Summarization

Srikanth Tamilselvam
IBM Research, India

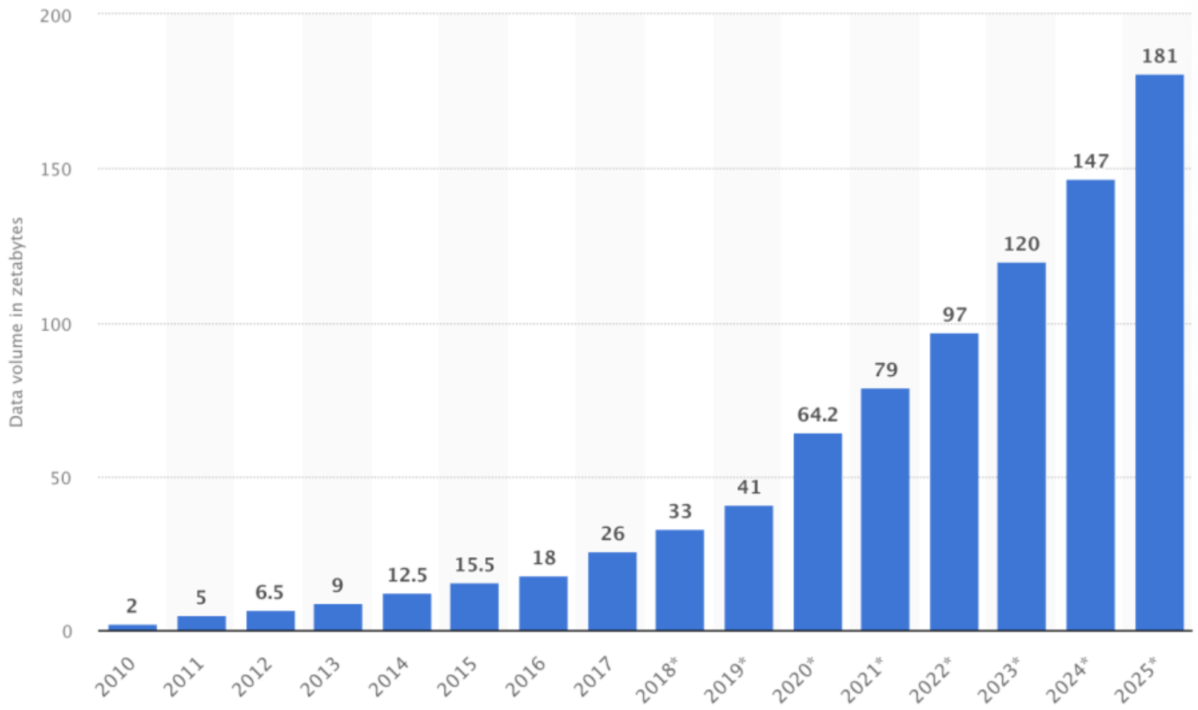
Agenda

- Motivation
- Text Summarization
- Code Representations
- Code Summarization
- Research Opportunities

What is summarization ?

- Extracting juice from Fruit. You keep the important parts and discard the pulp.
- In general, summarization is the process of **reducing large amounts of information** into a **shorter, concise form** while **preserving the core meaning** and **essential detail**s.
- It’s not just limited to NLP or text — summarization happens in daily life.

Data is growing



Source: [Data growth worldwide 2010-2025 | Statista]

In 2020, the amount of digital data was 64.2 zettabytes

In 2025, the expected amount of digital data is 180 zettabytes

In 2028, the expected amount of digital data is 400 zettabytes

Fun fact - A zettabyte equals 1 sextillion bytes (1,000,000,000,000,000,000,000 bytes) ☺

90% Of The Data Worldwide Is Unstructured !!
Source : [Research World]

Types of Summarization in a Broader Context

Text Summarization
Condensing articles, books, or reports

Audio Summarization
Creating transcripts or meeting summaries

Video Summarization
Producing highlights or short recaps

Data Summarization
Reducing large datasets into key metrics or charts

Code Summarization
Generating descriptions of functions, APIs, of classes

Multimodal Summarization
Combining text, images, video, and audio into unified summaries

Conversational Summarization
Summarizing chat logs, discussions, or customer support calls

Text Summarization

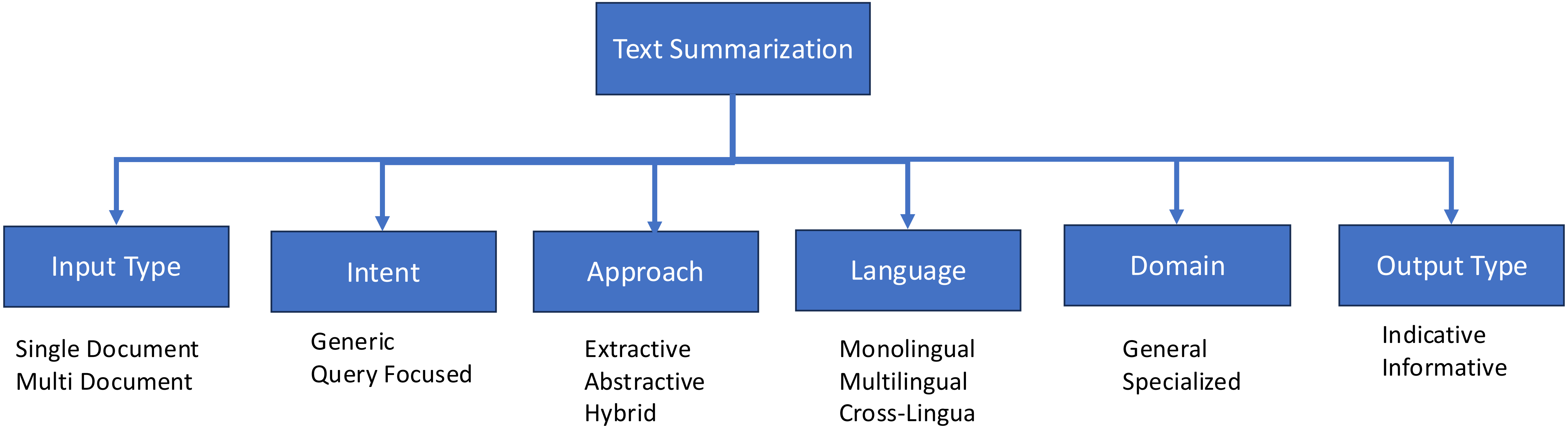
Natural language processing (NLP) is technology that allows computers to interpret, manipulate, and comprehend human language. Organizations today have large volumes of voice and text data from various communication channels like emails, text messages, social media newsfeeds, video, audio, and more. Natural language processing is key in analyzing this data for actionable business insights. Organizations can classify, sort, filter, and understand the intent or sentiment hidden in language data. Natural language processing is a key feature of AI-powered automation and supports real-time machine-human communication.

84 words

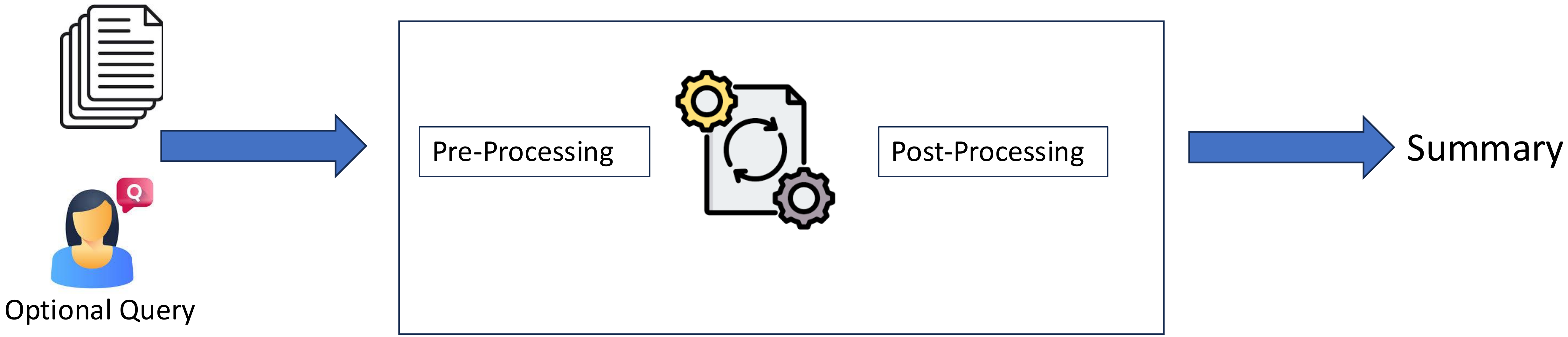
NLP enables computers to understand and analyze human language, helping organizations extract insights, detect intent or sentiment, and support AI-driven automation and real-time communication.

24 words

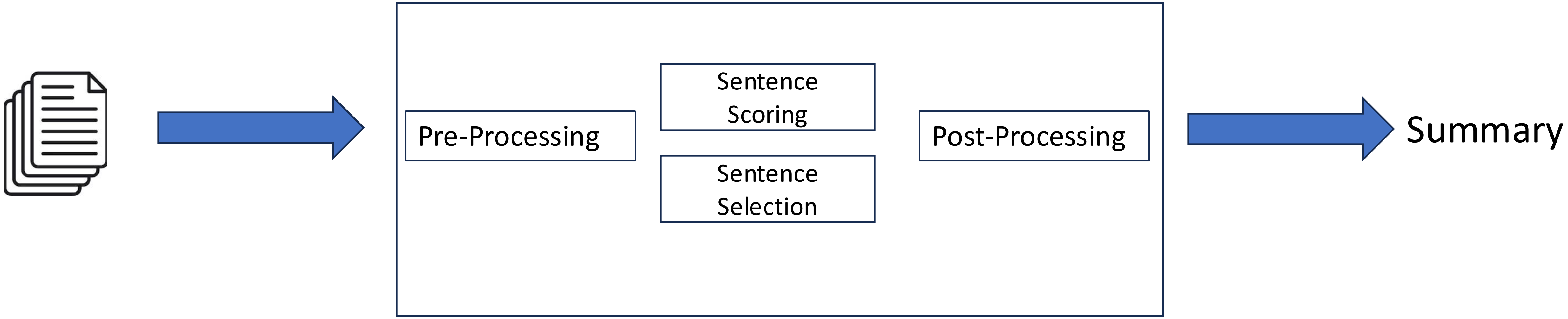
Classification of Text Summarization System



