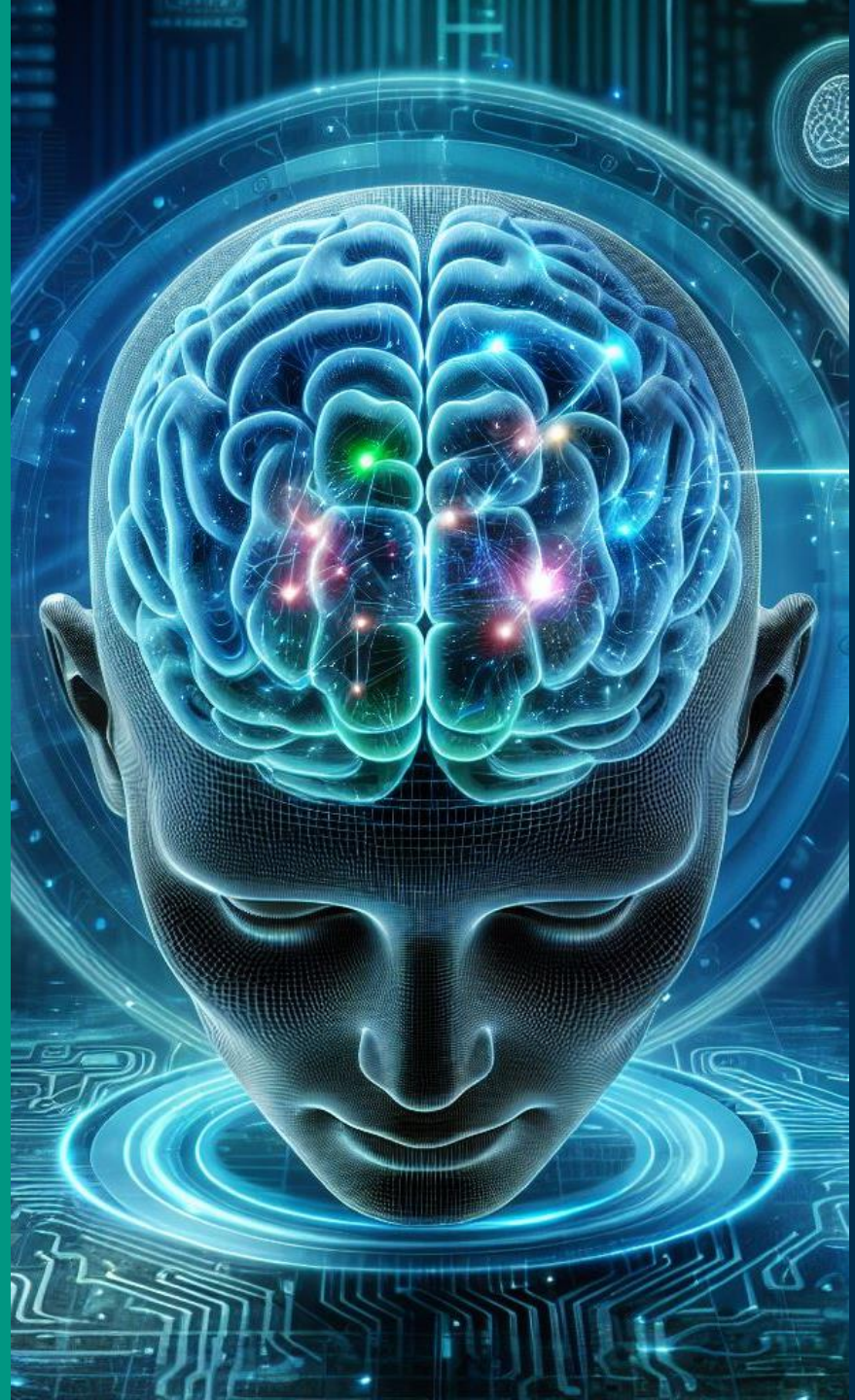
Software Engineering II

Natural Language Processing And Software Engineering

Lecture by Dr.-Ing. Tobias Hey and Dr.-Ing. Jan Keim



Motivating example

What kind of requirement is it?

Where is the requirement implemented? Where is the corresponding design document?

Requirement

After the user submitted the search query, the system displays the list of results containing matching hotels.

Does the requirement fulfill the quality standards?

```
public class Hotel
  public String getAddress();
  public List getRooms();
  public void bookRoom(Room);
```

```
public class Customer
  public int getIDNumber();
```

```
public class Room
  public RoomType getType();
  public float getCostsPerNight();
```

```
public class Booking
  public float getCosts();
  public String getCustomerName();
```

```
public class Search
  public List query(String);
```

```
public class Listings {

  private Map listing;
  private List hotels;

  public List retrieveListings(){
    ...
  }

  public List listHotels() {
    ...
  }
```

Motivating example

What kind of requirement is it?

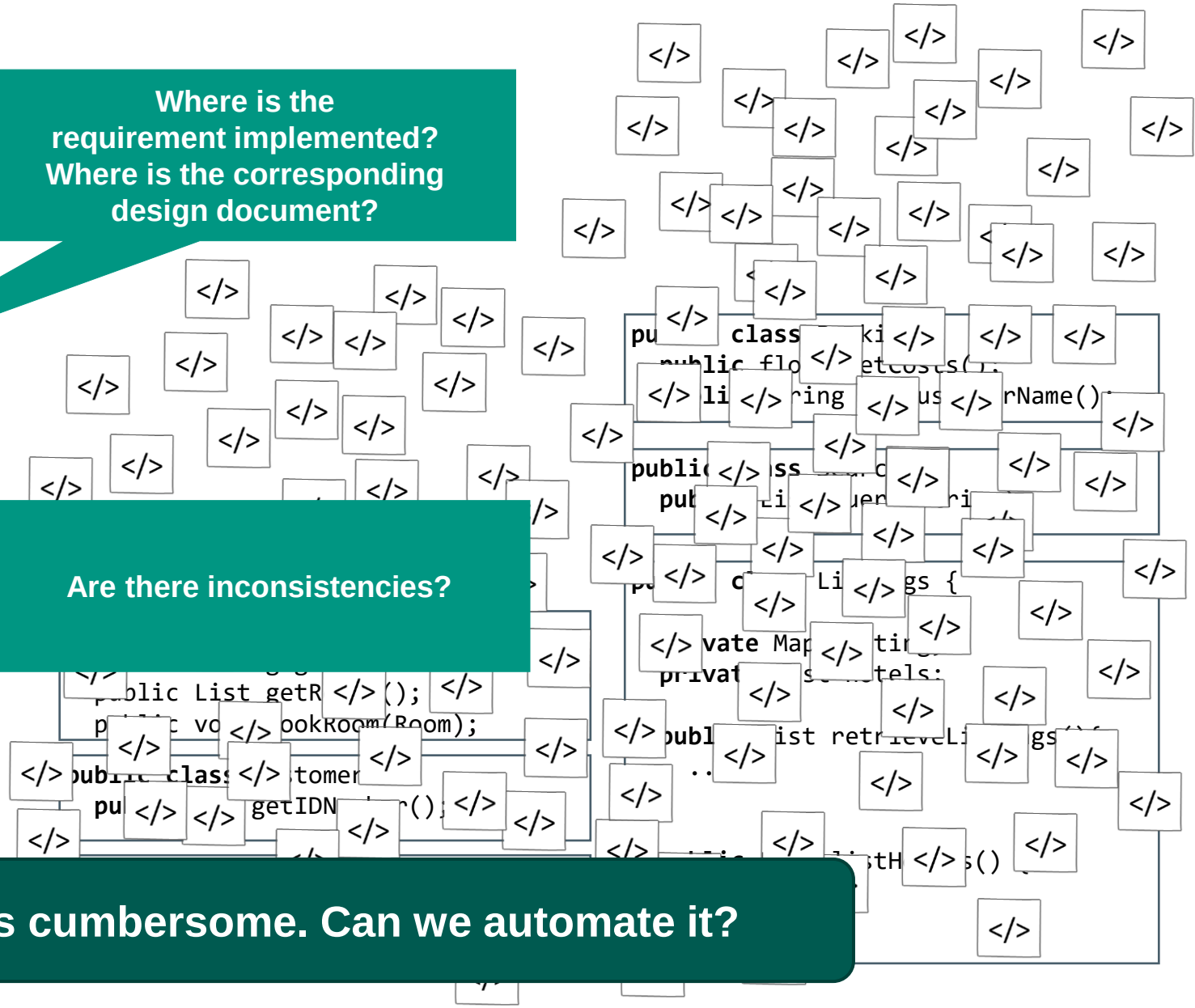
Where is the requirement implemented? Where is the corresponding design document?

