Quality metrics in AI-Models' output: Simantic Similarity Score

Saeed Siddik
Assistant Professor
IIT University of Dhaka

Where Semantic Quality Fits in AI Projects

- Content-generation tasks in products (chatbots, summarizers, recommender systems)
- Need for meaning-preserving outputs

How Semantic Similarity Works (Non-technical explanation)

- Measure undrlying meaning
- Idea of text embeddings
- Meaning represented as vectors

BERT

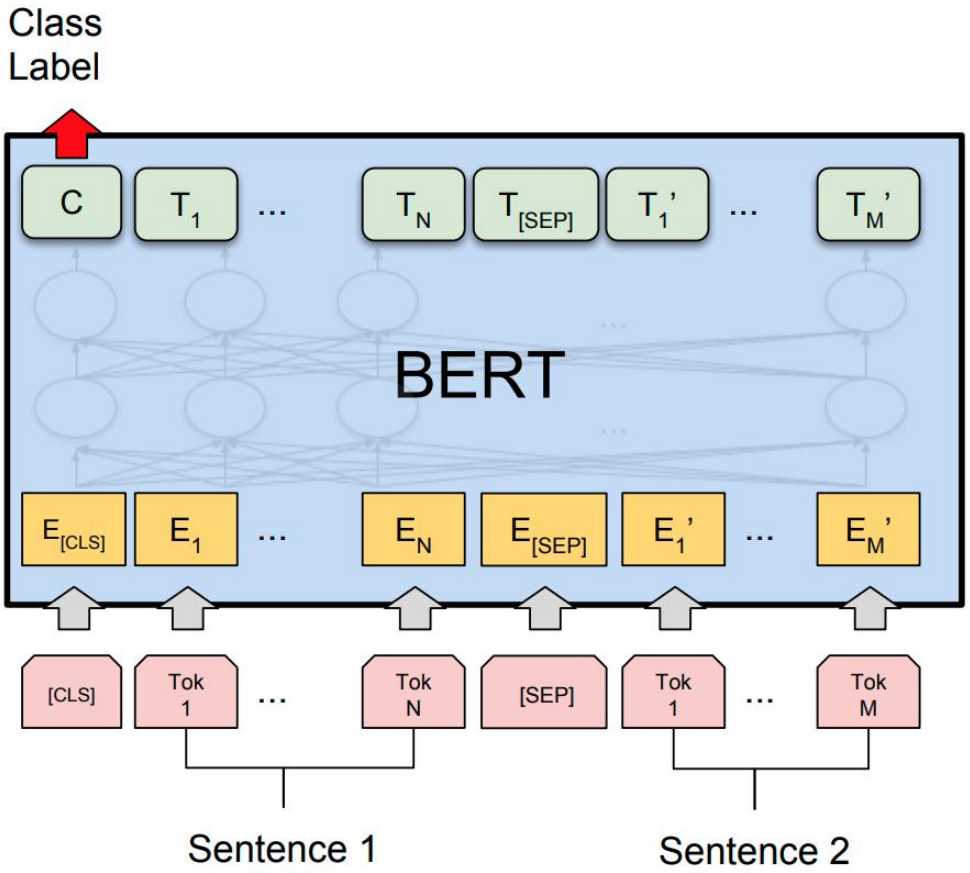
- Bidirectional Encoder Representations from Transformers
- published by researchers at Google AI Language.
- applying the bidirectional training of Transformer, a popular attention model to language modelling

Bidirectional and Attention model

- Bidirectional:
  - reads the text in both directions at the same time, left-to-right and right-to-left to understand full context.
  - This gives much better understanding
- Attention model:
  - is the mechanism that allows the system to look at all the words in a sentence at the same time and decide which words are most important for understanding each other.
  - It can weigh words differently depending on their importance, capturing long-range dependencies efficiently

Alternatives

- For Bidirectional Training:
  - Unidirectional models: GPT (Generative Pretrained Transformer) reads left-to-right only. It works well for generation tasks but may miss full sentence context for understanding.
- Attention Model
  - Shallow RNN / LSTM without attention: Can process sequences, but they treat all previous words equally or rely on hidden states, which may forget important information.
  - CNNs for text: Can detect local patterns but do not capture long-range dependencies effectively.