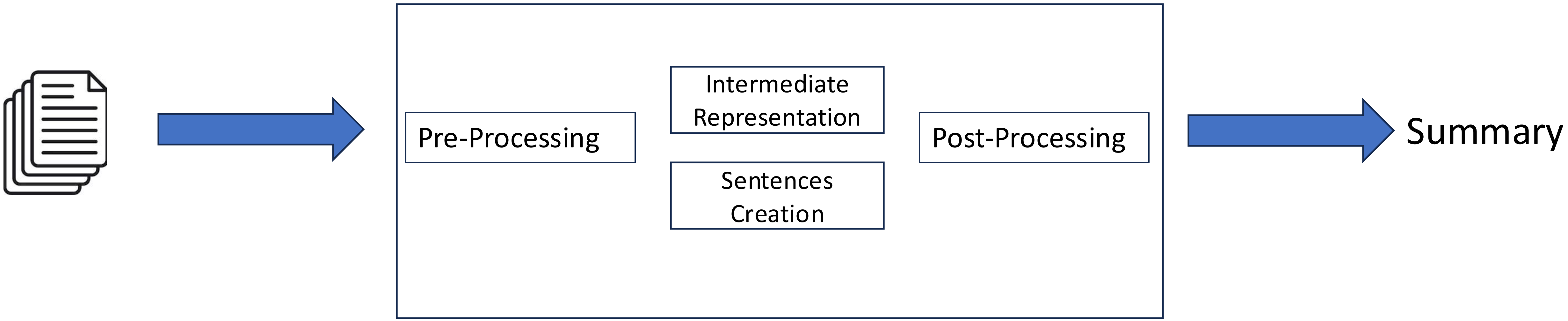
Typical Text Summarization System



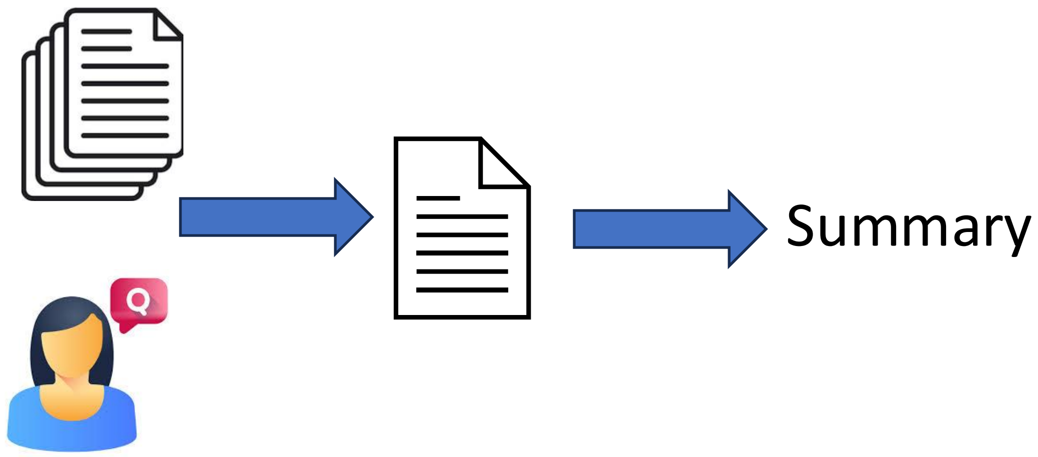
Typical Extractive Summarization System

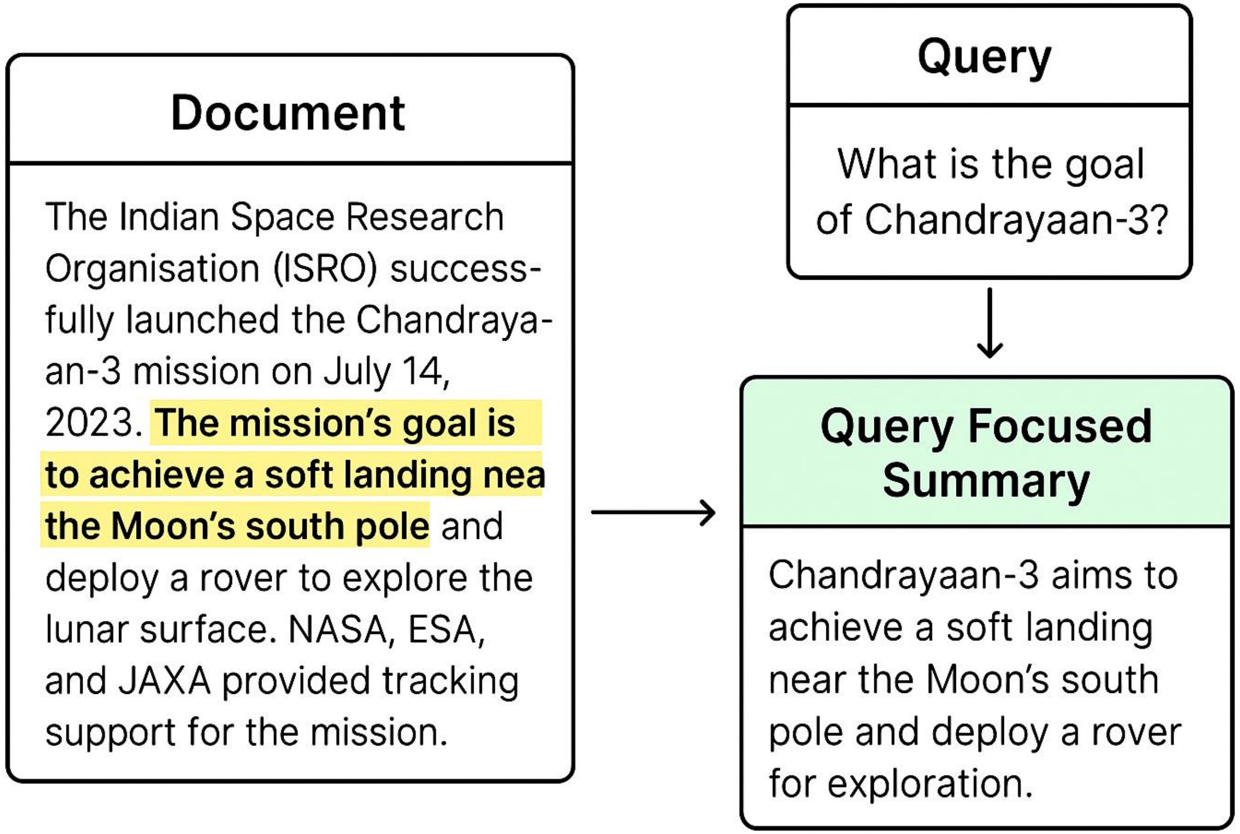


Typical Abstractive Summarization System

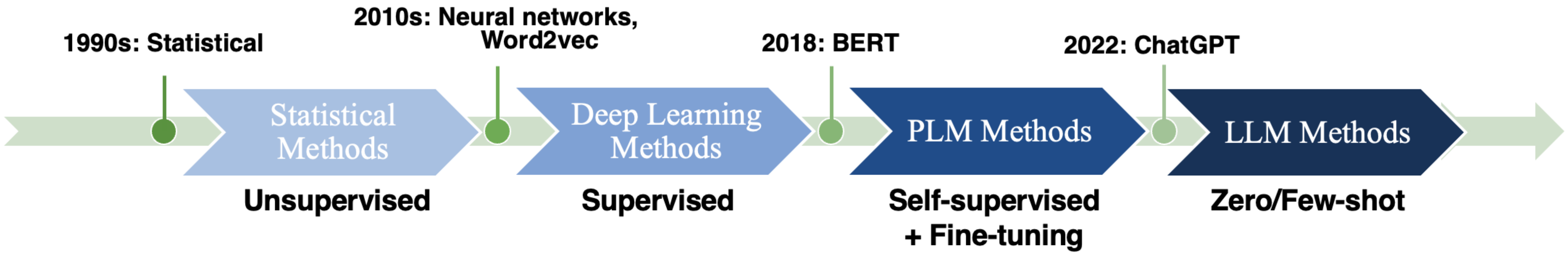


Query Focused Summarization



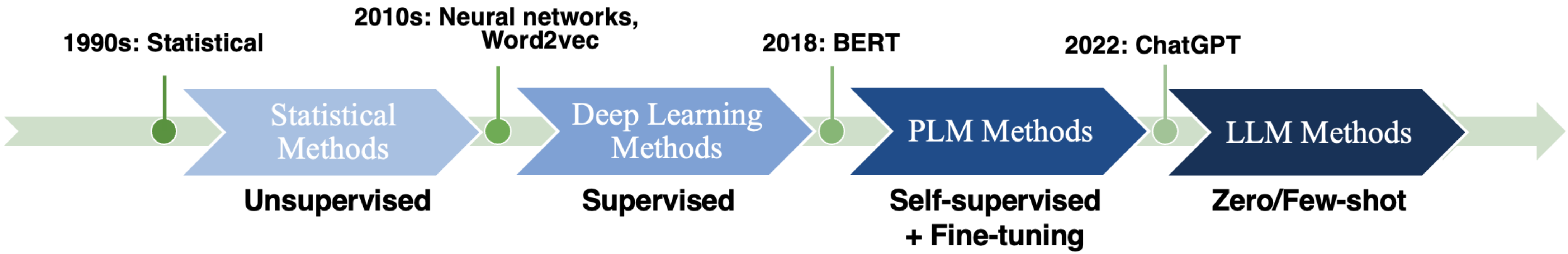


Text summarization research Evolution



Source [: A Systematic Survey of Text Summarization: From Statistical Methods to Large Language Models, Zhang et.al]

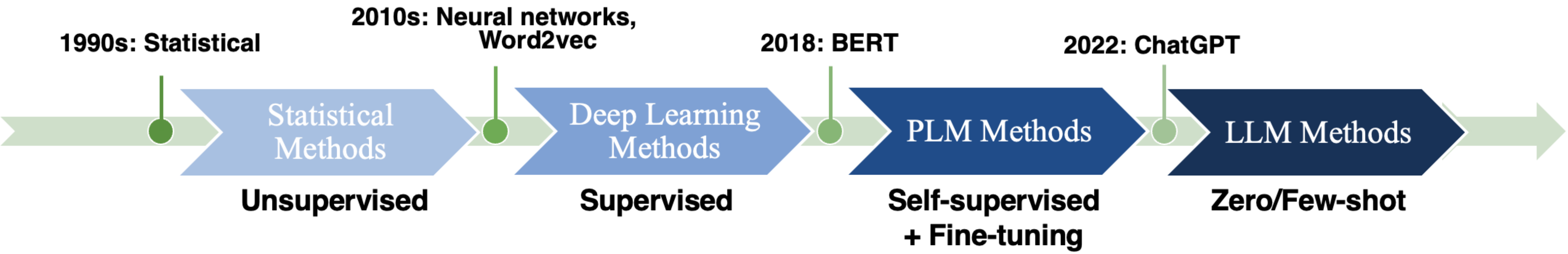
Text summarization research Evolution



Heuristic-based, Carbonell et.al
Optimization-based, Lin et.al
Graph-based Erkan et.al

- Relied heavily on handcrafted features (word frequency, sentence position, cue words).
- No deep semantic understanding of the text.

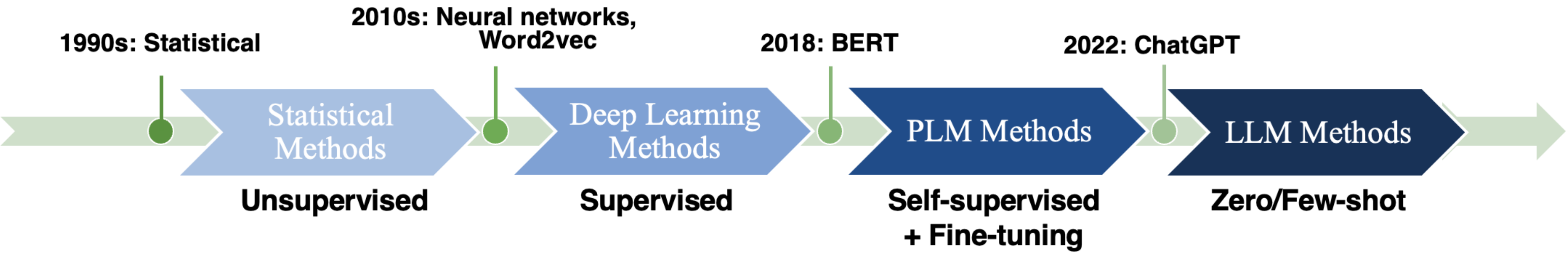
Text summarization research Evolution



Attention Based LSTM extractor, [Cheng et.al]
Sentence-level RNN + Feature fusion (SummaRuNNer), [Nallapati et.al]
LSTM + reinforcement learning, [Narayan et.al]

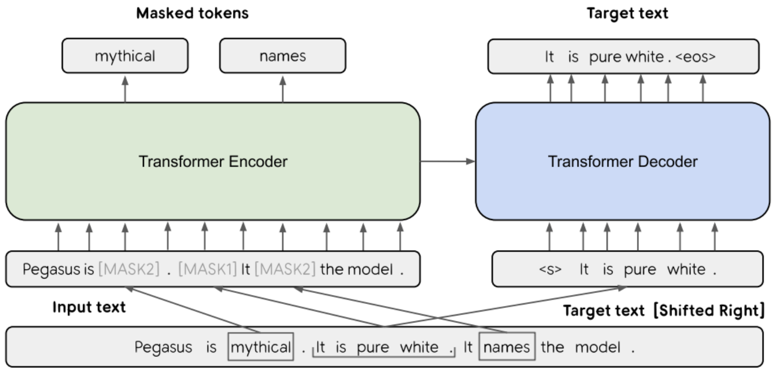
- Required parallel datasets of documents and summaries.
- Improved fluency compared to feature-based methods but struggled with long-context

Text summarization research Evolution

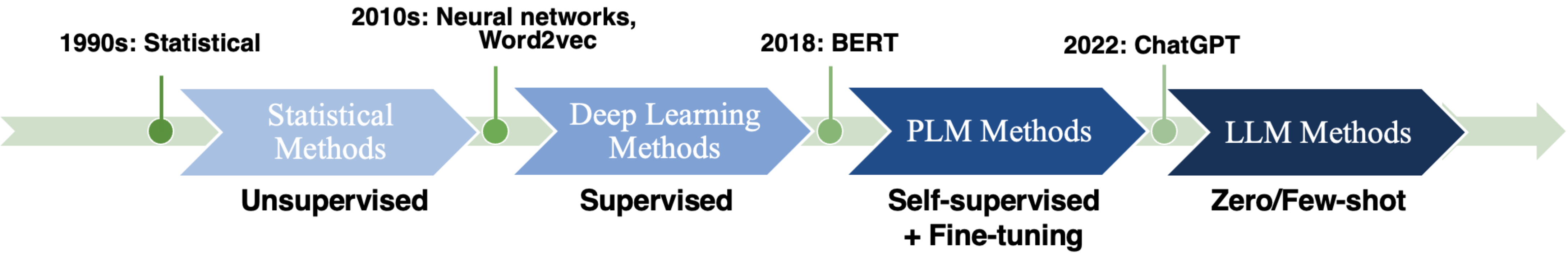


BERTSUMEXT, Liu et.al

PEGASUS, Zhang et.al



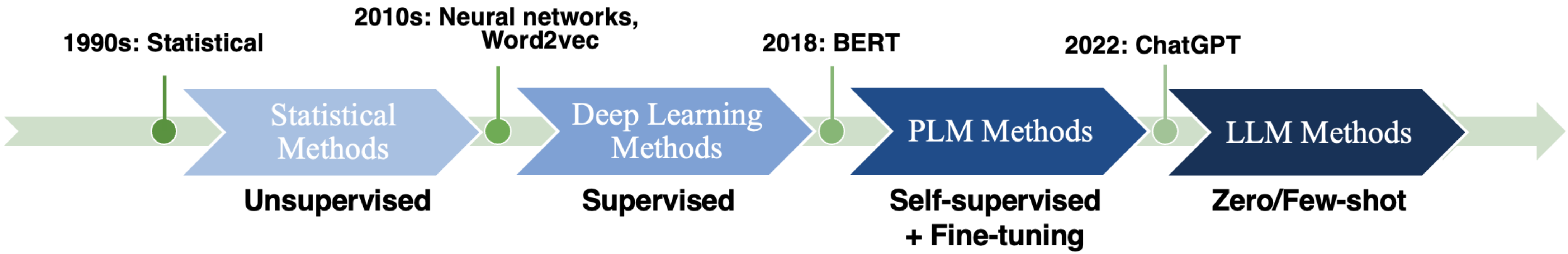
Text summarization research Evolution



BERTSUMEXT, Liu et.al
PEGASUS, Zhang et.al

- Captured rich contextual representations.
- Reduced dependence on large task-specific labelled data.
- Enabled better handling of diverse domains and abstractive summarization.

Text summarization research Evolution



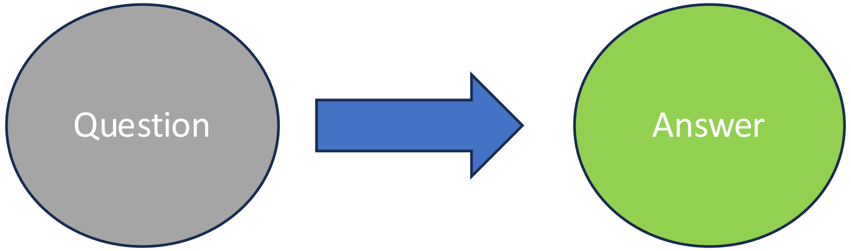
ICL, Zhang et.al
Tiny LLMs, Fu et.al
Style Focused, Liu et.al
CoT, Wang et.al
Multi-Agents, Zhang et.al
…

- Require minimal task-specific supervision.
- High adaptability to diverse summarization styles and domains via prompting

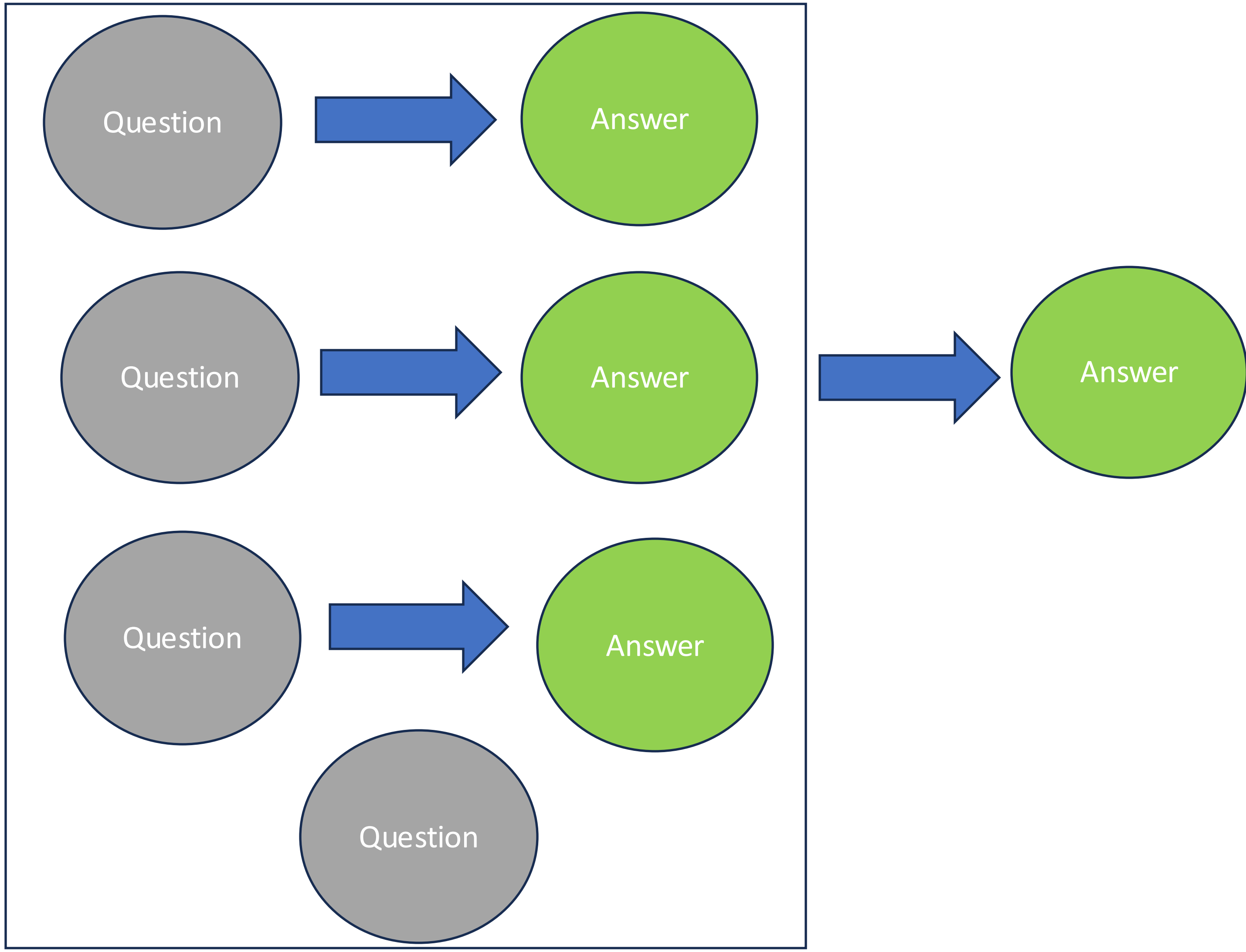
Recent Modelling approaches with LLMs

- ❑Prompting Based
- ❑Multi-Agent Based
- ❑Distillation Based
- ❑Other Innovations