Requirement

After the user submitted the search query, the system displays the list of results containing matching hotels.

Are there inconsistencies?

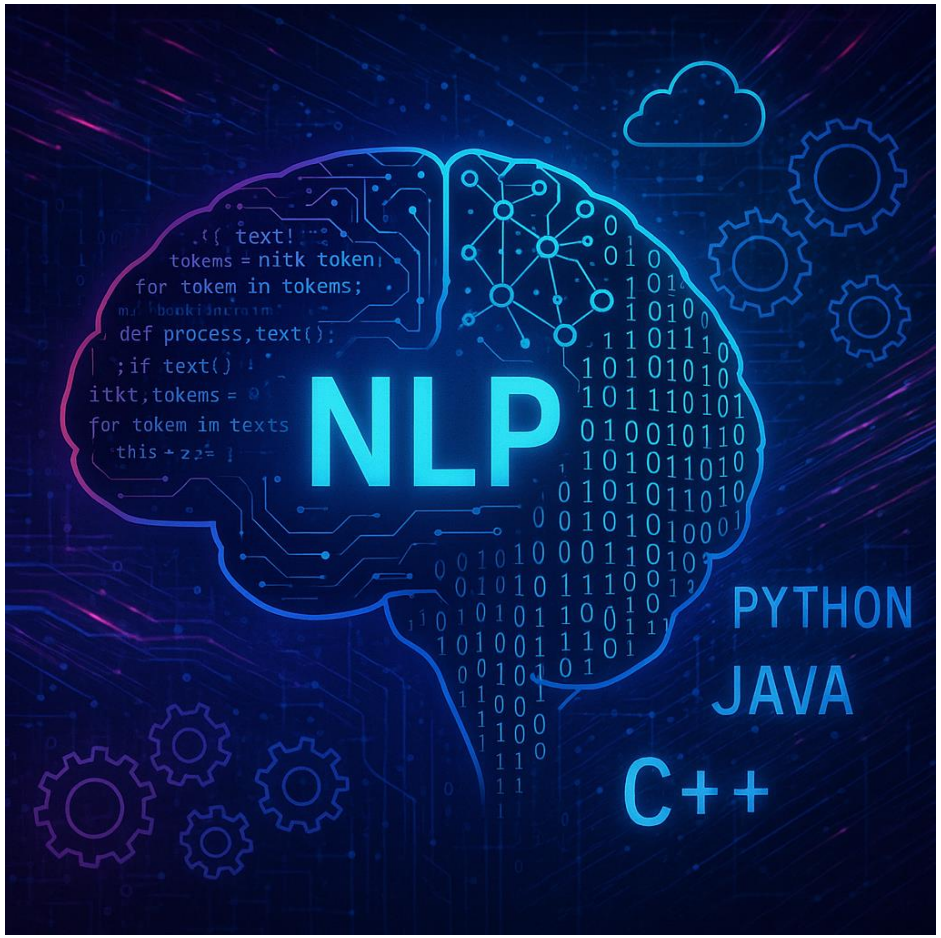
Does the requirement fulfill the quality standards?



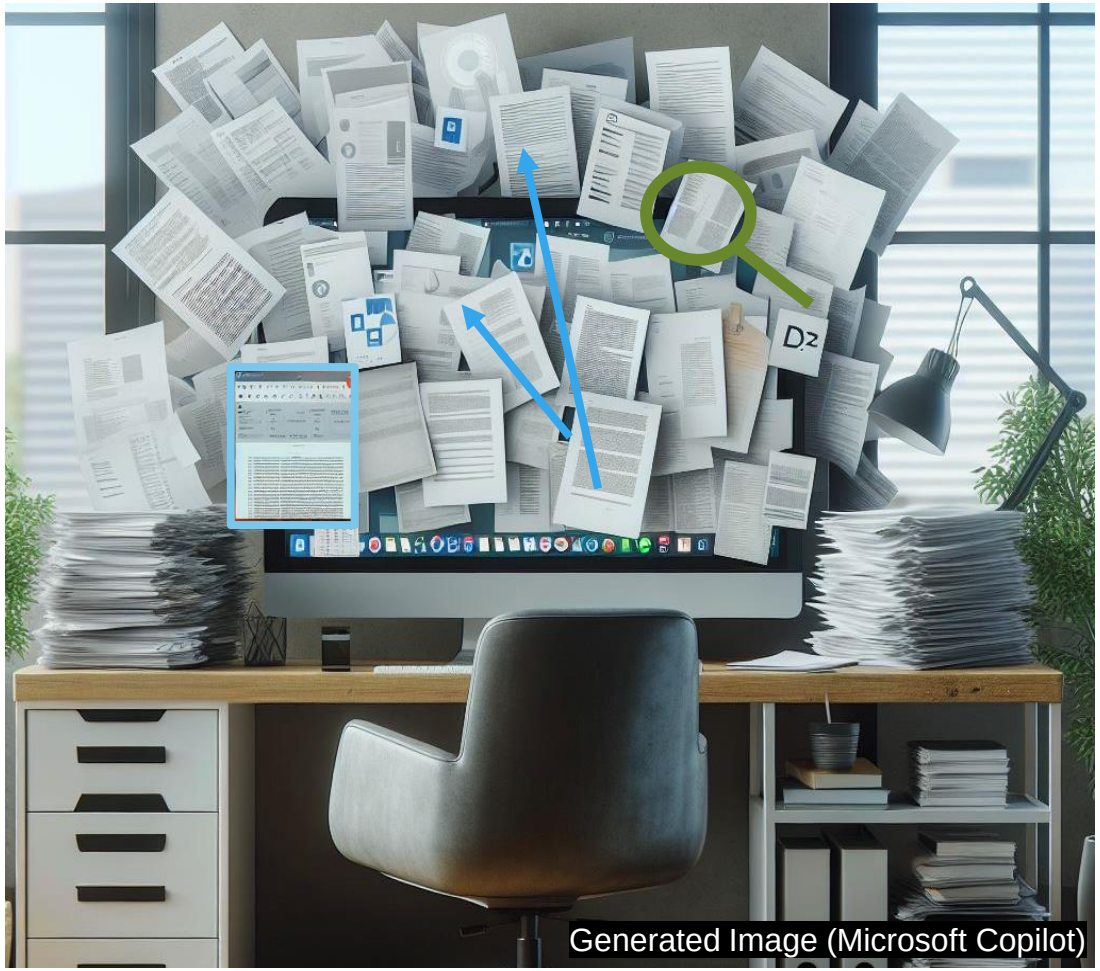
Manual processing is cumbersome. Can we automate it?

What is Natural Language Processing (NLP)?

- Applying theories and models about natural language (grammar, knowledge models, …) with computer systems, to „understand“ human language
- Investigate language with the help of algorithms and computations



Why NLP and Software Engineering?


