**—"big" and "large" would be treated as different.

Key Takeaways:

- **BLEU** is **precision-based** (measures how much of the generated text matches the reference).
- **ROUGE** is **recall-based** (measures how much of the reference text is preserved in the generated text).
- **BLEU is better for machine translation**, while **ROUGE is better for summarization**.
- Both metrics **struggle with capturing true meaning**, as they rely on exact word matches.
- Newer **semantic-based metrics** like BERTScore and GPTScore are improving evaluation techniques.

🚀 Challenge

Apply BLEU to this example

- **Reference:** "The cat sat on the mat."
- **Generated:** "The cat is on the mat."
- Identify bigrams and estimate the BLEU score.

Apply ROUGE to this example

- **Reference:** "The quick brown fox jumps over the lazy dog."
- **Generated:** "The brown fox jumps."
- Identify bigrams and estimate the ROUGE score.

How to implement it in Python : Computing ROUGE Scores in Python

We will follow a step-by-step tutorial for computing ROUGE-N and ROUGE-L metrics. You can also follow this colab

1. Imports and Example Texts

We'll reuse `Counter` for n-grams and implement our own LCS routine:

```
from collections import Counter

hypothesis = "Abandon all hope , ye who enter here"
references = [
    "All hope abandon , ye who enter here",
    "All hope abandon , ye who enter in !",
    "Leave every hope, ye that enter",
    "Leave all hope , ye that enter"
]
```

2. N-gram Extraction

```
def get_ngrams(text, order):
    """
    Returns a Counter of all n-grams of size `order` in `text`.
    """
    ngrams = Counter()
    words = text.split()
    for i in range(len(words) - order + 1):
        ngram = " ".join(words[i : i + order])
        ngrams[ngram] += 1
    return ngrams
```

🚀 Challenge

Print the 1- and 2-grams of the hypothesis!

3. ROUGE-N (Recall, Precision, F_1)

For each n (e.g., 1, 2), compute:

- **Recall** = overlap_ngrams / total_ngrams_in_reference
- **Precision** = overlap_ngrams / total_ngrams_in_hypothesis
- **F_1** = 2 × P × R / (P + R)

Average over references by taking the maximum overlap against any single reference.

```
def rouge_n(hyp, refs, n):
    """
    Compute ROUGE-N (recall, precision, f1) for one hypothesis vs. multiple references.
    """
    hyp_ngrams = get_ngrams(hyp, n)
    best = {"overlap": 0, "ref_count": 0}

    for ref in refs:
        ref_ngrams = get_ngrams(ref, n)
        overlap = sum((hyp_ngrams & ref_ngrams).values())
        if overlap > best["overlap"]:
            best["overlap"] = overlap
            best["ref_count"] = sum(ref_ngrams.values())

    hyp_count = sum(hyp_ngrams.values())
    recall    = best["overlap"] / best["ref_count"] if best["ref_count"] > 0 else 0.0
    precision = best["overlap"] / hyp_count         if hyp_count > 0        else 0.0
    f1 = (2 * precision * recall / (precision + recall)
          if (precision + recall) > 0 else 0.0)

    return recall, precision, f1
```

4. ROUGE-L (Longest Common Subsequence)

ROUGE-L uses the LCS length between hypothesis and reference:

- **Recall** = LCS / |reference|
- **Precision** = LCS / |hypothesis|
- **F_1** = $(1 + \beta^2) \cdot P \cdot R / (R + \beta^2 \cdot P)$, with $\beta = 1$ by default.

```
def _lcs_length(a, b):
    """Compute length of LCS between sequences a and b via dynamic programming."""
    m, n = len(a), len(b)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if a[i] == b[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j])
    return dp[m][n]
```

```
def rouge_l(hyp, refs, beta=1.0):
    """
    Compute ROUGE-L (recall, precision, f1) for one hypothesis vs. multiple references.
    Takes the reference yielding the highest F1.
    """
    best = {"f1": 0, "r": 0, "p": 0}
    hyp_tokens = hyp.split()

    for ref in refs:
        ref_tokens = ref.split()
        lcs = _lcs_length(hyp_tokens, ref_tokens)
        r = lcs / len(ref_tokens) if ref_tokens else 0.0
        p = lcs / len(hyp_tokens)   if hyp_tokens else 0.0
        denom = r + (beta**2) * p
        f1 = ((1 + beta**2) * p * r / denom) if denom > 0 else 0.0

        if f1 > best["f1"]:
            best.update({"f1": f1, "r": r, "p": p})

    return best["r"], best["p"], best["f1"]
```