Few Shot Prompting



Zero Shot Prompting



3 Shot Prompting

Prompt Chaining

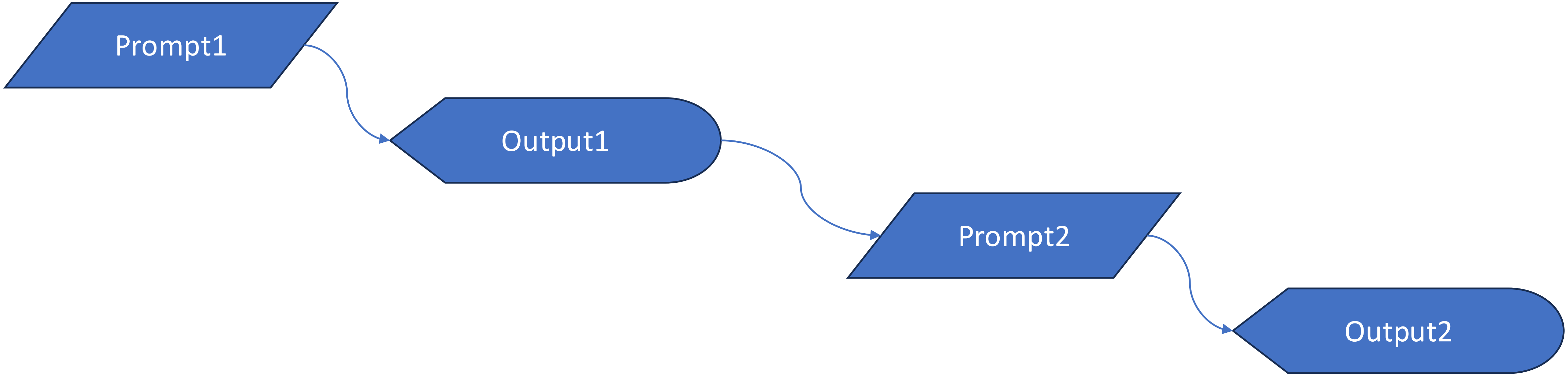


Figure 1 : Prompt Chaining

Prompt Chaining Vs Stepwise Prompt

Break task into multiple steps with each step involves a **separate prompt**

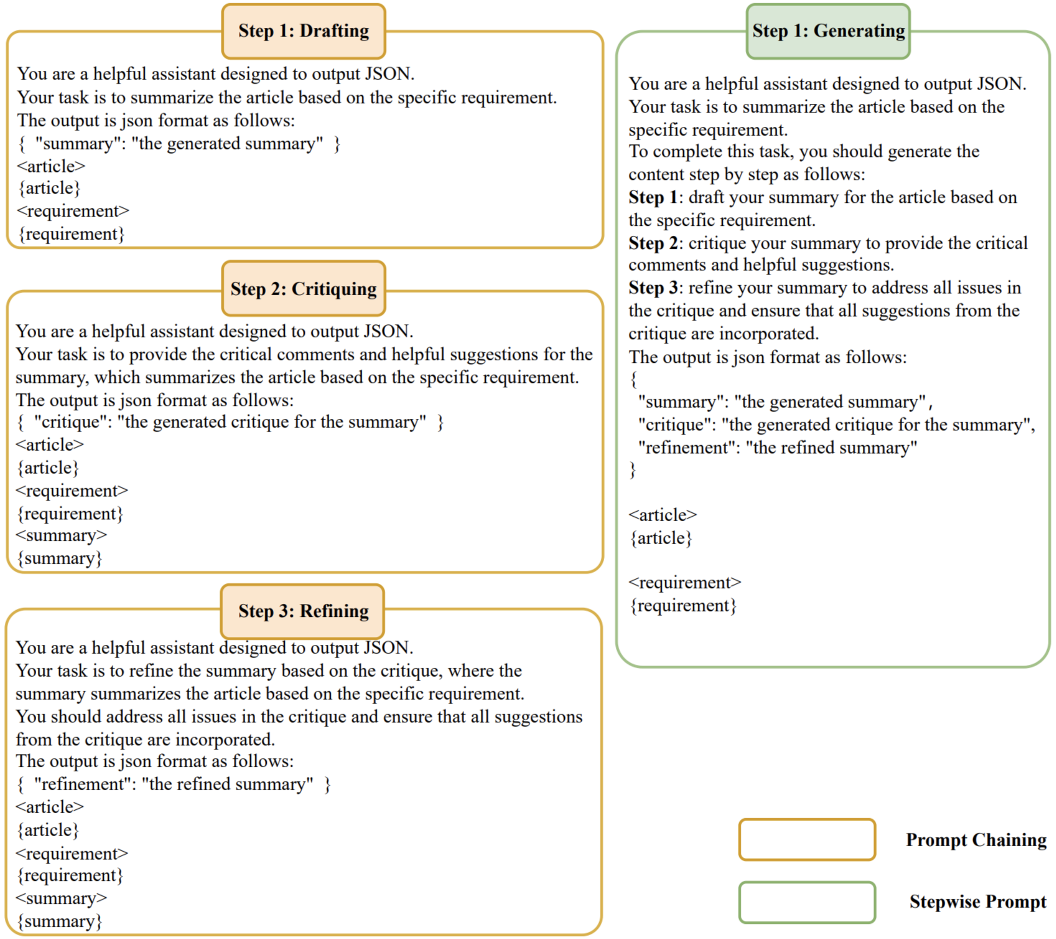


Figure 1: Prompt Chaining v.s. Stepwise Prompt.

“Think step by step” within **one prompt**

Source : Element-aware Summarization with Large Language Models: Expert-aligned Evaluation and Chain-of-Thought Method, Sun et.al

More Prompting Examples

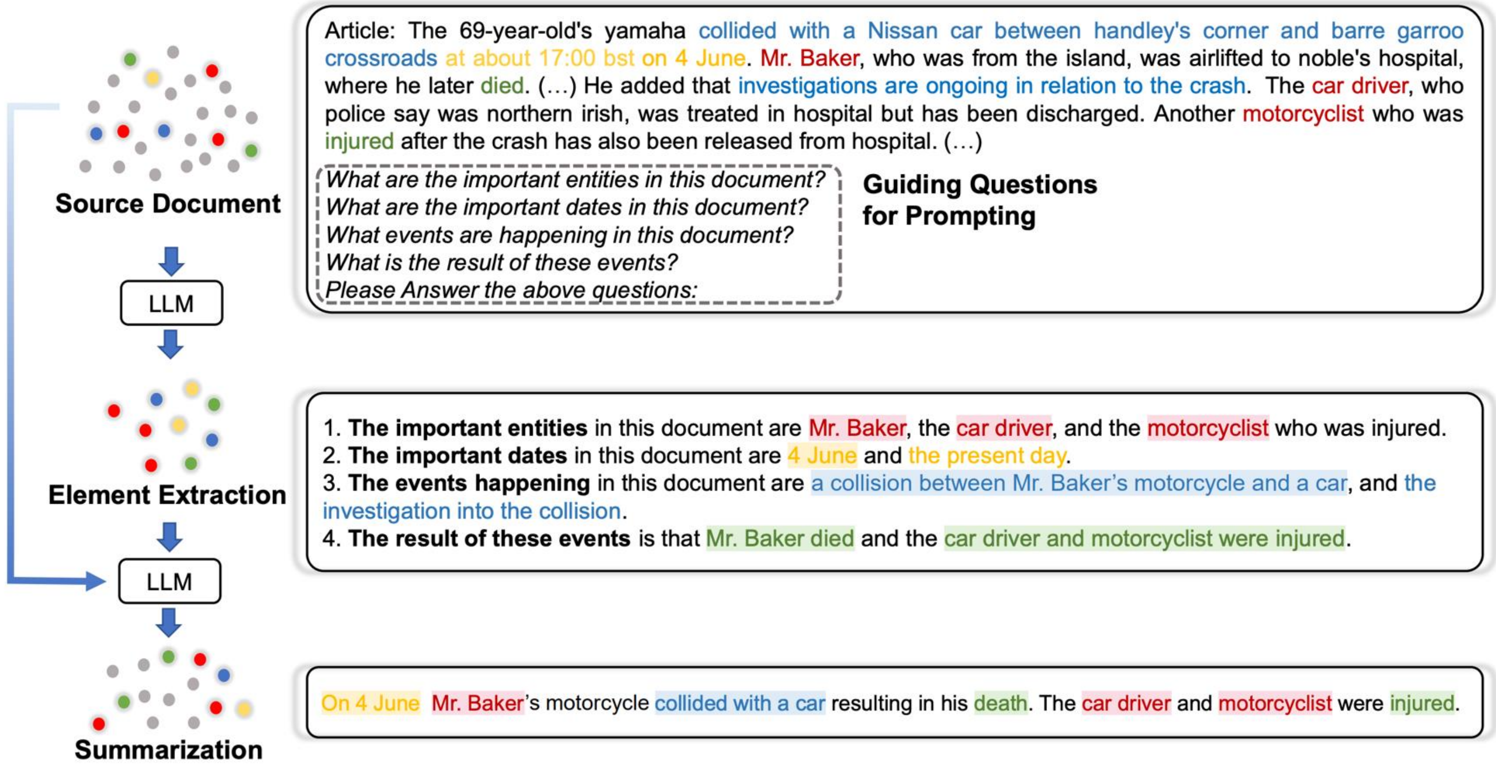
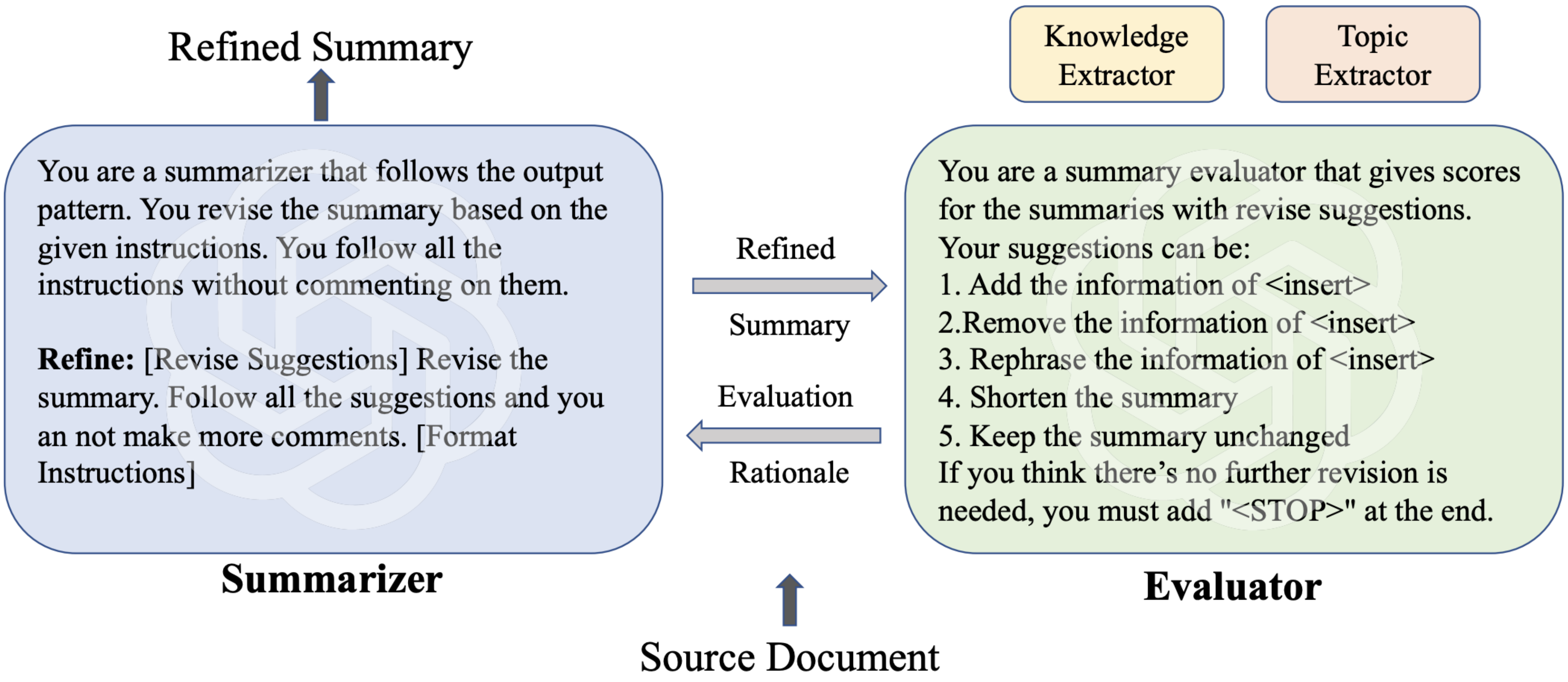


Figure 1 : Summary Chain-of-Thought **(SumCoT)** methodology

Source : Element-aware Summarization with Large Language Models: Expert-aligned Evaluation and Chain-of-Thought Method, Wang et.al

Multi-Agents



Source : SummIt: Iterative Text Summarization via ChatGPT, Zhang et.al

Instruction Fine-Tuning

Item Name: *"Blade Tail Rotor Hub Set B450 330X Fusion 270 BLH1669 Replacement Helicopter Parts"*

- `Summarize {Item_Name} to contain at most 3 words` → *"Blade Rotor Hub"*
- `Summarize {Item_Name} with Low specificity and to contain the words "B450 330X"` → *"Rotor Hub Set B450 330X"*
- `Summarize {Item_Name} with Low specificity` → *"Rotor Hub Set"*

Figure 1 : Product title summaries generated through instruction tuning for different instructions.

Source : InstructPTS: Instruction-Tuning LLMs for Product Title Summarization, Fetahu et.al

Knowledge Distillation

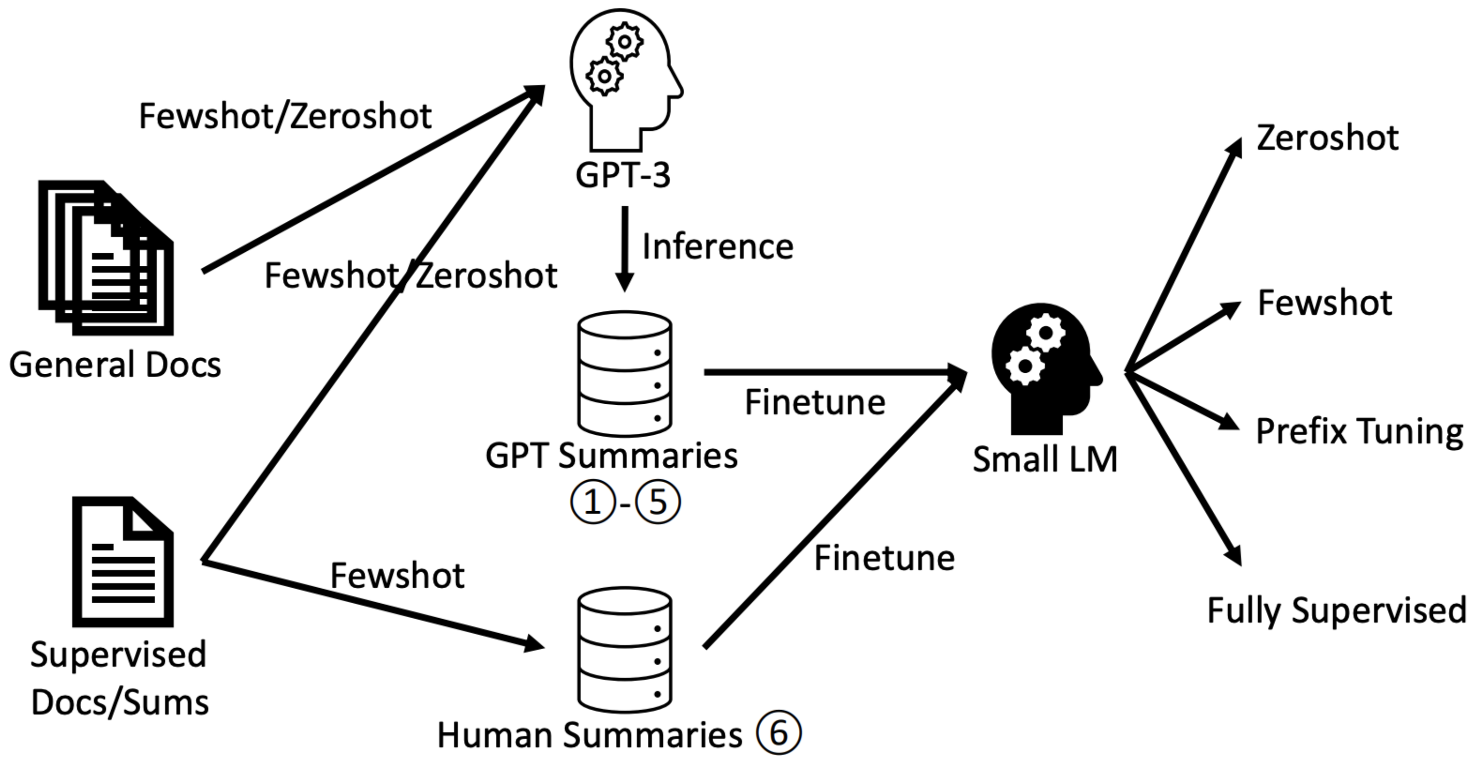


Figure 1 : Knowledge Distillation (transferring knowledge from a **big teacher** → **small student**)

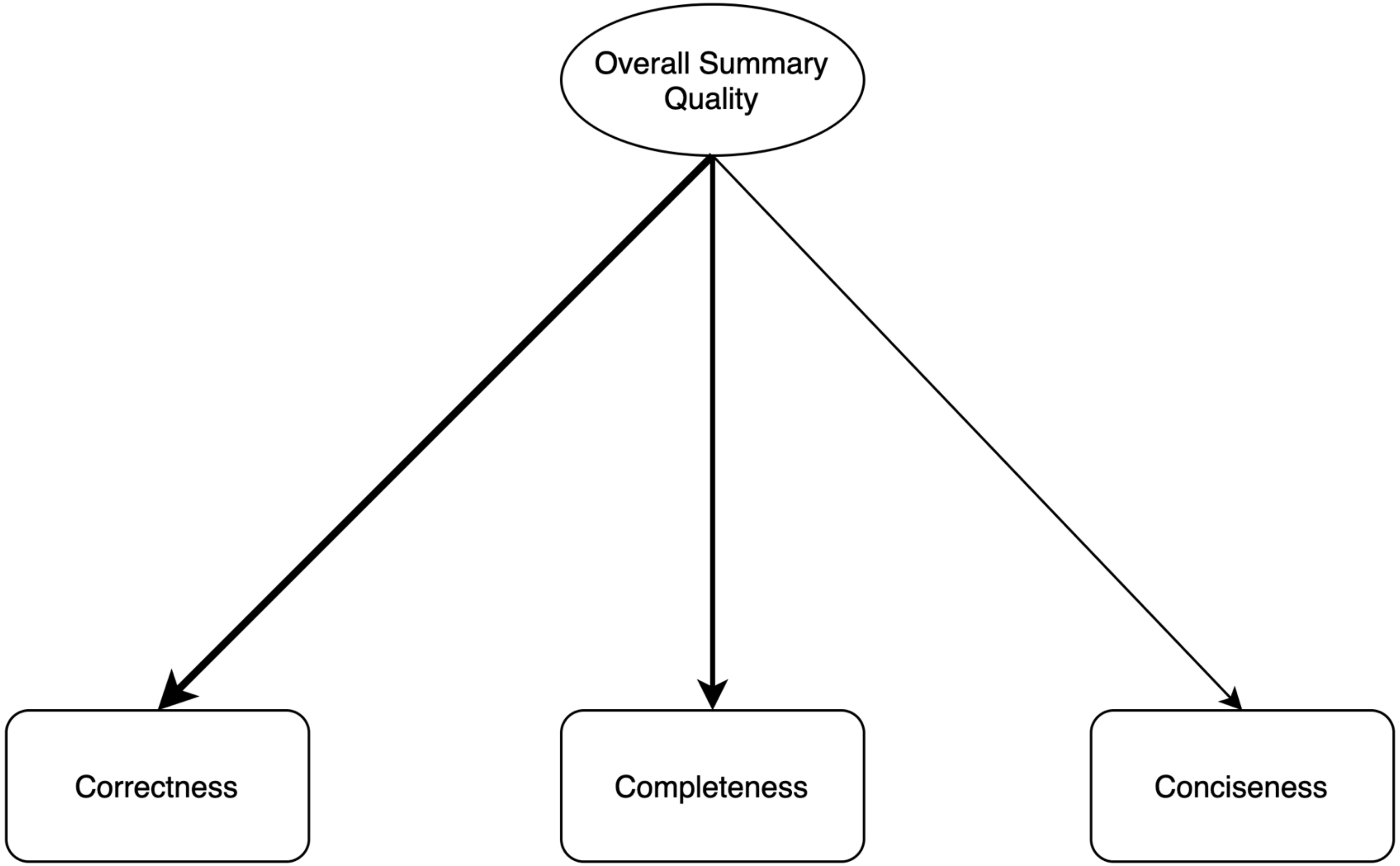
Source : InheritSumm: A General, Versatile and Compact Summarizer by Distilling from GPT, [Xu et.al]