Generated Image (Microsoft Copilot)

Text-bound Documents in Software Engineering

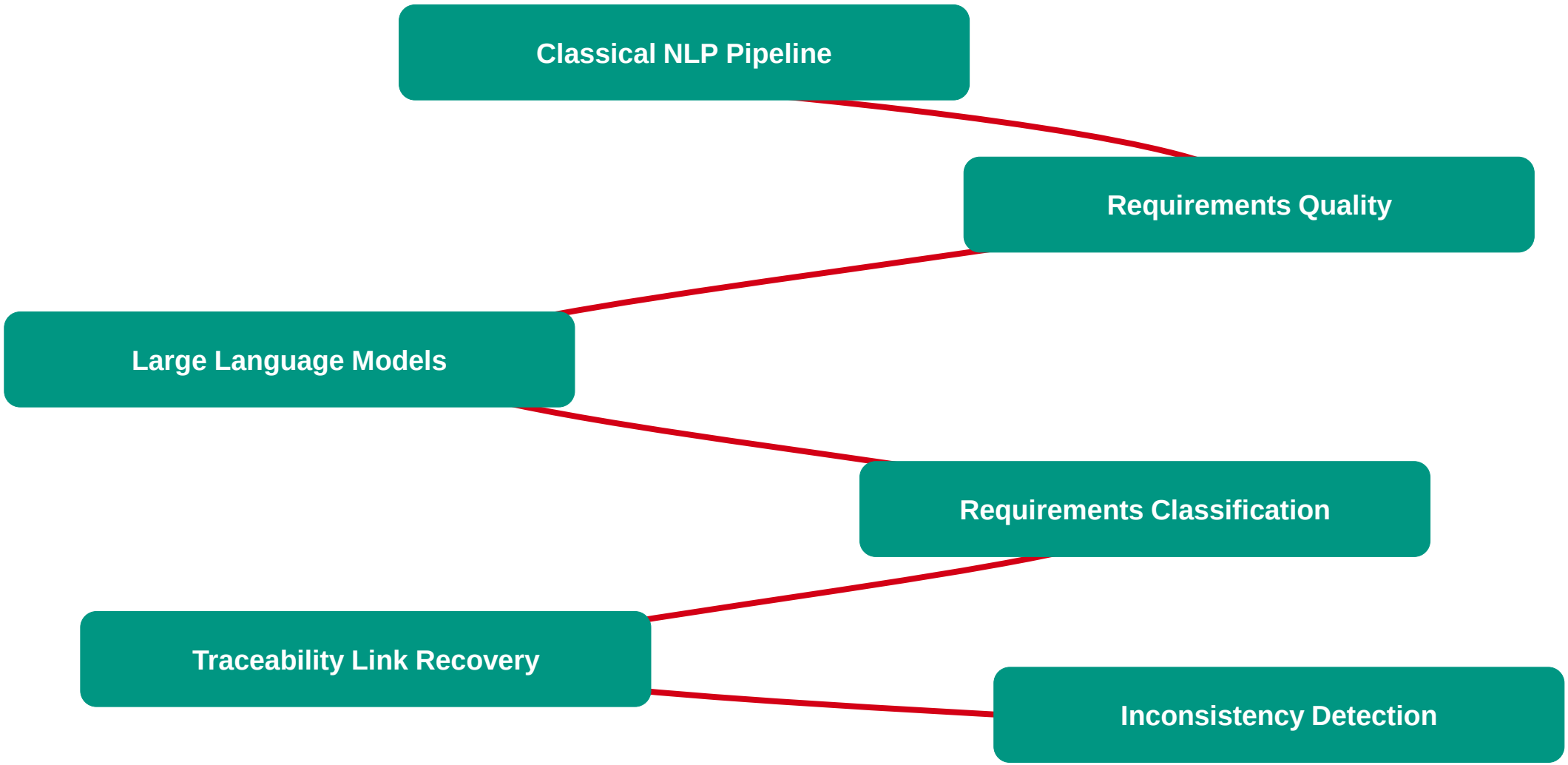
- Requirements
- Documentation
- Architecture Descriptions
- Issues
- Models
- Source code comments
- User feedback and error descriptions

…

➔ If we can automatically process these artifacts, we have a big lever!

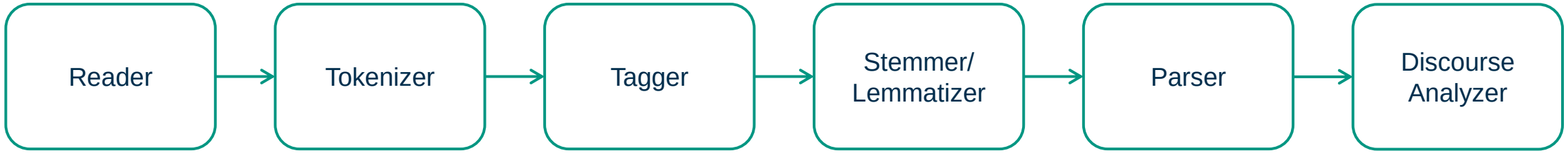
This lecture focuses on requirements, but NLP is useful in many other phases and for many other artifacts as well!

Today‘s Topics



Classical NLP Pipeline

Classical NLP-Pipeline

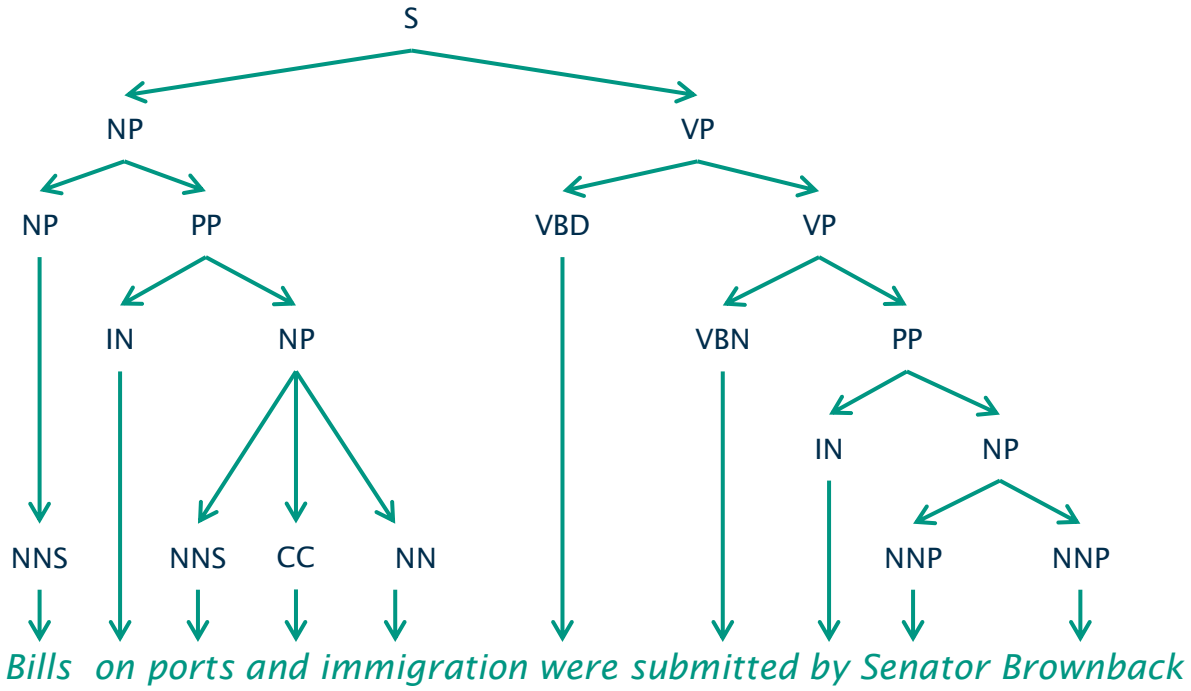


- **Tokenizer**: Create tokens, the smallest unit we process (e.g., words)
- **Tagger**: Label parts of the text (e.g., part-of-speech like noun, verb)
- **Stemmer/Lemmatizer**: Derive the base form (e.g., infinitive of verbs)
  - Stemmer: Universities → Univers; Universe → Univers
  - Lemmatizer: Universities → University; Universe → Universe
- **Parser**: Parse into, e.g., a tree (for phrases) or graph (for dependencies)
- **Discourse Analyzer**: analyze language to understand meaning, contexts, social constructs/relations etc.