5. Putting It All Together

Compute and print ROUGE-1, ROUGE-2, and ROUGE-L on our example:

```
# ROUGE-1
r1, p1, f1 = rouge_n(hypothesis, references, 1)
print(f"ROUGE-1 → recall: {r1:.3f}, precision: {p1:.3f}, F1: {f1:.3f}")

# ROUGE-2
r2, p2, f2 = rouge_n(hypothesis, references, 2)
print(f"ROUGE-2 → recall: {r2:.3f}, precision: {p2:.3f}, F1: {f2:.3f}")

# ROUGE-L
rl_r, rl_p, rl_f1 = rouge_l(hypothesis, references)
print(f"ROUGE-L → recall: {rl_r:.3f}, precision: {rl_p:.3f}, F1: {rl_f1:.3f}")
```

Expected output:

```
ROUGE-1 → recall: 0.857, precision: 0.857, F1: 0.857
ROUGE-2 → recall: 0.714, precision: 0.714, F1: 0.714
ROUGE-L → recall: 0.857, precision: 0.857, F1: 0.857
```

🚀 Challenge

What does it indicate?

---

3. Model-Based Metrics and Perplexity

Perplexity: Measuring Model Uncertainty

Perplexity is a metric that measures **how well a language model predicts a given text sample**. A **lower perplexity** indicates that the model is more confident in its predictions, meaning it assigns higher probabilities to the correct words.

Perplexity can be thought of as the **"uncertainty" of the model**. If a model is **very uncertain**, it assigns similar probabilities to many different words, leading to **high perplexity**. Conversely, if a model **accurately predicts** the next word, it assigns a high probability to the correct word and has **low perplexity**.

Example: Comparing Perplexity in Language Models
Suppose we have two language models predicting the next word in the sentence:

Sentence: `"The cat sat on the ____."`

Model	Probability of “mat”	Probability of “chair”	Perplexity
Model A	0.80	0.15	**Low (Better)**
Model B	0.40	0.35	**High (Worse)**

- **Model A has lower perplexity** because it assigns a high probability to the correct word (“mat”).
- **Model B has higher perplexity** because it is uncertain and distributes probability more evenly across words.

Perplexity is inversely related to the probability of the correct sequence of words.

Practical Interpretation of Perplexity

- **Perplexity = 1**: The model predicts the next word **with 100% certainty**.
- **Perplexity = 10**: The model is choosing among **10 likely words** for the next word.
- **Perplexity = 1000**: The model is extremely uncertain, considering **1000 possible words**.

Strengths of Perplexity

✓ **Intrinsic evaluation metric** that reflects a model’s internal performance.
✓ **Useful for comparing language models**—lower perplexity usually means better predictions.
✓ **Mathematically interpretable** and widely used in research.

Limitations of Perplexity

✕ **Does not measure real-world task performance** (e.g., fluency, factual accuracy).
✕ **Sensitive to tokenization and vocabulary size**—different models may have different tokenization schemes.
✕ **Not always correlated with human judgment**—a low-perplexity model may still generate ungrammatical or unnatural sentences.

🚀 Challenge

Compare perplexity scores

Model A assigns 0.8 probability to "mat," while Model B assigns 0.4.

Which model has lower perplexity? Why?

Conclusion

In this first part of our lesson on evaluating LLMs, we’ve laid the groundwork for understanding why evaluation is crucial and the challenges we face. We’ve explored the foundational concepts of content overlap metrics, particularly BLEU and ROUGE, and introduced the concept of Perplexity as a measure of model uncertainty.

We’ve learned that evaluating LLMs is not a straightforward task due to the inherent variability and subjectivity of natural language. Traditional metrics like BLEU and ROUGE offer objective measures of text similarity but have limitations in capturing semantic meaning and handling creative language use. Perplexity provides an intrinsic measure of a model’s predictive capability but doesn’t always correlate with real-world performance.

Moving forward, we’ll delve deeper into other evaluation methods, including model-based metrics, human evaluation, and adversarial testing, to gain a more comprehensive understanding of LLM performance. These diverse approaches will equip us with the tools necessary to assess and improve the quality and reliability of LLMs in various applications.

Remember that choosing the right evaluation metrics depends heavily on the specific task and goals of your LLM application.