Common Summarization Datasets

Year	Task/Domain	Dataset
2022	News	CNN/DM, XSum, Newsroom
2023	News	CNN/DM, XSum
2023	News	CNN/DM, Multi-News, Mediasum
2023	Extractive	CNN/DM, XSum, Reddit, PubMed
2023	Meeting	AMI, ICSI, QMSUM
2024	Meeting	In-Domain, QMSUM
2023	Multilingual	CLS
2023	Multi-doc	DIVERSESUMM
2023	QFS	CovidET, NEWTS, QMSum, SQuALITY
2023	QFS	ELIFE
2023	Controllable	INSTRUSUM
2023	Factuality	CNN/DM, XSum
2023	Factuality	AggreFact, DialSummEva
2023	Factuality, dialogue	MediaSum, MeetingBank
2023	Fairness	Claritin, US Election, Yelp, etc.
2023	Position bias	MiddleSum
2024	Position bias	CNN/DM, XSum, News, Reddit
2023	Medical	Alzheimer, Kidney, Skin, etc.
2023	Medical	RCT
2023	Medical	ProbSum, MeQSum, ACI-Bench, etc.
2023	Code	BinSum
2023	Code	CSN-Python
2024	Code	CodeXGLUE

Source : A Systematic Survey of Text Summarization: From Statistical Methods to Large Language Models, Zhang et.al

3C's For Summary Quality



All details are accurate?

All Important details captured?

Only Important details captured?

Key Metrics

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)** → Measures overlap between reference summary and generated text using n-grams or sub-sequences. Focus is on recall.

  Example:

  - Reference: "Chandrayaan-3 successfully landed on the Moon."
  - Generated: "Chandrayaan-3 landed on the Moon."
  - Most unigrams match → High ROUGE-1.

- **BERTScore** → Uses BERT embeddings to measure semantic similarity between generated and reference summary.

  Example:

  - Reference: "ISRO’s Chandrayaan-3 landed successfully."
  - Generated: "Chandrayaan-3 touched down safely."
  - Even though words differ, embeddings are similar → High BERTScore.

Key Metrics

- **QA Eval** → Uses QA models to check whether generated summaries can answer key questions from the source/reference.

  Example:
  - Question: "Where did Chandrayaan-3 land?"
  - If generated summary answers “Moon’s south pole” → High QAEval score.

- **FACTCC** → Checks factual consistency between the summary and the source using a fact-checking classifier.

  Example:
  - Source: "Chandrayaan-3 landed on the Moon in 2023."
  - Generated: "Chandrayaan-3 landed in 2022." → Low FactCC score.

- **LLM As a judge** → Uses a large language model to evaluate a generated summary based on **correctness**, **completeness**, **conciseness** or other dimensions interested.

  Example:
  - Reference: “ISRO launched Chandrayaan-3 on July 14, 2023, aiming for a soft landing on the Moon’s south pole.”
  - Generated Summary: “ISRO launched Chandrayaan-3 on July 14, 2023, aiming for a soft landing.”
  - LLM Score: Correctness = 5/5, Completeness = 4/5, Conciseness = 5/5, Final Score = 4.7/5

Code Summarization

Code Summarization

Code Summarization is a task that tries to comprehend code and automatically generate descriptions directly from the source code

```java
private String toIndexName(Battle.ServerAction action) {
        String name = "Attacker";
        if (action.getAttackerIndex() == -1) {
                name = "Defender";
        }
        return name;
}
```

```java
/**
 * Converts the attacker index to a readable name
 *
 * @param action the action containing an index
 * @return a readable name for the attacker
 */
```

Summary of the Java method

Image taken from https://www.arxiv-vanity.com/papers/1708.01837/

Need for Code Comments

- It is estimated that developers spend about significant amount of their time in the program comprehension activity during the software maintenance effort.
- Many time developers handle code written by someone else
- Writing comments during the development is time-consuming for developers.
- Further, comments often doesn’t stay updated to the code changes.
- Therefore, there is a need to generate code summaries automatically. It can help save the developer’s time in writing comments, program comprehension, and code search.

Deal with Lack of Code Comments

- ❑ Obviate comments by descriptive identifier names e.g. getParametersOfMethodCall()
- ❑ Encourage and facilitate writing comments. Automatically prompt developers to enter comments
- ❑ Generate Comments. Extract key code statements or generate phrases or long descriptive summary of the code block

Source Code Modelling - Code as Code (~~Vs~~ &) Code as Text

- Text-only representations
  - Treating source code as series of tokens
    - Pick interesting tokens

```
void foo() {
    int x = source();
    if(x < MAX) {
        int y = 2*x;
        sink(y);
    }
}
```

```
['void', 'foo', '(', ')', '{', 'int', 'x',
'=', 'source', '(', ')', ';', 'if', '(', 'x', '<', 'MAX', ')', '{', 'int', 'y', '=',
'2*x', ';', 'sink', '(', 'y', ')', ';', '}', '}']
```

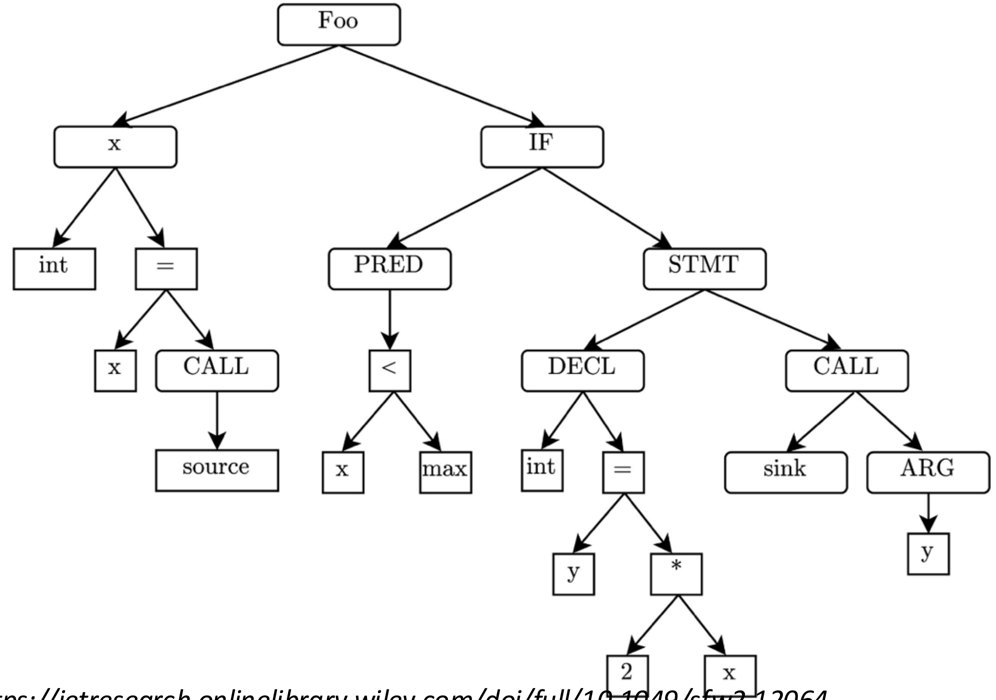
Ref : https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/sfw2.12064

Source Code Modelling - Code as Code (~~Vs~~ &) Code as Text

- Text-only representations
  - Treating source code as series of tokens
    - Pick interesting tokens

- Structured representations
  - Tree : Abstract Syntax Tree (AST)
    - AST into sequences, randomly extract AST paths, dividing AST into multiple sub-ASTs etc.
  - Graph : Control Flow Graph (CFG),
    Data Flow Graph (DFG),
    Program Dependency Graph (PDG) etc.

```
void foo() {
    int x = source();
    if(x < MAX) {
        int y = 2*x;
        sink(y);
    }
}
```



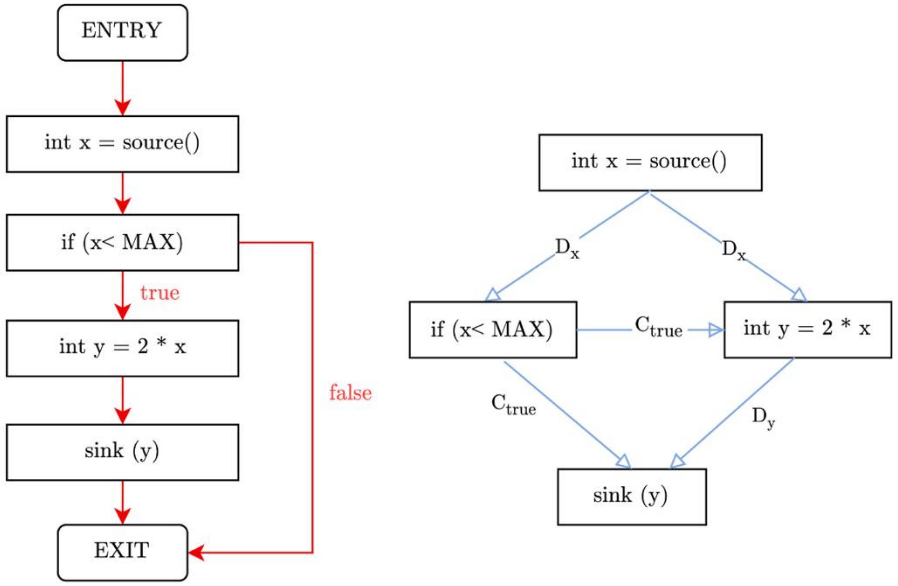
Ref : https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/sfw2.12064

Source Code Modelling - Code as Code (~~Vs~~ &) Code as Text

- Text-only representations
  - Treating source code as series of tokens
    - Pick interesting tokens

- Structured representations
  - Tree : Abstract Syntax Tree (AST)
    - AST into sequences, randomly extract AST paths, dividing AST into multiple sub-ASTs etc.
  - Graph : Control Flow Graph (CFG), Program Dependency Graph (PDG), Data Flow Graph (DFG), etc.

```
void foo() {
    int x = source();
    if(x < MAX) {
        int y = 2*x;
        sink(y);
    }
}
```



(a) Control flow graph (CFG) **(b)** Program dependence graph (PDG)

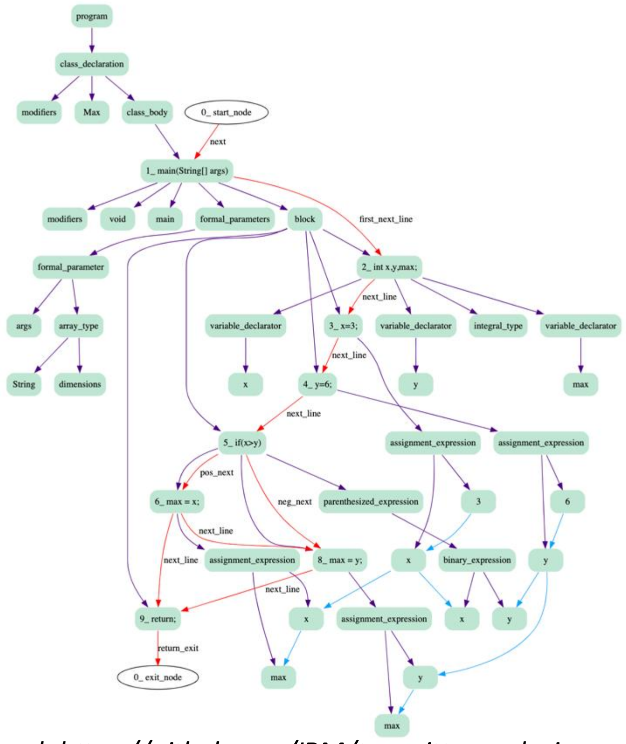
Ref : https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/sfw2.12064

Source Code Modelling - Code as Code (~~Vs~~ &) Code as Text

- ❑ Text-only representations
  - ▪ Treating source code as series of tokens
    - • Pick interesting tokens
- ❑ Structured representations
  - ▪ Tree : Abstract Syntax Tree (AST)
    - • AST into sequences, randomly extract AST paths, dividing AST into multiple sub-ASTs etc.
  - ▪ Graph : Control Flow Graph (CFG), Data Flow Graph (DFG), Program Dependency Graph (PDG) etc.
- ❑ Combined representations
  - ▪ Tokens +AST , Add different graphs-based edges to AST nodes etc.

```
public class Max {
    public static void main (String[] args) {
        int x,y,max;
        x=3;
        y=6;
        if (x>y)
            max = x;
        else
            max = y;

        return;
    }
}
```



Note : Generated through https://github.com/IBM/tree-sitter-codeviews

Code Summarization Techniques

- **Term Based :**

  Term-based summarization is to generate a summary that contains the most relevant terms for a specific software unit. Most of term-based summarization methods are connected with the information retrieval techniques.

- **Template Based :**

  In template-based summarization, there is a predefined set of summary templates, and the templates are filled based on the type of the target code segment and other information.

- **External Description Based :**

  External-description-based summarization uses external data such as comment-code mappings in other repositories or website forums.

- **Machine Learning Based :**

  Started with supervised & unsupervised learning. But Neural network based natural language generators are now more prevalent and show better efficiency.

Reference : *“Automatic Code Summarization: A Systematic Literature Review” Yuxiang Zhu et.al*

Term Based Summarization Techniques

- Works range from position of text to retrieval techniques
- The first step in almost of these techniques is to extract & process terms from code document.
- Apply different techniques from IR like LSI, VSM to extract top K weighted list or Identify top topics representing words through LDA etc.
- Just tags are not helpful for comprehension.

Term \ Document	Class A	Class B	Class C	Class A
setValue	0.043	0.001	0.21	0.13
counter	0.29	0.12	0.09	0.1
employee	0	0.078	0.03	0.22

The content of a cell in this matrix represents the weight of a code token (the row) with respect to a code document (the column) could be log, tf-idf, binary-entropy etc.

Haiduc et al. (2010), Eddy et al. (2013), Movshovitz-Attias and Cohen (2013), Rodeghero et al. (2014)

Template Based Summarization Techniques

- A Practical approach where summary templates are predefined. Based on the target code segments the templates are filled.
- Templates could cover program structural information such as the number of interfaces in a package or what kind of parameter does a method use or actions performed by different code fragments
- Many works leveraged the Software Word Usage Model (**SWUM**) to generate descriptions. SWUM is a technique for representing program statements as sets of nouns, verbs, and optional secondary arguments of a statement grouping
- The quality of the summary relies heavily on the quality of identifier names and method signatures in the source code

```
public boolean remove(Listener listener)
{
  if (listeners != null {
     int index = listeners.lastIndexOf(listener);
     if (index != -1) {
        try{
        listeners.remove(index);
        return true;
        }
        catch (numberException e)
        {
          return false;
        }
        if (listeners.isEmpty()) {
           listeners = null;
        }
     }
  }
}
```

(a)

This method checks if it can remove a listener.

It checks if [listeners] is not equal to null to do actions.

This method calls the method which get last index of the Listener in listeners to get its value and assign to variable [index].

This method finally returns [true] if remove [index] from [listeners].

It handles the errors using try-catch mechanism and throwing an exception.