Parser: Syntax Tree

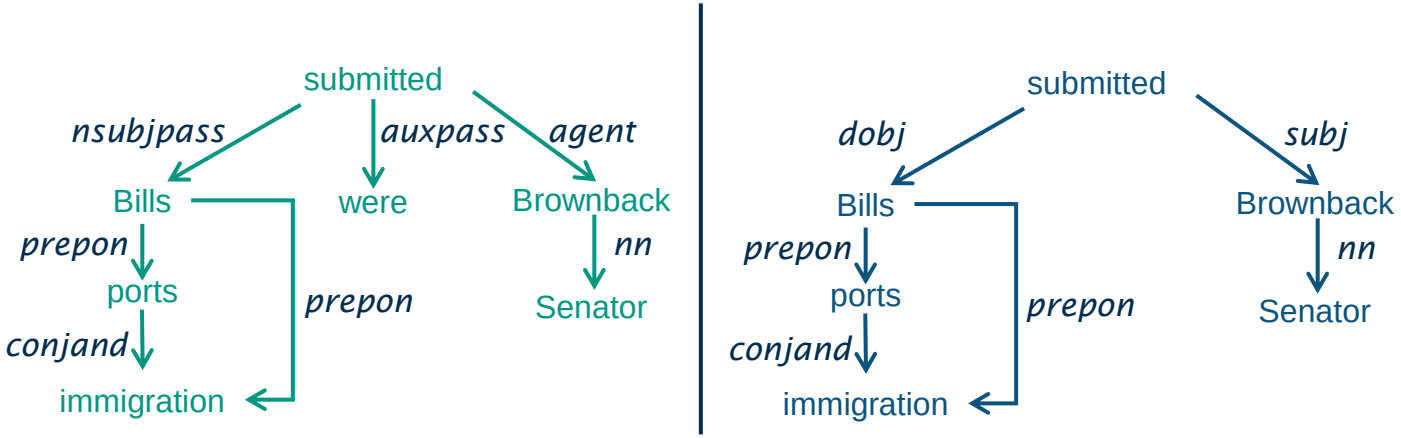
The phrase structure divides sentences into stacked constituents

S	Sentence
NP	Nominal phrase
VP	Verbal phrase
PP	Prepositional phrase
IN	Prepositon
CC	Coordinating Conjunction
NN*	Noun
VB*	Verb
.	Period



Parser: Dependency Graph

The dependency structure shows the relations between words like modifying words or argument-relations



Bills on ports and immigration were submitted by Senator Brownback.

Senator Brownback submitted bills on ports and immigration.

Requirements Quality

Requirements Quality

- Textual specifications are the foundation of the SE process
- Yet, they often have problems due to natural language like ambiguity, inconsistency etc.

- At best, these errors and problems are identified and fixed early

- **Requirements Quality Model by Fabbrini et al.**
  - Quality attributes (unambiguity, clarity, completeness, consistency) are not or barely measurable
  - Yet, indicators for problems/errors can be found objectively and are identifiable and measurable

Fabbrini et al. : The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. *Software Engineering Workshop, 2001*

Requirements Quality
High Level Properties (Quality Attributes)

Non-Ambiguity

Completeness

Consistency

Understandability

Fabbrini et al. : The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool.
Software Engineering Workshop, 2001

Requirements Quality Indicators

Non-Ambiguity:

- **Vagueness**: the sentence includes words holding inherent vagueness, i.e., words having a non uniquely quantifiable meaning (*clear, easy, strong, good, bad, useful, ...*)
- **Subjectivity**: the sentence refers to personal opinions or feeling (*similar(ly), take into account, as [adjective] as possible, …*)
- **Optionality**: the sentence contains an optional part (*possibly, eventually, if possible, if appropriate, if needed, …*)
- **Weakness**: the sentence contains a weak main verb (*could, might*)

Completeness:

- **Underspecification**: in sentence when the subject of the sentence contains a word identifying a class of objects without a specifier of this class
- Examples: flow (*data flow, control flow*, …), access (*write access, remote access, authorized access*, …), testing (*functional testing, structural testing, unit testing*, …), etc.

Fabbrini et al. : The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. *Software Engineering Workshop, 2001*

Requirements Quality Indicators

Consistency:

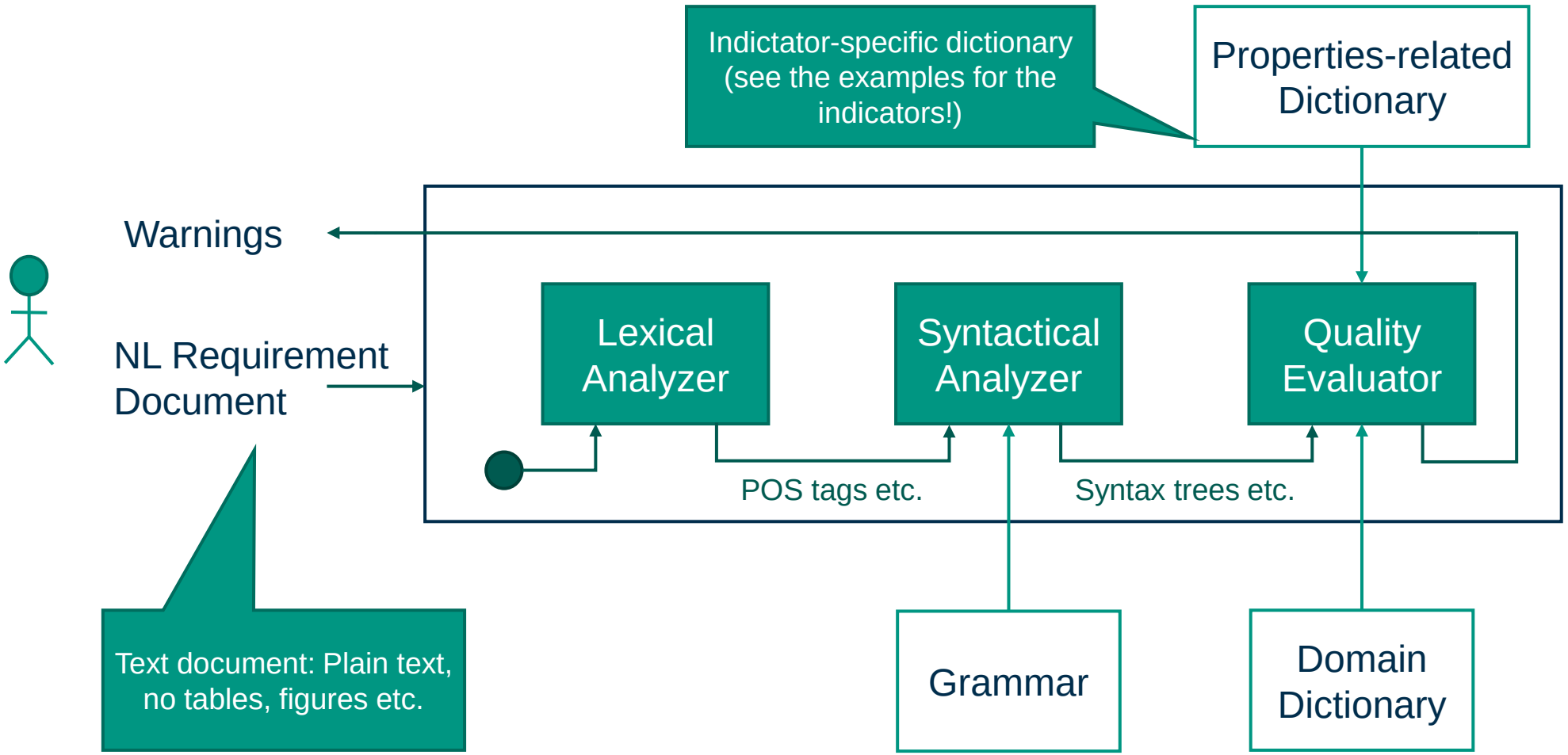
- **Underreference**: a sentence contains explicit references to 1) not numbered sentences, 2) documents that then are not referenced, 3) entities not defined nor described