Class
Label
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok
1
...
Tok
N
[SEP]
Tok
1
...
Tok
M
Sentence 1
Sentence 2

High-Level Example

- Sentence: “`The movie was fantastic.`”
- Tokenize: BERT splits the sentence:

`[CLS] | the | movie | was | fantastic | [SEP]`

- [CLS] represents the whole sentence,
- [SEP] marks the end of the sentence.
- BERT converts each token into something like this:

Each token becomes a vector

- BERT converts each token into something like this:
  - [CLS] → [0.12, 0.68, −0.24, …]
  - the → [−0.03, 0.44, 0.10, …]
  - movie → [0.56, 0.90, −0.12, …]
  - was → [0.11, 0.03, 0.02, …]
  - fantastic → [0.92, 0.33, 0.78, …]
- Each list = a vector that represents meaning.
- Think of them as:
  - “movie” → a meaning-vector about films
  - “fantastic” → a meaning-vector about positivity

BERT mixes vectors and self-attention

- BERT looks at how the meaning-vectors relate:
- “fantastic” vector is strongly linked to “movie” vector
- “was” connects them
- “the” contributes little

Final [CLS] vector

- After processing:
  - [CLS] → [1.22, −0.44, 0.89, …]
- This final [CLS] vector represents the meaning of the whole sentence.
- If the task is sentiment analysis, this vector tells BERT:
  - This sentence is positive.

Types and Alternatives

- `RoBERTa & T5` → Higher accuracy but heavier computational cost.
- `DistilBERT & ALBERT` → Faster and lighter but may lose some nuance.
- `ELECTRA` → Efficient training but more complex setup.
- `GPT models` → Excellent for generation but expensive to fine-tune.

Using Type-4 Code Clone

- Type-4 clones are semantic or functional clones, meaning the code performs the same functionality but may have different implementations, variable names, or structure.
- Unlike Type-1 (exact) or Type-2 (renamed variables) or Type-3 (small edits), Type-4 focuses on behavioral equivalence rather than textual similarity.
- We can use Type-4 clone detection to identify whether different outputs are functionally equivalent, even if the code looks different.

Type-4 Code Clone Example

Initial Code Fragement CF_0

```
for(i = 0; i < 10; i++)
{
    // foo 2
    if (i % 2 == 0)
        a = b + i;
    else
        // foo 1
        a = b - i;
}
```

CF_1 – Type-1 Clone

```
for(i = 0; i < 10; i++)
{
    if (i % 2 == 0)
      a = b + i; //cmt 1
    else
      b = b - i; //cmt 2
}
```

CF_4 – Type-4 Clone

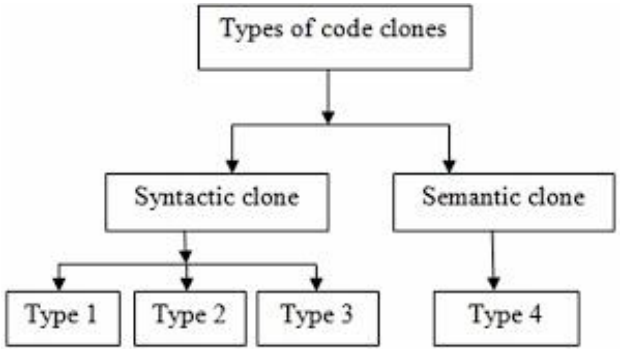
```
while(i < 10)
{
    // a comment
  a = (i % 2 == 0) ?
  b + i : b - i;
  i++;
}
```

CF_2 – Type-2 Clone

```
for(j = 0; j < 10; j++)
{
    if (j % 2 == 0)
      y = x + j; //cmt 1
    else
      y = x - j; //cmt 2
}
```

CF_3 – Type-3 Clone

```
for(i = 0; i < 10; i++)
{
    // new statement
    a = 10 * b;
    if (i % 2 == 0)
      a = b + i; //cmt 1
    else
      a = b - i; //cmt 2
}
```