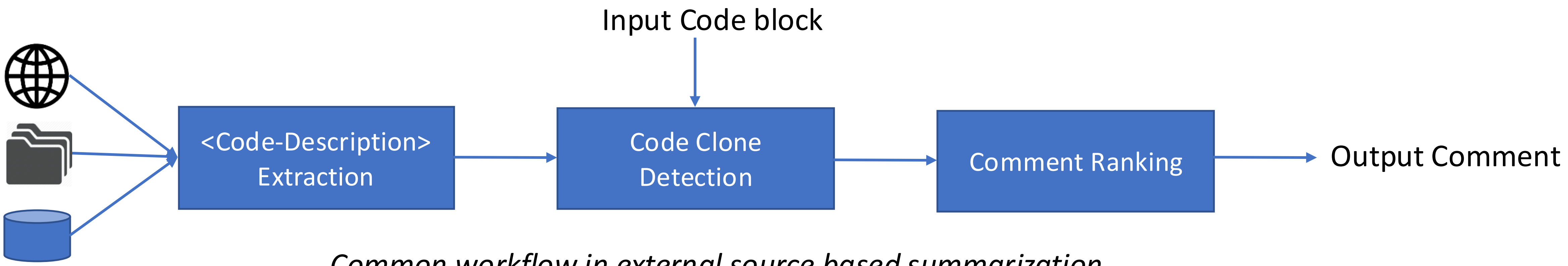
(b)

Badihi et.al (2017) CrowdSummarizer : Employing the Software Word Usage Model (SWUM) to generate a summary. (a) An example method. (b) The automatically generated summary. SWUM captures a methods linguistic elements in terms of its action, theme, and any secondary arguments

Sridhara et al. (2010,2011), Dawood et al. (2017) and Hammad et al. (2016)

External Description Based Summarization Techniques

- Developers post questions and receive solutions in online (Q&A) sites & many contribute to the open-source projects
- These sites/repositories contains code segments together with their descriptions
- Apply different similarity measure to find most similar code and then re-use the comment
- Correctness & Quality is again highly dependent on the community. Likely to miss out comments for code segments not discussed in QA sites. Lack of standardization of comments in open source can slow down processing.

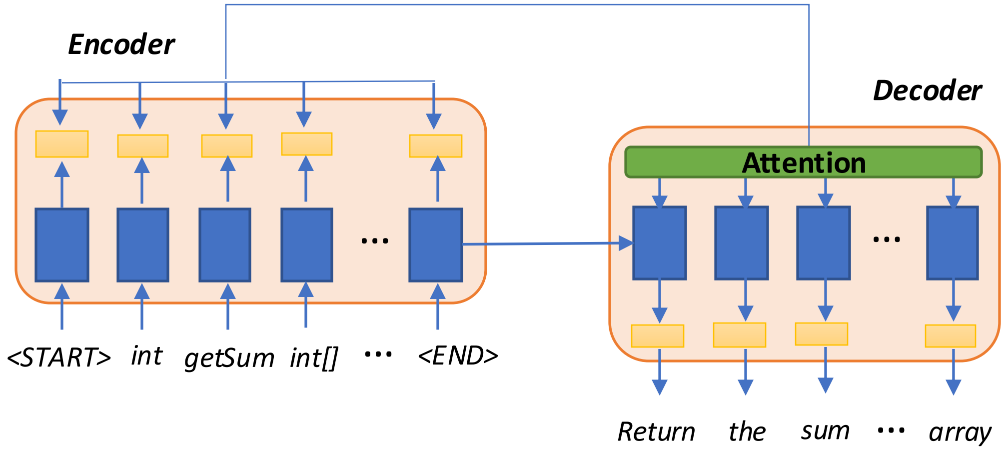


Common workflow in external source based summarization

Wong et al. (2013), Huang et al. (2017)

Summarizing source code using a neural attention model, Iyer et al (2016)

Simplified view of the approach



Primary Results

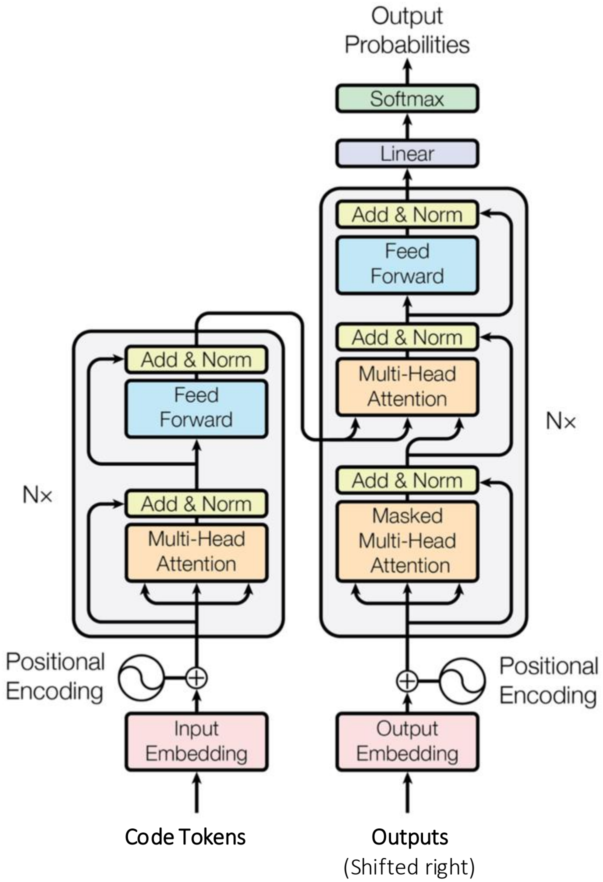
	Model	METEOR	BLEU-4
C#	IR	7.9 (6.1)	13.7 (12.6)
	MOSES	9.1 (9.7)	11.6 (11.5)
	SUM-NN	10.6 (10.3)	19.3 (18.2)
	CODE-NN	**12.3 (13.4)**	**20.5 (20.4)**
SQL	IR	6.3 (8.0)	13.5 (13.0)
	MOSES	8.3 (9.7)	15.4 (15.9)
	SUM-NN	6.4 (8.7)	13.3 (14.2)
	CODE-NN	**10.9 (14.0)**	**18.4 (17.0)**

- Address both code summarization and Code retrieval tasks
- Inspired by similar models in NLP tasks

- Crawl 934K C# and 977K SQL posts from StackOverflow
- Perform several preprocessing and cleaning steps. Used a small annotated dataset to clean data.
- Retain 66,015 C# (title, query) pairs and 32,337 SQL pairs, split as 80-10-10

- Also provide an analysis of attention weights learned and evaluation using additional quality measures such as ‘naturalness’ and ‘informativeness’

A Transformer-based Approach for Source Code Summarization, Ahmad et al (2020)



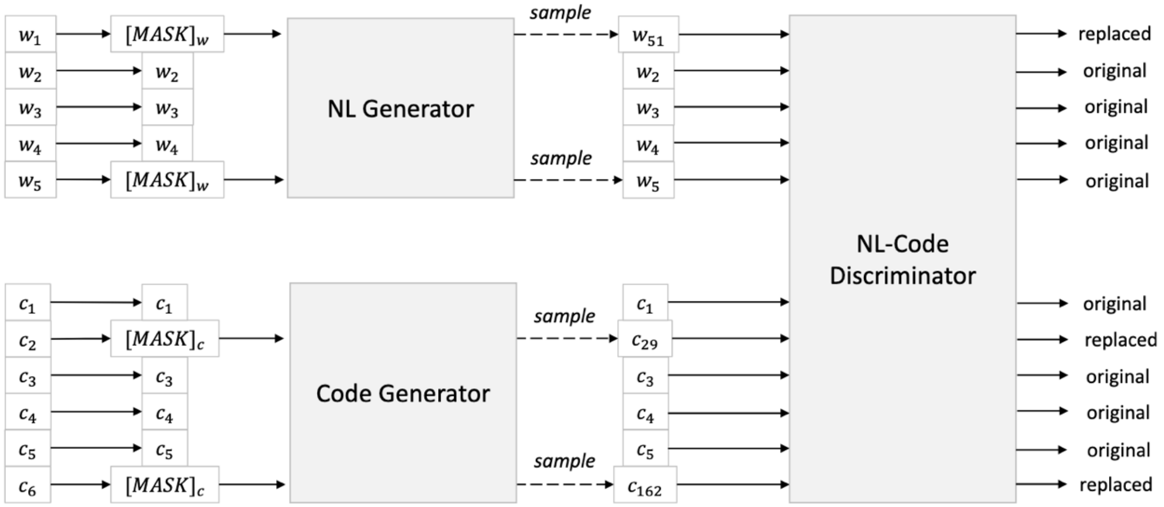
Relative Positional Encodings!
Copy-Attention!

- Transformers for source code had been applied previously, but not directly to source code summarization
- The RNN-based sequence models:
    - Do not model the non-sequential structure of source code
    - Unable to capture the long-range dependencies among code tokens
- Model the pairwise relationship between code tokens using relative position representation (Shaw et al, 2018)
- Use copy-attention to retain OOV words

Conduct experiments on a Java dataset (Hu et al) and a Python dataset(Wan et al)
The Base model out-performs the baselines in most cases
The Full model improves the performance further

Methods	Java			Python		
	BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L
CODE-NN (Iyer et al., 2016)	27.60	12.61	41.10	17.36	09.29	37.81
Tree2Seq (Eriguchi et al., 2016)	37.88	22.55	51.50	20.07	08.96	35.64
RL+Hybrid2Seq (Wan et al., 2018)	38.22	22.75	51.91	19.28	09.75	39.34
DeepCom (Hu et al., 2018a)	39.75	23.06	52.67	20.78	09.98	37.35
API+CODE (Hu et al., 2018b)	41.31	23.73	52.25	15.36	08.57	33.65
Dual Model (Wei et al., 2019)	42.39	25.77	53.61	21.80	11.14	39.45
Our models and ablation study						
Base Model	43.41	25.91	52.71	31.08	18.57	44.31
Full Model	**44.58**	**26.43**	**54.76**	**32.52**	**19.77**	**46.73**
Full Model w/o Relative Position	44.26	26.23	53.58	31.38	18.69	44.68
Full Model w/o Copy Attention	44.14	26.34	53.95	31.64	19.17	45.42

CodeBERT: A Pre-Trained Model for Programming and Natural Languages, Feng et al (2020)



- Evaluation is performed on CodeSearchNet dataset for 6 programming languages
- Also perform a study on languages not in pretraining

MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	OVERALL
SEQ2SEQ	9.64	10.21	13.98	15.93	15.09	21.08	14.32
TRANSFORMER	11.18	11.59	16.38	15.81	16.26	22.12	15.56
ROBERTA	11.17	11.90	17.72	18.14	16.47	24.02	16.57
PRE-TRAIN W/ CODE ONLY	11.91	13.99	17.78	18.58	17.50	24.34	17.35
CODEBERT (RTD)	11.42	13.27	17.53	18.29	17.35	24.10	17.00
CODEBERT (MLM)	11.57	14.41	17.78	18.77	17.38	24.85	17.46
CODEBERT (RTD+MLM)	**12.16**	**14.90**	**18.07**	**19.06**	**17.65**	**25.16**	**17.83**

Table 4: Results on Code-to-Documentation generation, evaluated with smoothed BLEU-4 score.

- A bimodal pre-trained model for natural and programming languages capturing semantic connections between the two
- Trained with a hybrid objective function, including standard **masked language modeling(MLM)** and **replaced token detection (RTD)**

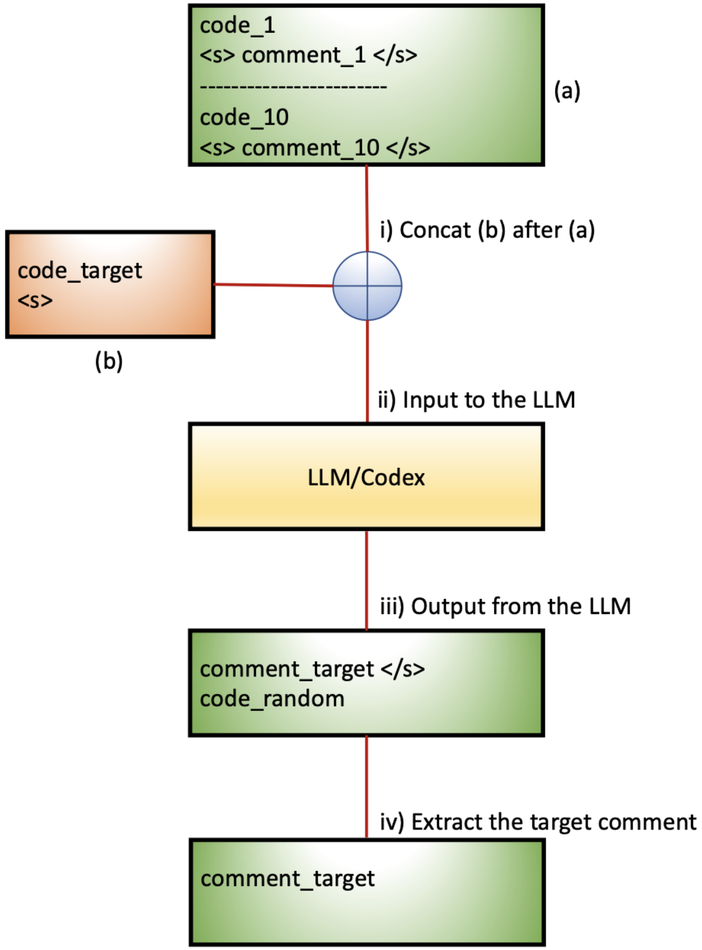
MODEL	BLEU
MOSES (KOEHN ET AL., 2007)	11.57
IR	13.66
SUM-NN (RUSH ET AL., 2015)	19.31
2-LAYER BILSTM	19.78
TRANSFORMER (VASWANI ET AL., 2017)	19.68
TREELSTM (TAI ET AL., 2015)	20.11
CODENN (IYER ET AL., 2016)	20.53
CODE2SEQ (ALON ET AL., 2019)	**23.04**
ROBERTA	19.81
PRE-TRAIN W/ CODE ONLY	20.65
CODEBERT (RTD)	22.14
CODEBERT (MLM)	22.32
CODEBERT (MLM+RTD)	**22.36**

Table 5: Code-to-NL generation on C# language.

Pretraining:

- Input is the concatenation of code and language tokens separated by [SEP]
- The MLM objective is to predict the original tokens that are masked out
- The discriminator is trained to determine if the predicted words are the original ones or not
- After training, the Generators are discarded and the Discriminator is used for fine-tuning tasks

Few-shot training LLMs for project-specific code-summarization, (Ahmad et al 2022)



- Task → prepend n functions (cross-project/ same-project), each followed by a comment, followed by the target function for which the model is to generate the comment

Observations :

- With 10 samples, Codex outperforms all finetuned foundation models CodeT5, CodeBERT
- Same-project few-shot training improves the Codex model’s performance

Tasks and Datasets

Code Summarization

1. Method name prediction*

```
public boolean ______(Set<String> set, String value) {
  for (String entry : set) {
    if (entry.equalsIgnoreCase(value)) {
      return true;
    }
  }
  return false;
}
```

contains ignore case

2. Method documentation generation (Code captioning)*

```
void Main() {
  string text = File.ReadAllText(@"T:\File1.txt");
  int num = 0;
  text = (Regex.Replace(text, "map", delegate(Match m) {
    return "map" + num++;
  }));
  File.WriteAllText(@"T:\File1.txt", text);
}
```

Replace a String from a Text File

Related Tasks

Code Search
Code Generation
Bug/Vulnerability detection

Datasets

MultiLingual

CodeSearchNet – Husain et al (2019)
- 2.3M (code, comment) pairs over 6 languages

Java

Hu et al (2018)
- 69K (code, comment) pairs (training set)

Alon et al (2019)
- Java-small, Java-medium, Java-large
- 700K, 4M, 16M (code, comment) pairs

Python

Barone et al (2017)
- 55k (code, comment) pairs (training set)

C#

Iyer et al (2016)
- 66k (code, comment) pairs

Metrics

BLEU, Meteor, Rouge, Precision-Recall-F1

* Examples from Alon et al (2019)

Agentic Multi-Modal LLMs for Software Comprehension

Structuring Code Summarization with Business Process Awareness

Srikanth Tamilselvam and Ashita Saxena

Accepted at IEEE SSE 2025

Why Current Application Summaries Fall Short?

User: Can you give summary of the business capabilities of the uploaded Daytrader repo?

Current state of summarization (GithubCoiplot with Sonnet3.5):

1. Account Management
    - Create new trading accounts
    - Login/Logout ...
2. Stock Trading
    - Buy and sell stock orders
    - View current quote prices ...
3. Market Information
    - Real-time stock quotes
    - Market Summary Views ...
4. Authentication & Security
    - User authentication
    - Session management ...
5. Performance Features
    - Caching for frequently accessed data
    - Connection pooling

Copilot + Claude Sonnet on DayTrader

Why Current Application Summaries Fall Short?

❌ Flat, syntax-centric summaries

User: Can you give summary of the business capabilities of the uploaded Daytrader repo?

Current state of summarization (GithubCoiplot with Sonnet3.5):

1. Account Management
   - Create new trading accounts
   - Login/Logout ...
2. Stock Trading
   - Buy and sell stock orders
   - View current quote prices ...
3. Market Information
   - Real-time stock quotes
   - Market Summary Views ...
4. Authentication & Security
   - User authentication
   - Session management ...
5. Performance Features
   - Caching for frequently accessed data
   - Connection pooling

Copilot + Claude Sonnet on DayTrader

Why Current Application Summaries Fall Short?