Understandability:

- **Multiplicity**: sentence has more than one main verb or more than one direct or indirect complement that specifies its subject
- **Implicity**: the subject is generic rather than specific (*this, these, it, they, previous, next, above, below*)
- **Unexplanation**: sentence contains acronyms not explicitly and completely explained within the document itself

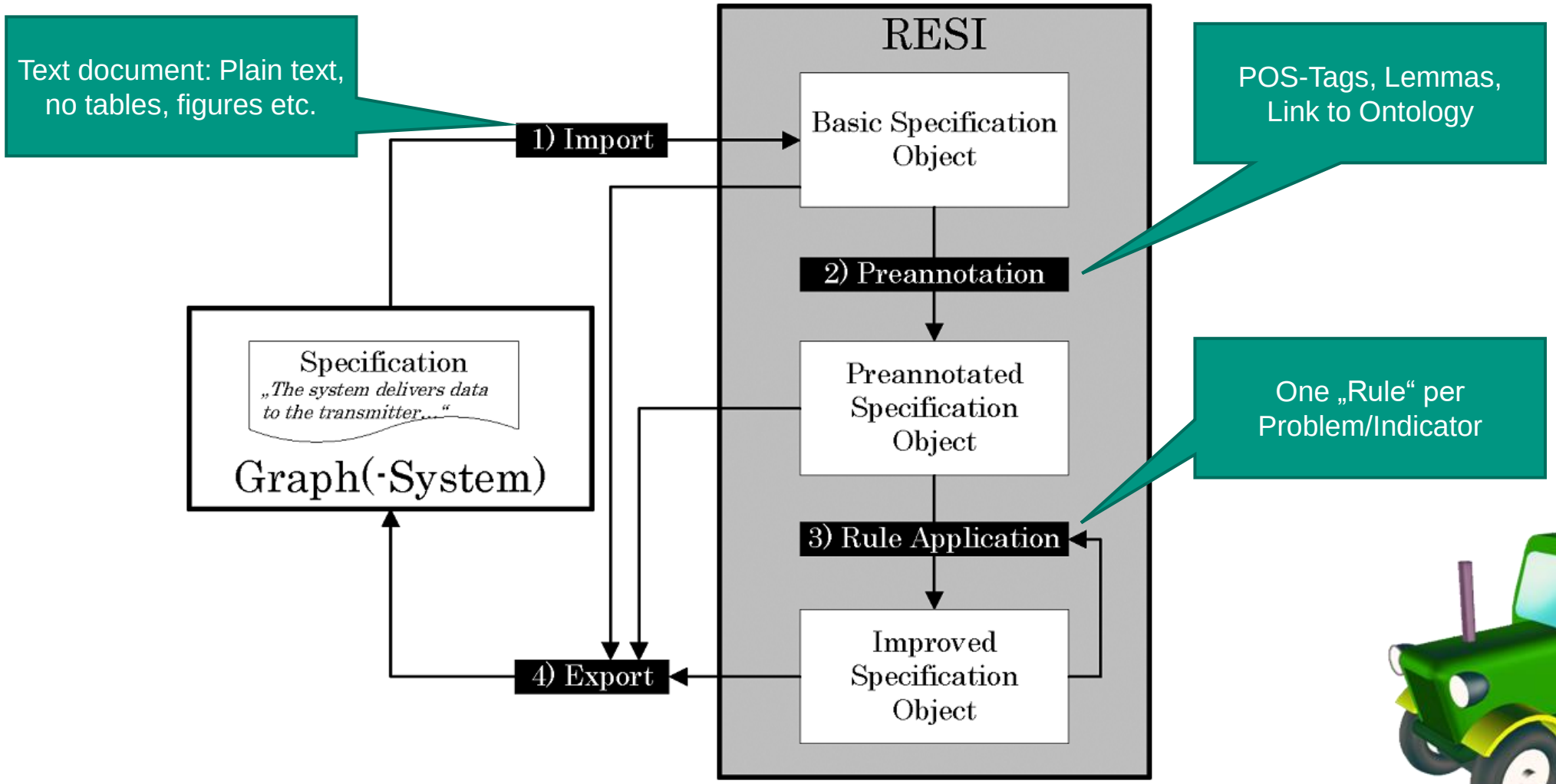
Fabbrini et al. : The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. *Software Engineering Workshop, 2001*

QuARS: Detect indicators for low quality requirements



Fabbrini et al. : The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. *Software Engineering Workshop, 2001*

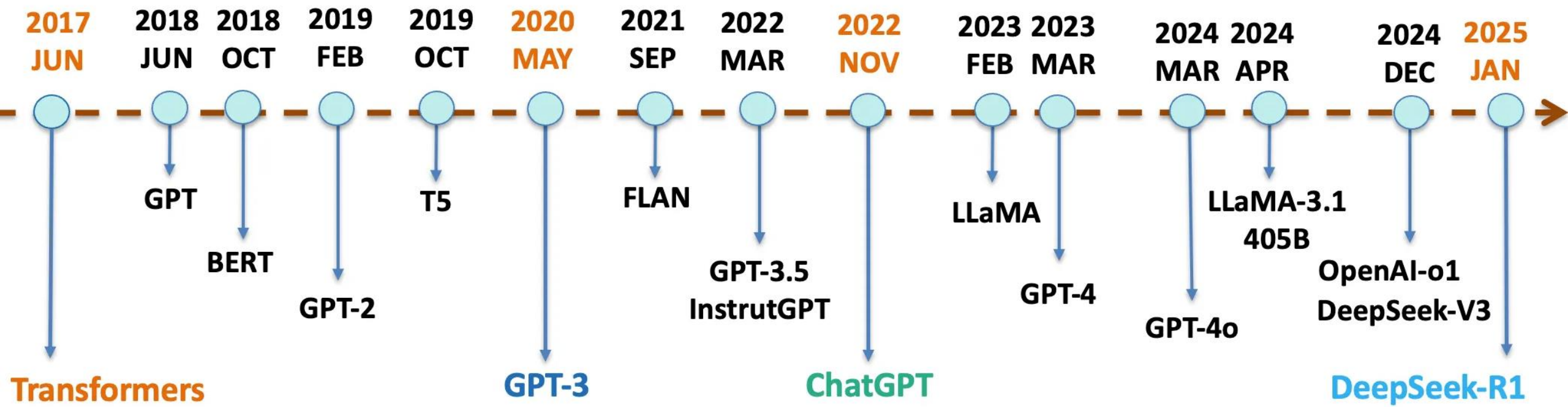
RESI: Requirements Engineering Specification Improver



Körner & Brumm : RESI - A Natural Language Specification Improver. *Proceedings of the IEEE ICSC 2009*

1. Which LLMs do you know?
2. When was the first LLM released?

Large Language Models



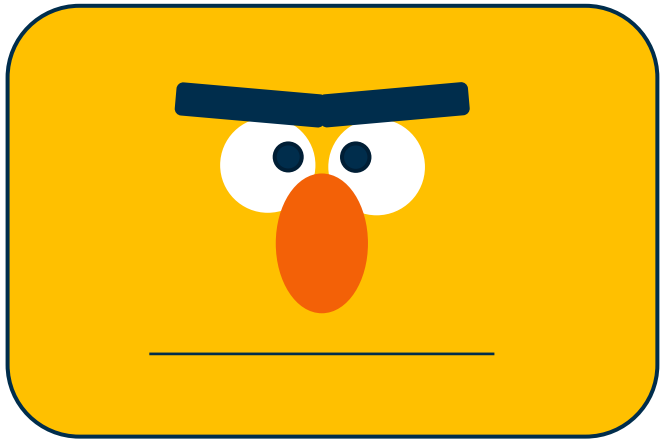
Large Language Models

Large Language Models (LLMs)