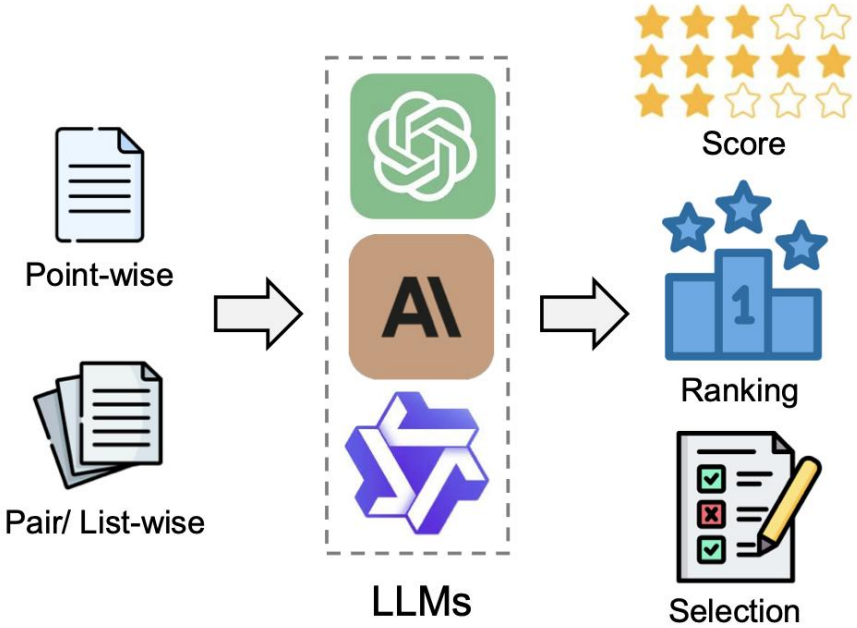


Source: https://www.researchgate.net/figure/Examples-of-various-types-of-code-clones-proposed-in-the-literature-22_fig1_371943883

LLMs as a Judge

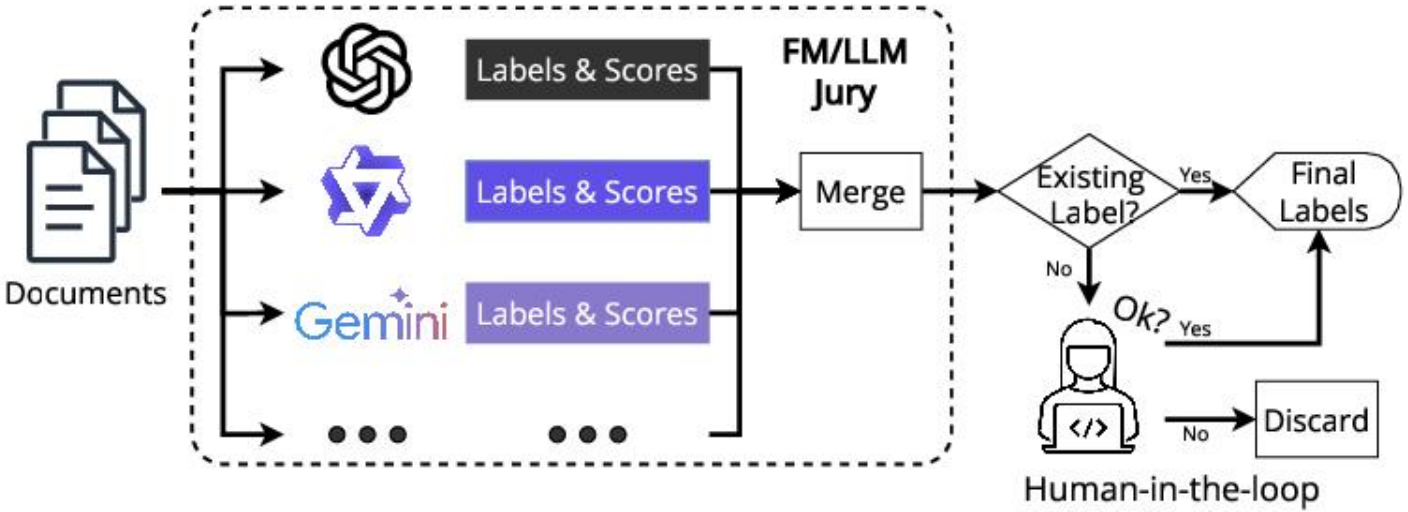
- LLM-as-a-Judge uses LLMs to evaluate AI-generated texts based on custom criteria defined in an evaluation prompt.
- Treat an AI-Model as an automated evaluator that judges whether outputs (code or text) meet certain quality criteria.
- A prompt asking it to judge: e.g., “Does this candidate produce the same functionality or meaning as the reference? Rate or explain.”
- Ask a Jury of LLM as a Judge Pool, and take majority voting. If there is a tie, a manual interruption is needed as human-in-the-loop to break the system and proceed next.