❌ Flat, syntax-centric summaries

❌ Unnatural sequencing (e.g., "trading" before "login")

User: Can you give summary of the business capabilities of the uploaded Daytrader repo?

Current state of summarization (GithubCoiplot with Sonnet3.5):

1. Account Management
   - Create new trading accounts
   - Login/Logout ...
2. Stock Trading
   - Buy and sell stock orders
   - View current quote prices ...
3. Market Information
   - Real-time stock quotes
   - Market Summary Views ...
4. Authentication & Security
   - User authentication
   - Session management ...
5. Performance Features
   - Caching for frequently accessed data
   - Connection pooling

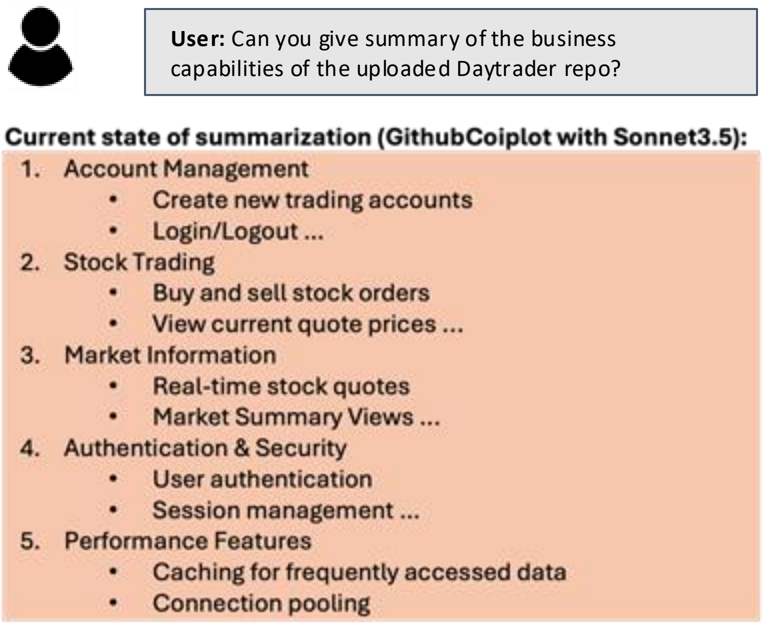
Copilot + Claude Sonnet on DayTrader

Why Current Application Summaries Fall Short?

❌ Flat, syntax-centric summaries

❌ Unnatural sequencing (e.g., "trading" before "login")

❌ Low domain understanding



User: Can you give summary of the business capabilities of the uploaded Daytrader repo?

Current state of summarization (GithubCoiplot with Sonnet3.5):

1. Account Management
   - Create new trading accounts
   - Login/Logout ...
2. Stock Trading
   - Buy and sell stock orders
   - View current quote prices ...
3. Market Information
   - Real-time stock quotes
   - Market Summary Views ...
4. Authentication & Security
   - User authentication
   - Session management ...
5. Performance Features
   - Caching for frequently accessed data
   - Connection pooling

Copilot + Claude Sonnet on DayTrader

Business-Aware Summarization

We aim to generate summaries that:

🧠 Abstract technical details into business functions

🔀 Follow the real user flow

📋 Are structured, meaningful, and domain-aligned

User: Can you give summary of the business capabilities of the uploaded Daytrader repo?

Current state of summarization (GithubCoiplot with Sonnet3.5):

1. Account Management
    - Create new trading accounts
    - Login/Logout ...
2. Stock Trading
    - Buy and sell stock orders
    - View current quote prices ...
3. Market Information
    - Real-time stock quotes
    - Market Summary Views ...
4. Authentication & Security
    - User authentication
    - Session management ...
5. Performance Features
    - Caching for frequently accessed data
    - Connection pooling

Expected state of summarization:

The DayTrader application is a comprehensive trading and portfolio management platform that enables users to manage their investments through a series of streamlined services. The application begins with **User Login**, ensuring secure access to the platform. Once logged in, users can access **Portfolio View**, where they can monitor their current investments and evaluate performance. The **Quote Lookup** service allows users to check real-time stock information, providing crucial insights to guide their investment decisions. When ready to act, users can utilize the **Trade Execution** service to buy or sell stocks in real time, seamlessly updating their portfolio.

LLMs can reason—but better with the right signals

Inputs are rich but fragmented

🧩 Code Entry Points

📄 Textual Docs (e.g., README.md)

💡 Domain Knowledge

LLMs can reason—but better with the right signals

Inputs are rich but fragmented

🧩 Code Entry Points

📄 Textual Docs (e.g., README.md)

💡 Domain Knowledge

Use **specialized LLM agents** to reason through these and synthesize structured outputs

LLMs can reason—but better with the right signals

Inputs are rich but fragmented

🧩 Code Entry Points

📄 Textual Docs (e.g., README.md)

💡 Domain Knowledge

LLM Agents help organize reasoning

- Modular, step-by-step processing
- Clear responsibility and verification

Use **specialized LLM agents** to reason through these and synthesize structured outputs

LLMs can reason—but better with the right signals

Inputs are rich but fragmented

🧩 Code Entry Points

📄 Textual Docs (e.g., README.md)

💡 Domain Knowledge

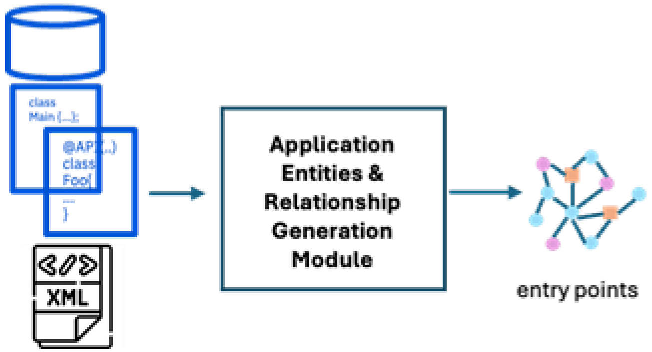
LLM Agents help organize reasoning

- Modular, step-by-step processing
- Clear responsibility and verification

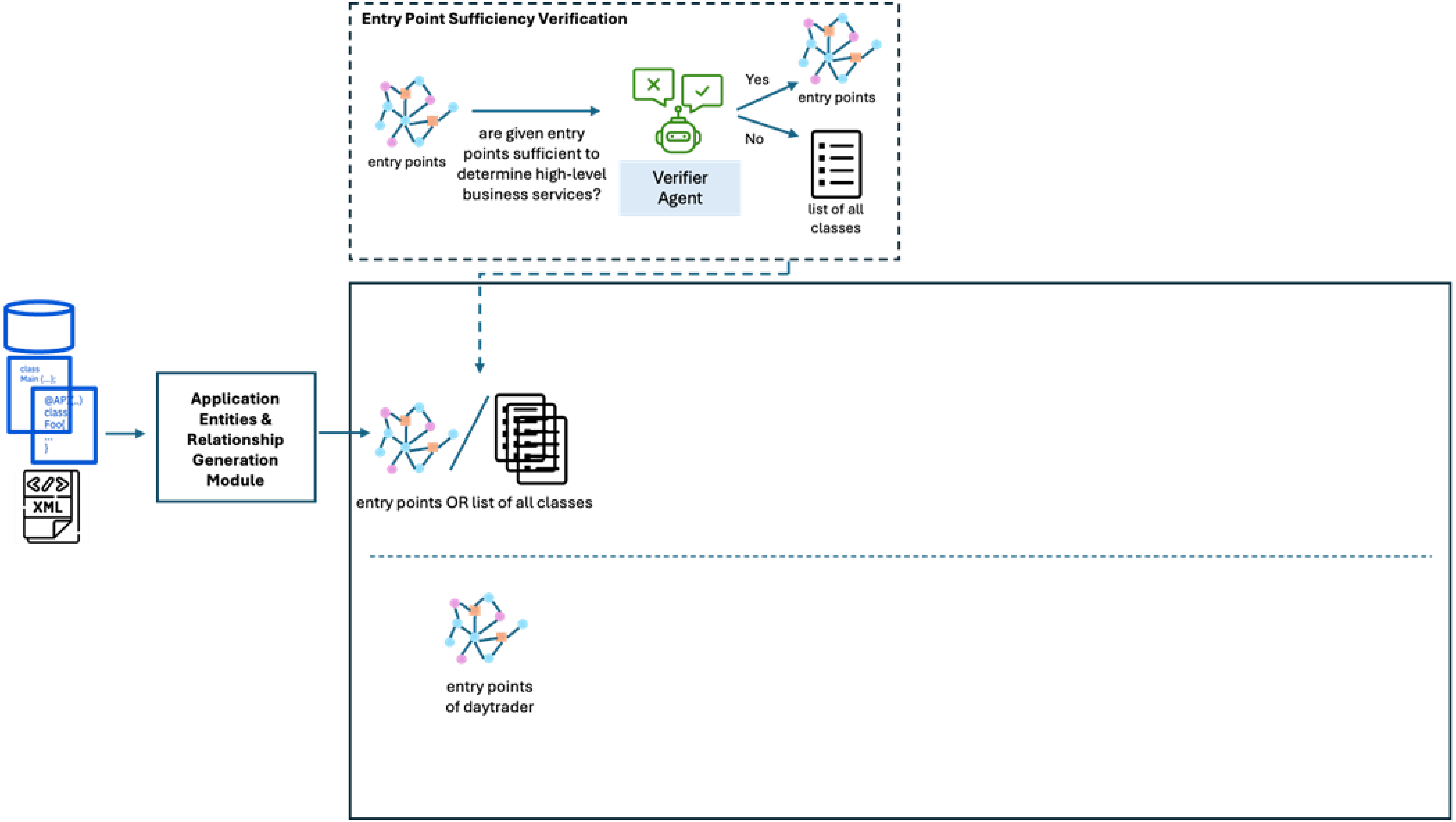
Use **specialized LLM agents** to reason through these and synthesize structured outputs

Agents enable traceability, reuse, and better failure handling.

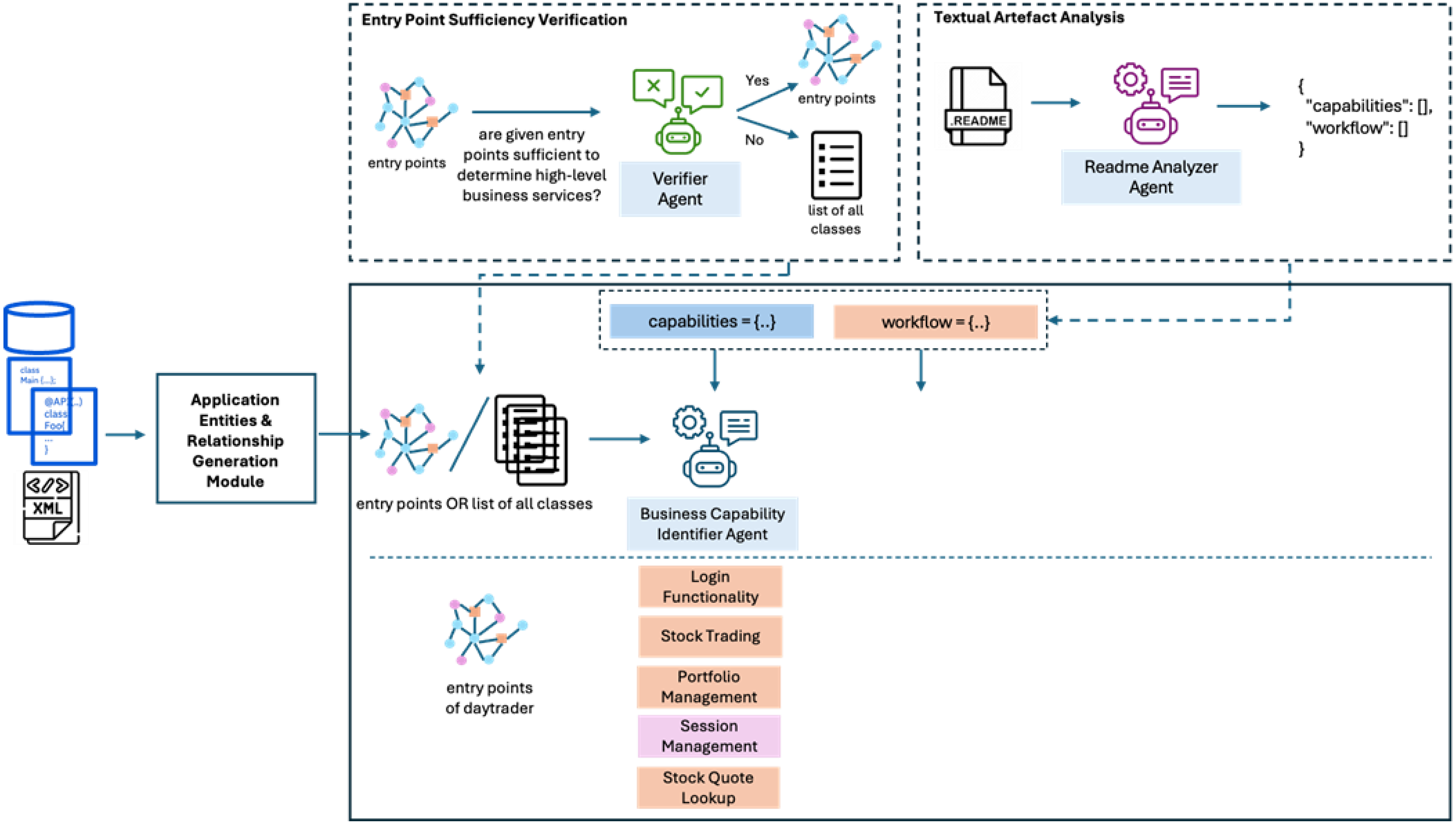
Agentic Multi-Modal Summarization Pipeline



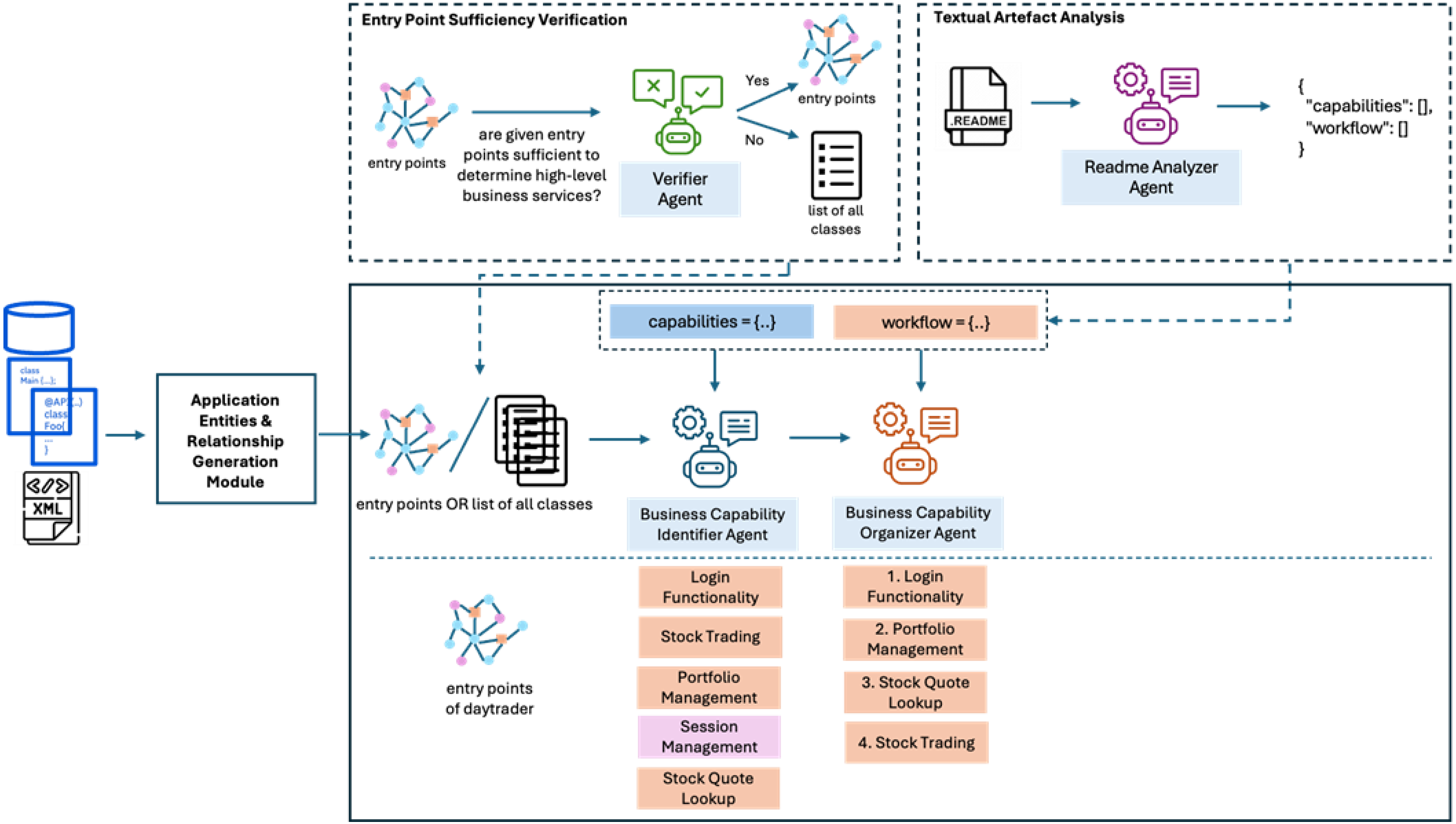
Agentic Multi-Modal Summarization Pipeline