- An LLM is an approach to predict words/sentences
- Core idea: „fill the gap“ or „continue the text“

The dog __ the homework

I like the colors red and __



Language Model BERT

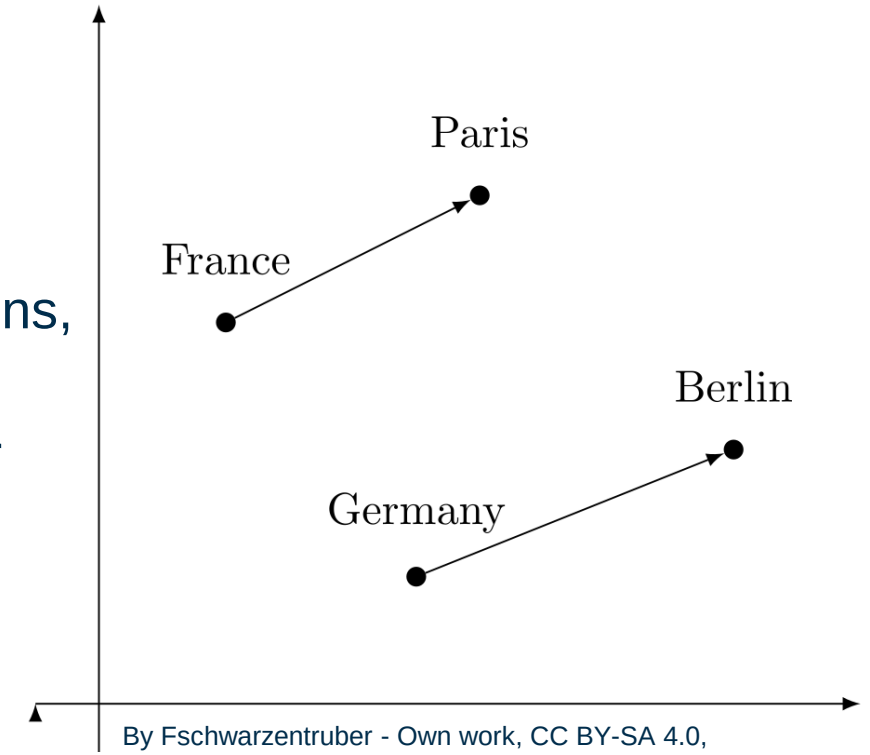
The dog ate the homework

I like the colors red and blue

- The LLM estimates the probability of a (word) sequence and *promises* a semantic understanding
- Many capabilities of LLMs (e.g., context-awareness) are useful for software engineering!

Brief Introduction to Word Embeddings

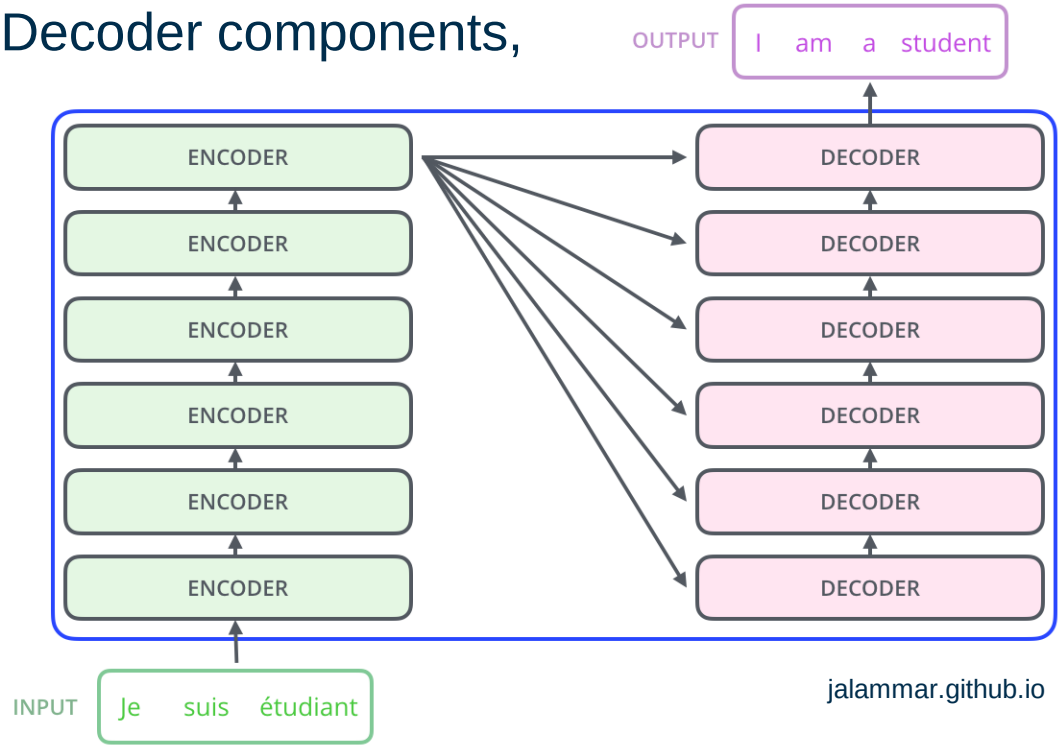
- Presentation of words as vectors
  → Embedding in a vector space
- Goal: “short“ but meaningful vectors
- Why?
  - Vectors allow us to do arithmetic operations (transformations, distance, …)
  - Good embeddings even contain temporal, geographical or spatial relations
    (e.g., city-country)
- Foundation of all LLMs



By Fschwarzentruber - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=162969550

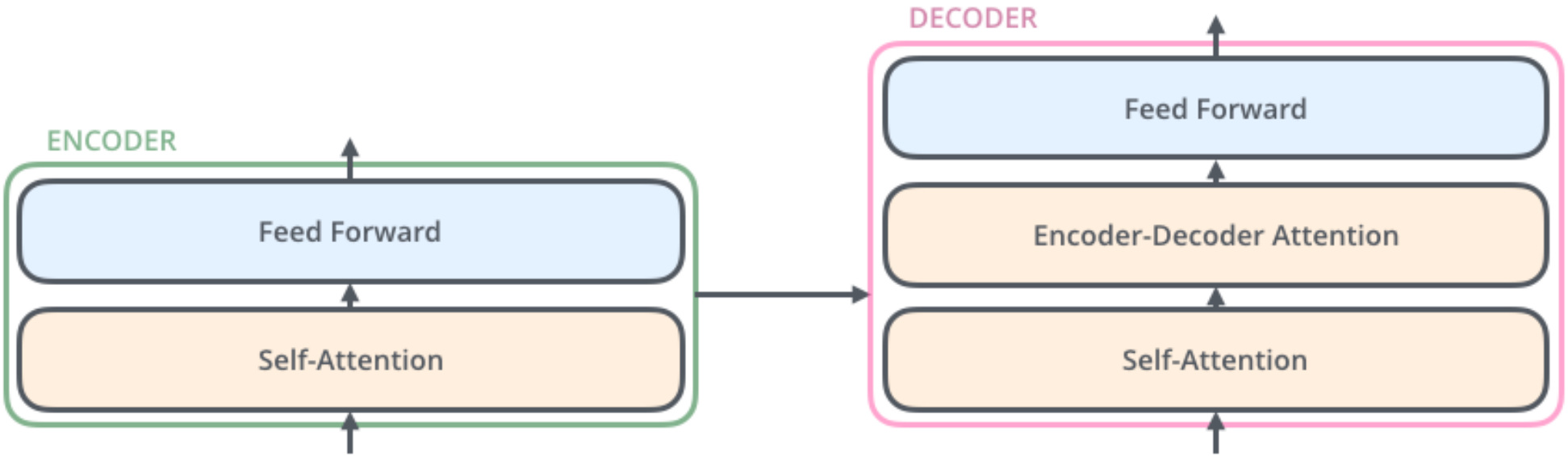
Transformer-Architecture: What is it?