Overview of I/O of LLM-as-a-judge



Source: https://aclanthology.org/2025.emnlp-main.138.pdf

An overview of LLM Jury

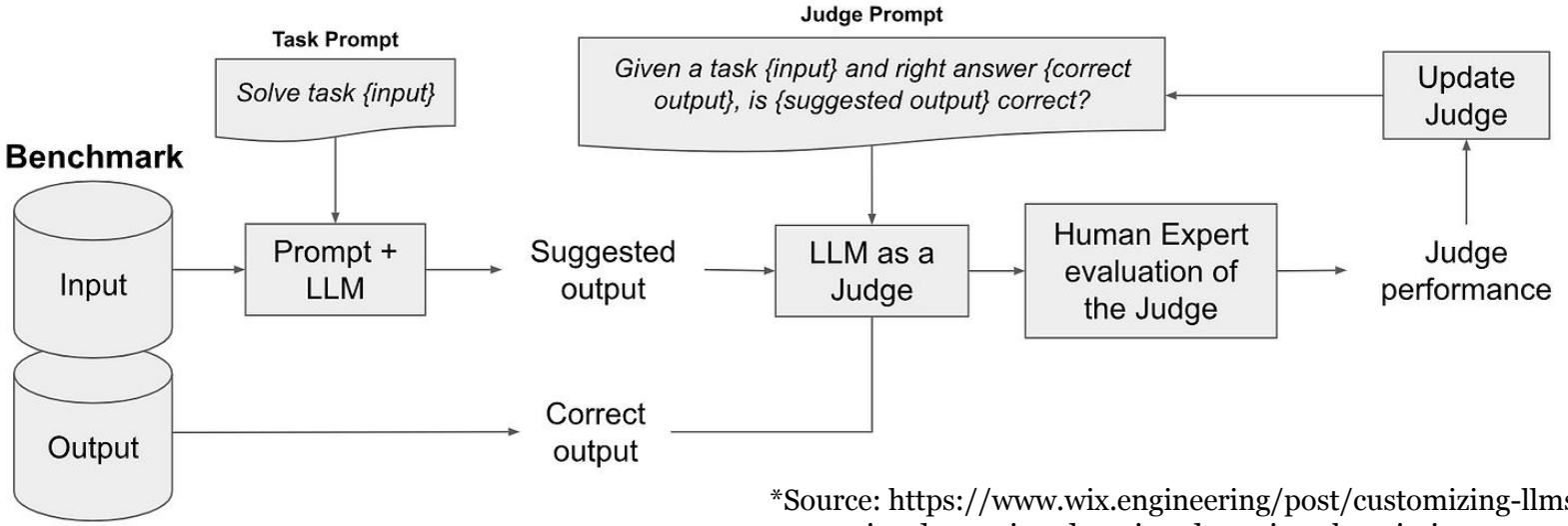


*Source: https://arxiv.org/pdf/2410.09012

Types of LLM judges

- **Pair wise comparison**: give LLM two responses and ask to choose the better one. This lets you compare models, prompts, or configurations to see which performs best.
- **Evaluation by criteria (reference-free)**: ask the LLM to assess a response or conversation based on tone, clarity, correctness, or other dimensions.
- **Evaluation by criteria (reference-based)**: provide extra context, like a source document or reference, and ask the LLM to score the response.

Development process for LLM-as-a-Judge



*Source: https://www.wix.engineering/post/customizing-llms-for-enterprise-data-using-domain-adaptation-the-wix-journey

Limitations of LLM-as-a-Judge: Biases

- **Position Bias**: An LLM may prefer a response simply because it appears earlier or later in a sequence.
- **Verbosity Bias**: It may unfairly equate longer responses with higher quality, regardless of content.
- **Authority Bias**: The LLM can be unduly influenced by references to authoritative sources rather than evaluating objectively.
- **Diversity Bias**: Judgments can be skewed by identity-related markers like gender or ethnicity.
- **Self-enhancement Bias**: An LLM judge may favor outputs from the same model or family of models, leading to a lack of impartiality.

Other Limitations of LLM-as-a-Judge

- **Difficulty with complex tasks**: Evaluating tasks that require complex reasoning, creative thinking, or practical expertise remains challenging for LLMs.
- **Lack of knowledge and context**: An LLM may lack the specific or proprietary context needed for accurate judgment.
- **Misleading confidence**: They can be "fooled" by answers that sound convincing but are incorrect, giving a false sense of confidence in the quality of an output.
- **Need for human oversight**: Because of these limitations, using LLM-as-a-judge requires a human to evaluate the judge itself and provide ground truth, especially for complex or critical tasks.

End of **Simantic Similarity** Score