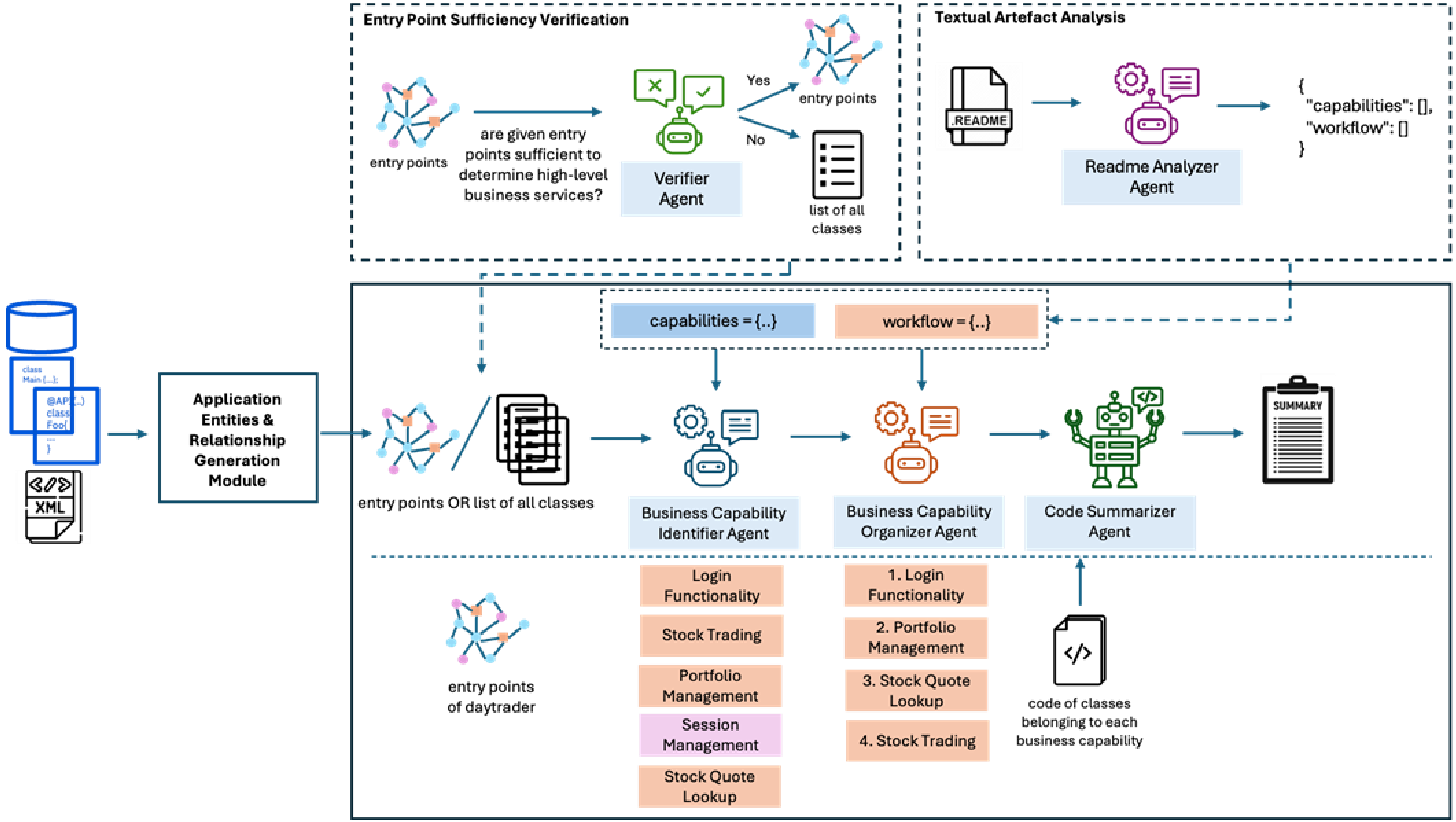
Agentic Multi-Modal Summarization Pipeline



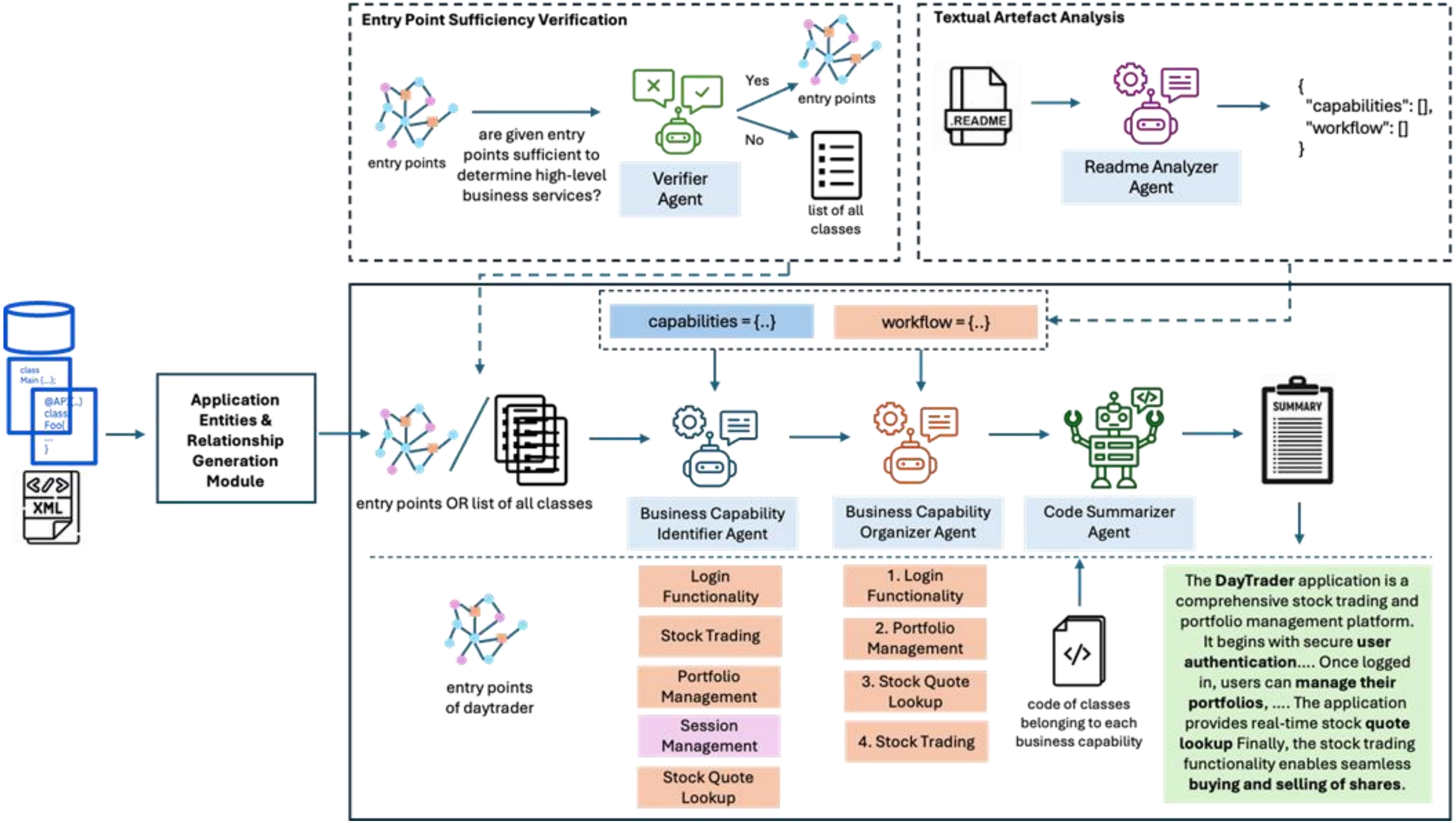
Agentic Multi-Modal Summarization Pipeline



Agentic Multi-Modal Summarization Pipeline



Agentic Multi-Modal Summarization Pipeline



Evaluations

Application	Are Entry Points Sufficient?	Generated Clusters of Business Capabilities	Extracted Order of Business Services	Cluster Correctness & Mapping Accuracy	Work Flow Correctness
DayTrader [Trading App, #classes = 111]	Yes	Login Functionality, Stock Trading, Stock Quote Lookup, Session Management, Portfolio Management	Login Functionality → Portfolio Management → Stock Quote Lookup → Stock Trading	Complete Match	Fully Aligned
PlantsByWebSphere [Plant Inventory App, #classes = 36]	No	Account Management, Shopping and Order Management, Inventory and Backorder Management, Supplier Management, Help and Utilities, Image and Populate	Account Management → Shopping and Order Management → Inventory and Backorder Management → Supplier Management	Complete Match	Fully Aligned
Acme-Air [Airline App, #classes = 38]	Yes	User Management, System Configuration and Monitoring, Flight Management, Booking Management, Data Analytics	User Management → Flight Management → Booking Management	Complete Match	Fully Aligned
PetClinic [Veterinary Clinic Management System,#classes = 30]	Yes	Pet Management, Owner Management, Veterinarian Management, Error Handling	Owner Management → Pet Management → Veterinarian Management	Partial Match	Fully Aligned

Human validation from SME with 18+ years of experience

✅ All workflows aligned
❌ Only PetClinic missed one service

ETF: An Entity Tracing Framework for Hallucination Detection in Code Summaries

Kishan Maharaj, Vitobha Munigala, Srikanth G. Tamilselvam, Prince Kumar, Sayandeep Sen,
Palani Kodeswaran, Abhijit Mishra, Pushpak Bhattacharyya

Accepted at ACL 2025

What is Hallucination?

- The term hallucination was inspired by psychology
  - In medical science, hallucinations refer to the particular type of perception realised by an individual without any external stimulus (Blom, 2010).

- In context of Natural Language Processing:
  - In the same way, the generated text may contain information that might look correct, but maybe unfaithful to the reference (context or world knowledge)

- Types of Hallucination
  - Intrinsic Hallucination
  - Extrinsic Hallucination

Intrinsic Hallucination

- Intrinsic Hallucination
  - Occurs when the output generated by a model contradicts the source text
  - Input document:
    - “Marie Curie discovered radium in 1898 at the University of Paris, marking a groundbreaking moment in the history of science. Her pioneering research in radioactivity, conducted alongside her husband Pierre Curie….”
  - Generation:
    - “Marie Curie invented radium in 1898.”
  - Explanation:
    - discovery → Invention

Extrinsic Hallucination

- Extrinsic Hallucination
  - Occurs when the generated output cannot be verified from the source text
  - Input document:
    - "Marie Curie discovered radium in 1898 at the University of Paris, marking a groundbreaking moment in the history of science. Her pioneering research in radioactivity, conducted alongside her husband Pierre Curie…."
  - Generation:
    - "Marie Curie was born in Warsaw, Poland."
  - Explanation:
    - No information about the birthplace of Marie Curie.

Hallucination Detection in Code Summarization

- The current code model have high tendency to produce unrelated code summaries
  - Also includes Imaginary entities
- The current SOTA language models are very good at hallucinating in a convincing way:
  - Guessing the summary based on lexical interpretation of the code.
  - references to some functions/classes not available during the input time

Input

Code: *int getJobID (String jobName) {*

```
int getJobID (String jobName) {
    return -1;}

private void runJob (String jobName) {
        int x = getJobId(jobName);
    }
```

Summary: *The getJobID method is a private method that takes a job name as a parameter and returns the job ID after fetching it from a database. The runJob method uses the getJoobID method to get the job ID and then uses the job ID to run the job.*

Output: Hallucinated

Dataset Creation

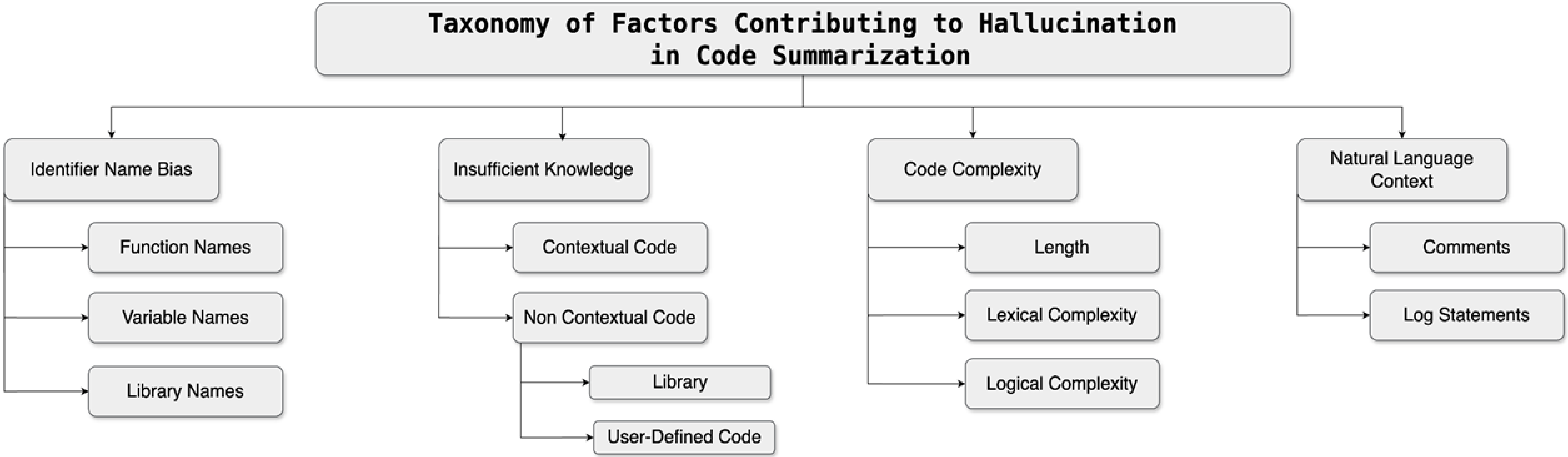
- Programming language: Java
  - Code snippets: CodeXGLUE (Lu et al., 2021) – Code-To-Text dataset.

- Study Natural Hallucination in Code Summaries
  - Do not introduce perturbations → Artificial Hallucination

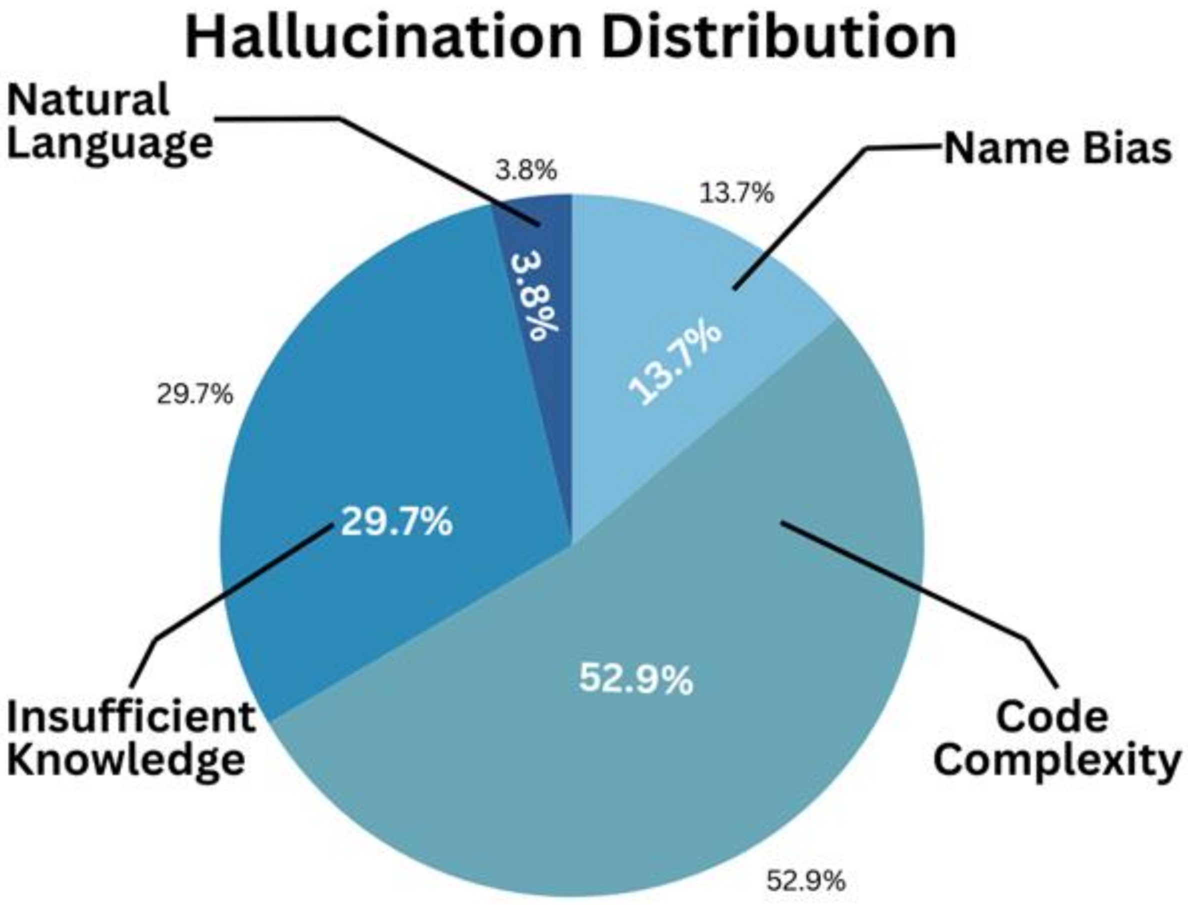
  - Rely on Natural Generation of Code/Language Models
    - IBM-Granite family (20B and 34B)
    - Llama3 family (8B and 70B);
    - CodeLlama family (7B and 34B)
    - Mistral-7B