- Core architecture of almost all LLMs like BERT, GPT, Llama, …
- Originally developed for translation, like French to English
- Consists of stacks of Encoder components and Decoder components, but LLMs not necessarily use both
  - Encoder models like BERT
  - Decoder models like GPT
  - Encoder-Decoder models like T5



Transformer: Encoder and Decoder

- Key part: (Self-) Attention Layer
  - Assesses which part of the input should be focused? Which/How do parts of the input relate?
  - Fundamental for the capabilities and success of LLMs
- Feed-Forward Neural Network: Main training layers



jalammar.github.io

How to train LLMs?

- Usually multiple training tasks
- Training Task 1: Masked Language Modelling
  - Mask a part of the text (BERT: 15%) and let the LLM predict the missing word
  - For resilience: exchange some words randomly
- Training Task 2: Next Sentence Prediction: Predict if a sentence follows another sentence
- Reinforcement Learning with Human Feedback (RLHF)
  - „Optimization“ of the output by incorporating human feedback
  - reward function is trained with human feedback

Using LLMs for your task

Fine-Tuning

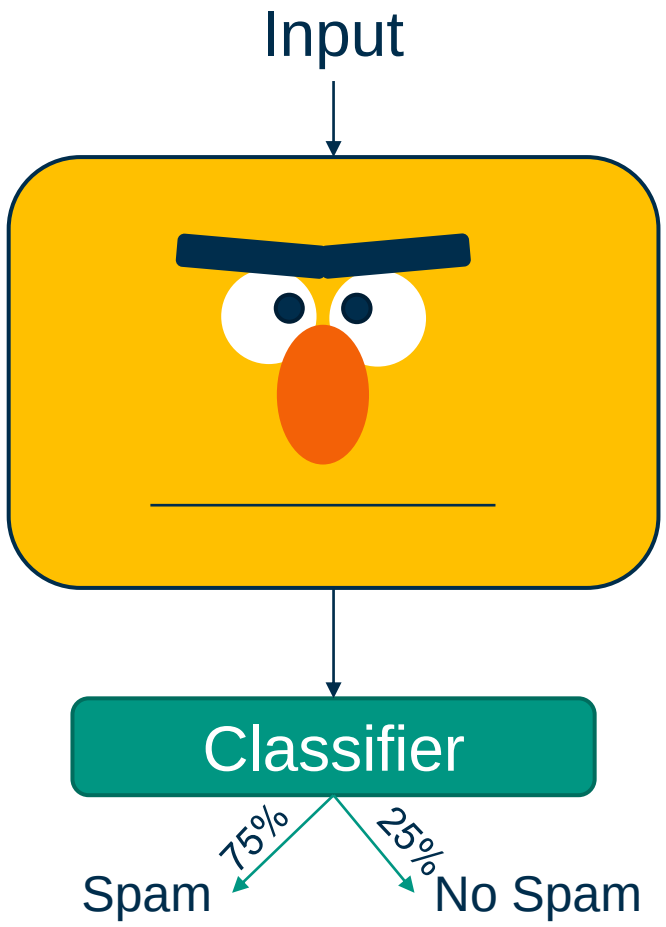
- Training based on a specific task using a labeled dataset, e.g., for spam detection
- Two main kind:
  - Training of a new, simple neural network (layer) with the outputs of the LLM
  - Continue training of the LLM‘s the existing layers

Zero-Shot

- Give the LLM the task and hope that the task is known to it

One/Few-Shot

- Give the LLM the task and additionally some labeled data
- LLM might infer from the examples



Retrieval-augmented Generation (RAG)

- Preparation: textual knowledge is embedded as vectors and stored in a database (vectorstore)
- For each query:
  1. Embed the query (as vector)
  2. Retrieve similar sources based on vector similarity (→ retrieval)
  3. Provide to the LLM the query together with retrieved sources

→ Advantage: LLM uses the provided knowledge and hallucinates less

Requirements Classification

Task: Requirements Classification

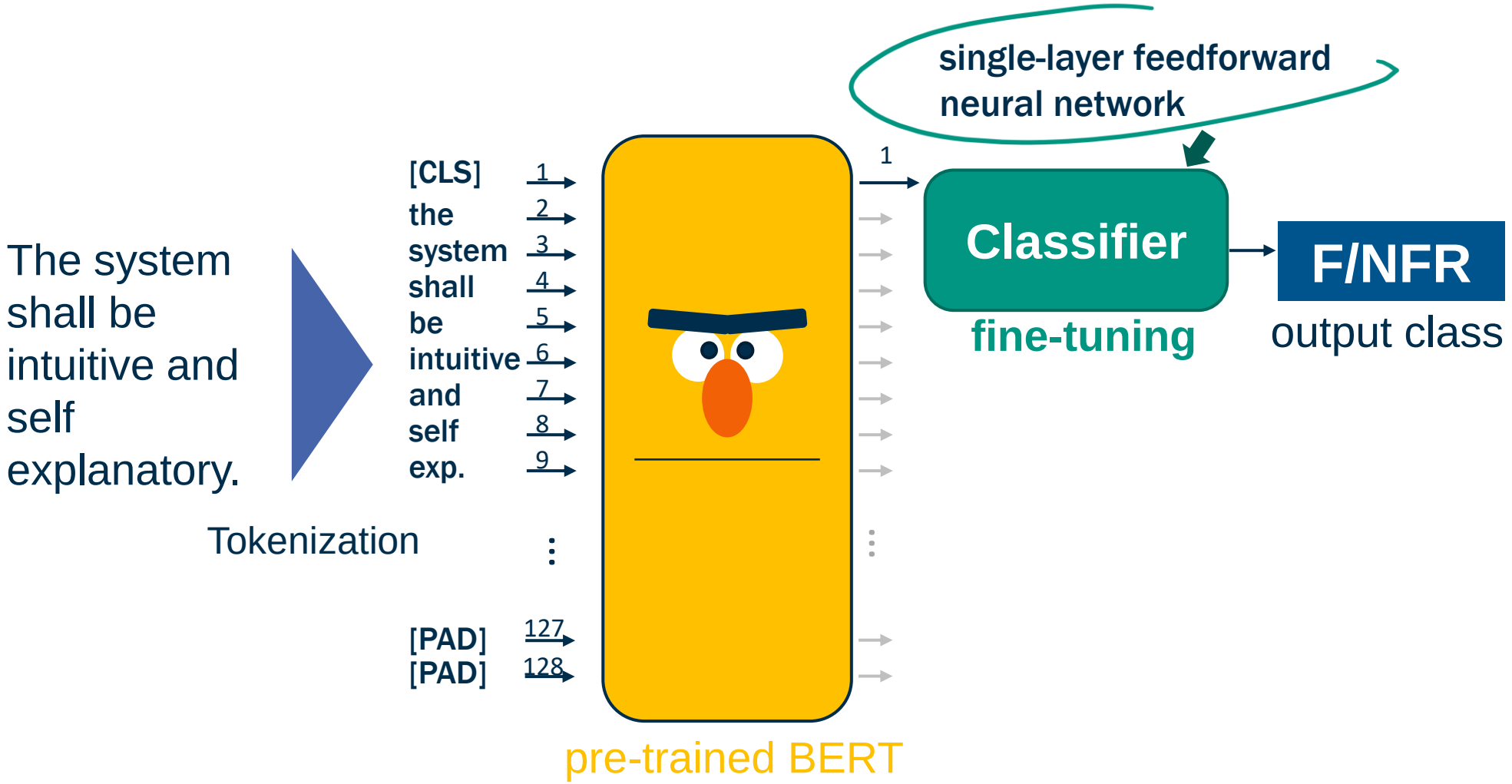
- Requirements can be classified in various classes
  - Functional vs. Non-functional
  - Sub-categories of non-functional requirements (Usability, Security, Performance, ...)
  - Functional and quality aspects (with overlap)
  - Sub-categories of functional requirements (Function, Data, Behavior)

- Manual classification is cumbersome

- Idea: use BERT‘s transfer learning capabilities for automated requirements classification
  → **No**nfunctional and Functional **R**equirements Classification using **BERT** (**NoRBERT**)

T. Hey, J. Keim, A. Koziolek, and W. F. Tichy: NoRBERT: Transfer Learning for Requirements Classification
2020 IEEE 28th International Requirements Engineering Conference (RE)

NoRBERT: Requirements Classification



T. Hey, J. Keim, A. Koziolek, and W. F. Tichy: NoRBERT: Transfer Learning for Requirements Classification
2020 IEEE 28th International Requirements Engineering Conference (RE)

How good is it? Evaluation!