  - Include different level of abstractions:
    - Low Abstraction → Describe the code line by line (detailed)
    - High Abstraction → Describe the business purpose of the code (high level)

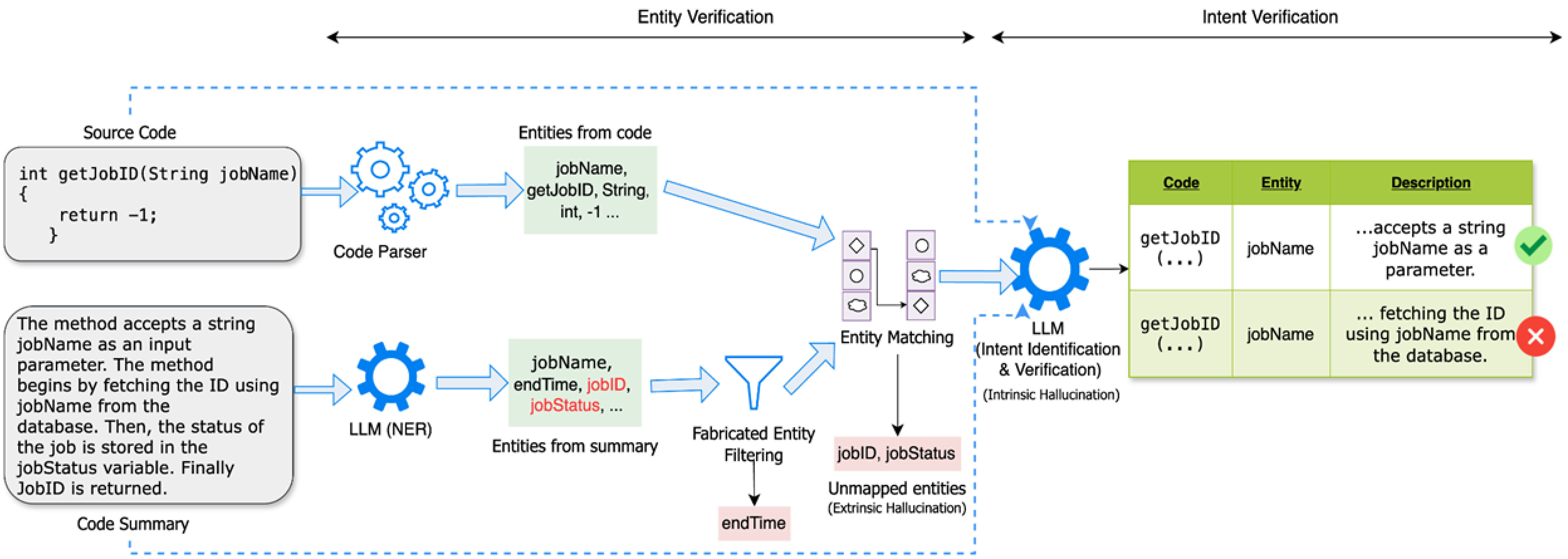
Proposed Taxonomy



Dataset Statistics



Approach: Entity Tracing Framework



Measuring What Matters: An Aggregate Metric for Assessing Enterprise Code Summaries

Ashita Saxena, Palanivel Kodeswaran, Sayandeep Sen, Srikanth Tamilselvam

Accepted at FSE 2025

Motivation

- Existing summarization benchmarks focus on small code snippets.
- Enterprise Java codebases: avg. 231 code tokens vs ~30 (5 LoC) in public datasets.
- SME-written summaries are longer and more informative.

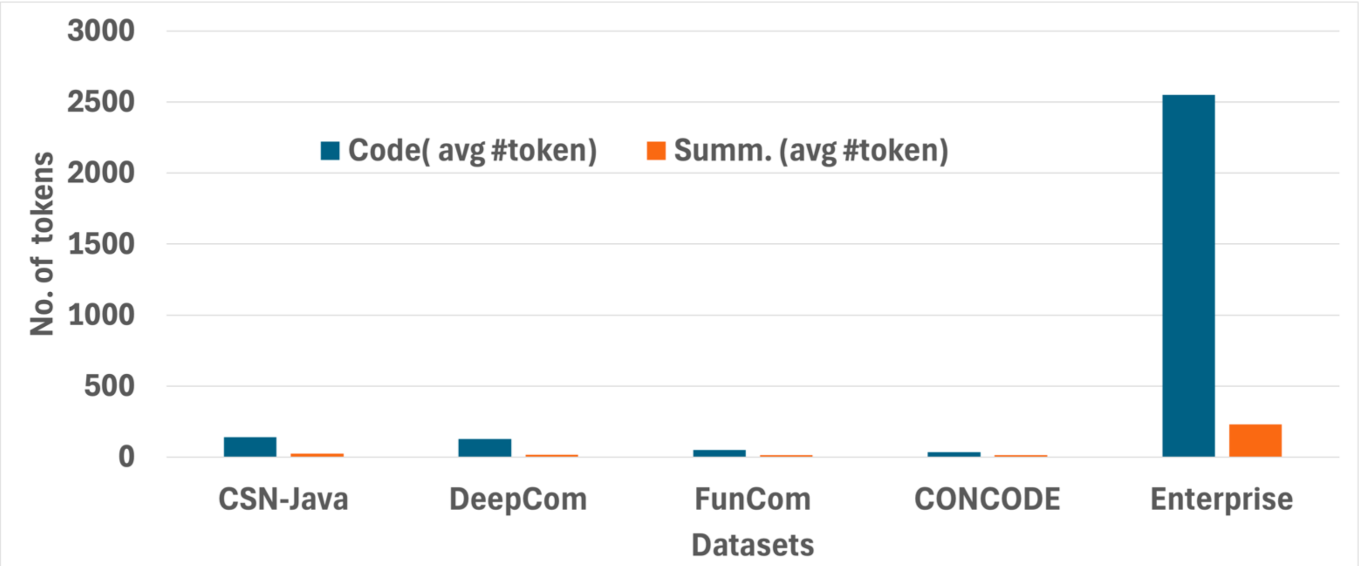


Fig 1 : Comparison of average no. of code tokens and average no. of summary tokens across current public summarization datasets and samples from enterprise

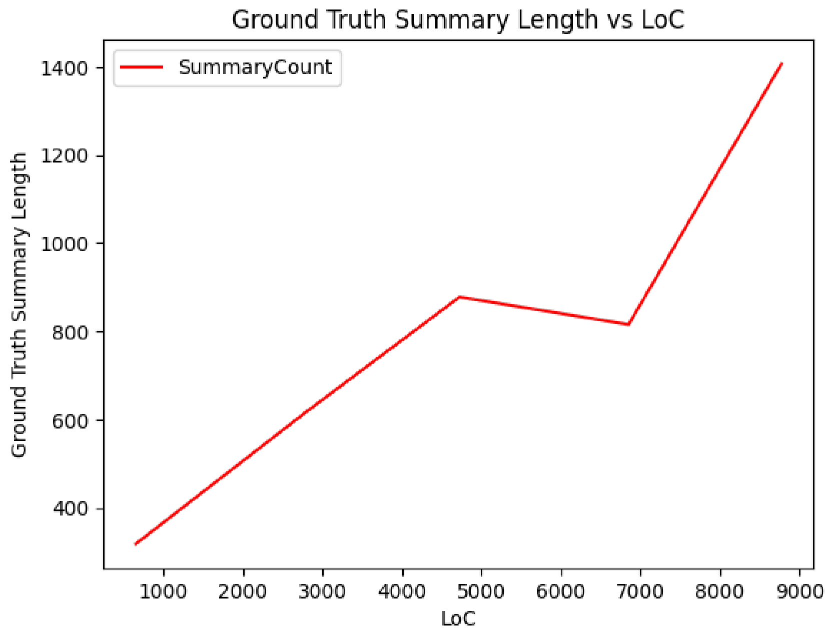
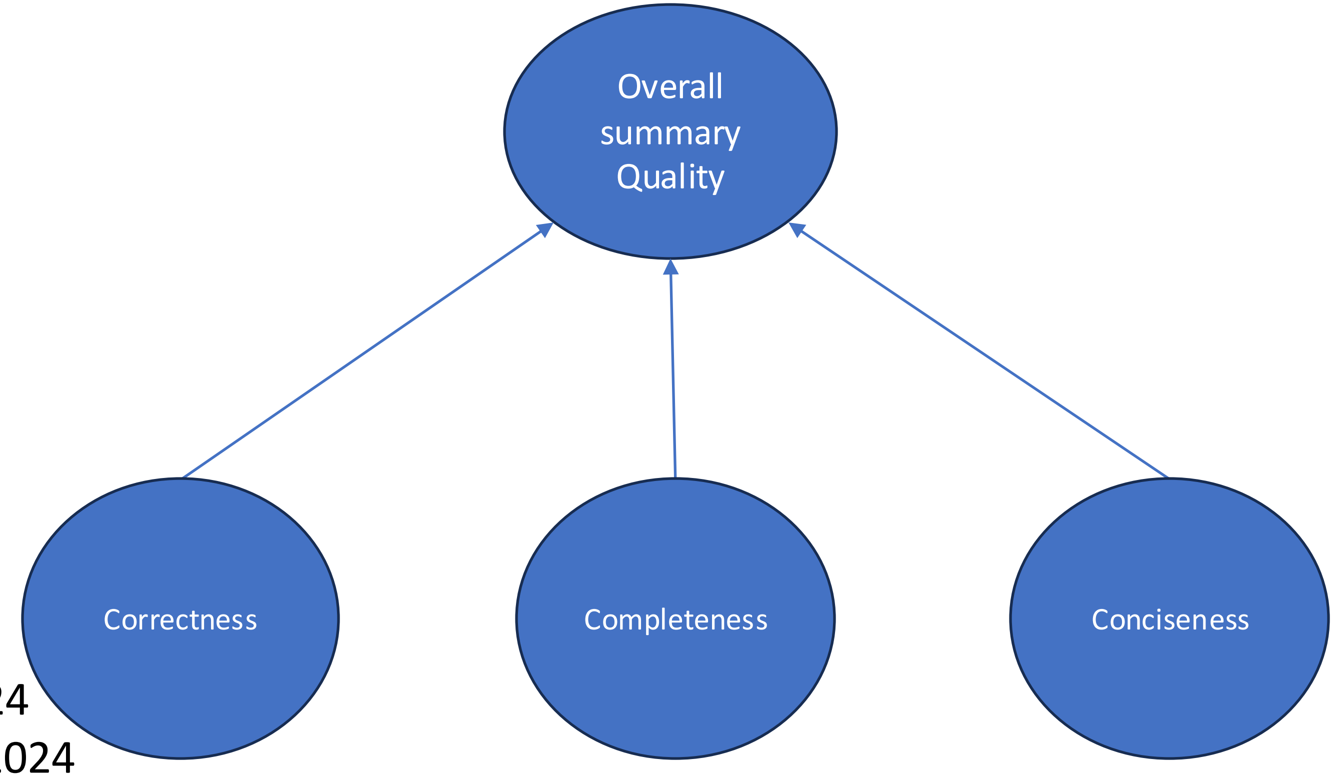


Fig 2 : Ground Truth Summary Length (no. of tokens) vs Lines of Code (LoC) from SMEs

Motivation



Tian et.al, 2024
Haldar et.al, 2024
Maharaj et.al, 2024

- Ignore verbosity, repetitiveness, and incompleteness.
- Need for **new evaluation dimensions**.

Example

Summary of the WeatherServlet class of ModResort

The `WeatherServlet` class handles weather data retrieval in the ModResort application. It processes HTTP requests using the `doGet` and `doPost` methods, ensuring users receive weather data. The `doPost` method processes POST requests by calling the `doGet` method. The class also initializes resources during startup and cleans up during shutdown. Exception handling and logging are used extensively to monitor and manage errors.

The `doGet` method retrieves weather information. If the API key is available, real-time weather data is fetched; otherwise, default data is provided. Logging ensures system activities are tracked, and errors are logged at various levels for troubleshooting. The `mockKey` method masks sensitive API keys for security.

The `init` method sets up the environment for the servlet, preparing MBeans and the initial context required for application management, while ensuring all properties are correctly configured for smooth operation.

The class is versatile, handling weather data retrieval, exception management, and lifecycle operations, making it an essential part of the application.

Issues:

- Redundant logging mentions
- Superficial method references
- Misses complex, public methods

Ref : https://github.com/IBM/appmod-resorts/blob/master/src/main/java/com/ibm/ta/modresorts/WeatherServlet.java

Contributions -Distinctness Metric

Penalizes:

- Repetitive sentences (clustering on SBERT embeddings)
- Verbose language

Key idea:

- Cluster count ≠ sentence count ⇒ redundancy
- Summary/code token ratio ⇒ verbosity

Contributions - Completeness Metric

Captures:

- Coverage of complex/public methods (by cyclomatic complexity)
- Diversity of content using inverse self-BLEU

- Scalable, language-agnostic design
- Aligns with SME perceptions of usefulness

Contributions – Aggregated Metric

- Support Vector Regression (SVR)
  - Input: Distinctiveness + Completeness
  - Output: Predicted overall summary quality (1–10)
- Learns to match SME judgments
- Robust to noise and small dataset size

Experiments

- Dataset: 70 Java classes from 9 internal apps (HR, SSO, travel)
- SMEs: 5 senior engineers (15+ yrs experience)
- Rating dimensions: correctness, completeness, conciseness, overall

Header Name	Description	Expected Feedback
Correctness (Accurate Information)	Does generated summary accurately represent the functionality of the code? **Note**: Give high score even if it has extraneous information (right or wrong)	1-10
Completeness	The generated summary provides all necessary information to understand the Source code functionality and purpose. **Note**: Give high score even if it has extra information.	1-10
Extra Details (Conciseness)	If text was too long, what to cut (**OR** What trivial information was emphasised). **Note**: Separated by '---' separator if more than 1	1-10
Acceptable	In your opinion is the output on a whole acceptable?	Yes/No
Overall Score	Overall grade for this summary	[1-10]

Figure : SME questionnaire for evaluating summaries

Results

Table 1: Comparison of Root Mean Squared Error (RMSE) for different metric combinations used in the aggregate metric.

Metrics Used for Aggregate Score	RMSE
Baseline	1.38
Laaj	0.68
Baseline + Laaj	0.97
Distinctiveness + Completeness	**0.63**

Table 2: Correlation of different metrics with human evaluation of the summary (pearson correlation).

Metric	Correlation
Side score	-0.37
CodeT5 based similarity score	0.51
Diversity score based on Self-BLEU	0.56
Completeness	**0.59**
Distinctiveness	**0.74**

Conclusion - Summarization Focus Areas

- **Foundational**
- **Application**
- **System**

Foundational

- **Accuracy & Hallucination Mitigation**

  Challenges
  - Models fabricate facts, APIs, or results.
  - Hard to verify factual correctness automatically.

- **Context & Long-sequence Understanding**

  Challenges
  - Handling books, multi-document corpora, and large enterprise codebases.
  - Models struggle with maintaining coherence over long contexts.

- **Evaluation Metrics & Benchmarking**

  Challenges
  - ROUGE, BLEU, and METEOR fail to capture semantic quality.

Application

- **Domain-specific & Task-oriented Summarization**

  Challenges

  - Generic summarizers underperform in specialized domains.
  - Summarizing enterprise codebases and regulatory documents needs deep domain knowledge.

- **Personalization & User Intent Awareness**

  Challenges

  - Many models generate “one-size-fits-all” summaries.
  - Different stakeholders (developers, managers, researchers) need different levels of detail.

System

- **Scalability & Efficiency**

  Challenges
  - Handling enterprise-scale repositories and large document sets.
  - Reducing computational cost for edge deployments.

- **Explainability**

  Challenges
  - Summaries are black boxes; users can’t verify why content was included.
  - Lack of provenance tracking in code and text summaries.

- **Multilingual & Cross-lingual Summarization**

  Challenges
  - Limited datasets for low-resource languages and code comments.
  - Cross-lingual summarization of code + documentation is underexplored.

THANK YOU

References

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- https://medium.com/@henrymao/reinforcement-learning-using-asynchronous-advantage-actor-critic-704147f91686
- https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a
- https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#0458
- https://towardsdatascience.com/an-intuitive-explanation-of-self-attention-4f72709638e1
- https://jalammar.github.io/illustrated-transformer/
- https://arxiv.org/abs/2406.11289