There are two major kinds of evaluation of (automated) tooling

- **Intrinsic**: How well does the approach solve the task, measured with the direct results?
- **Extrinsic**: How does the approach influence another task that uses the approach?

Classification metrics

- **Precision**: How many of the predicted positive results are actually correct?
- **Recall**: How many of the actual positive cases were correctly identified?
- **F_1-Score**: Harmonic mean between precision and recall

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$F_1 - \text{Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

		Reality: True	Reality: False
Classification	True	True Positive	False Positive
	False	False Negative	True Negative

How to Evaluate? Experiment Design

Given: Benchmark Dataset of 15 projects

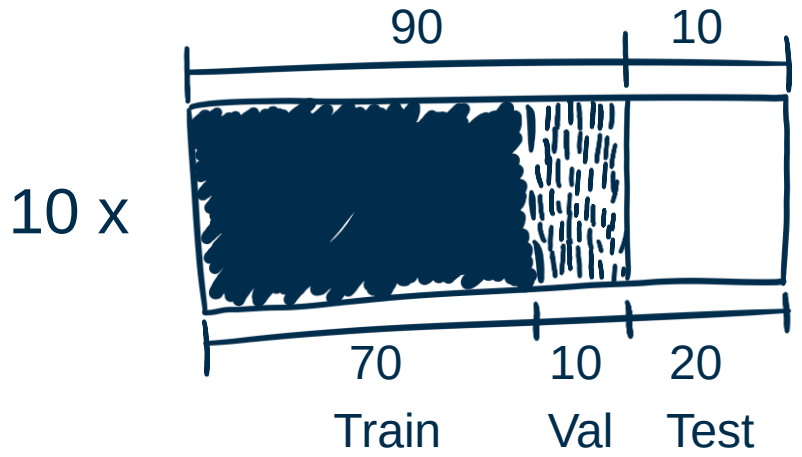
How to split the data?

- Classical: 70/10/20 split (train/validate/test)
- K-fold cross validation
- Project-specific folding strategies
  - e.g. Leave-one-project-out (LoPo)

- What to take depends on the usage scenario!

Only 20% of data tested (biased)

Project-specific information seen during training



Evaluation Results: Functional/Non-functional

	F/NFR	F_1-score
10-fold	Naïve Bayes	0.90
	WE + CNN	0.92
	SVM	0.93
	NoRBERT (Base)	0.92
	NoRBERT (Large)	0.93
	DBGAT	**0.94**
LoPo	**NoRBERT (Base)**	0.74
	NoRBERT (Large)	**0.83**

T. Hey, J. Keim, A. Koziolek and W. F. Tichy: NoRBERT: Transfer Learning for Requirements Classification
2020 IEEE 28th International Requirements Engineering Conference (RE)

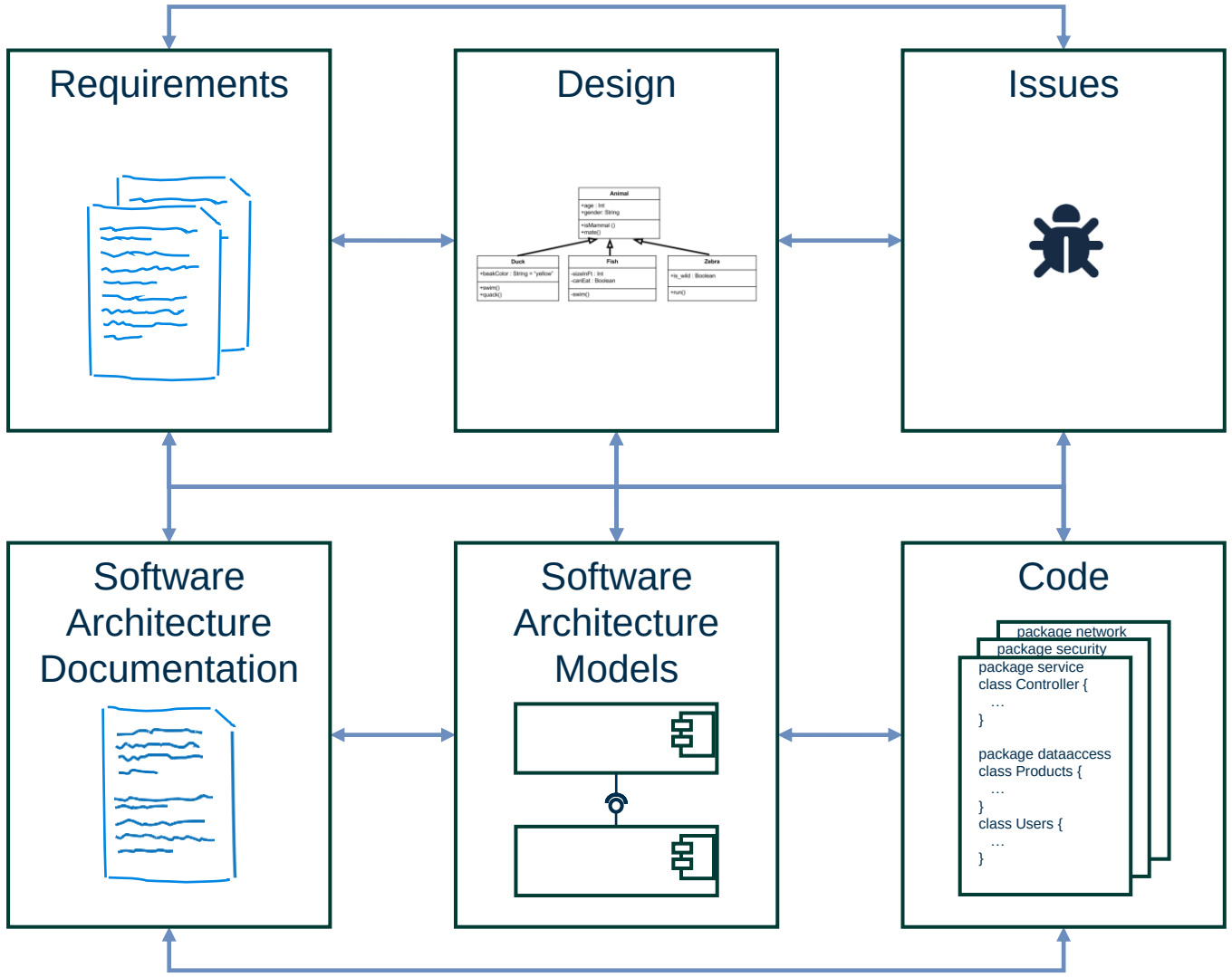
Evaluation Results: Functional/Non-functional

Approach (Parameters)	F (255) P	F (255) R	F (255) F_1	NFR (370) P	NFR (370) R	NFR (370) F_1
10-fold						
K. & M. (word features w/o feat. sel.)	**.92**	.93	**.93**	.93	.92	.92
K. & M. (500 best word features)	**.92**	.79	.85	.82	.93	.87
K. & M. (500 best features)	.88	.87	.87	.87	.88	.87
A. et al. (unprocessed data)	.84	.93	.88	.95	.88	.91
A. et al. (processed data)	.90	**.97**	**.93**	**.98**	.93	**.95**
D. & F. (word2vec, ep.=100, f.=50)	—	—	—	.93	.92	.92
NoRBERT (base, ep.=10)	.91	.90	.90	.93	.94	.93
NoRBERT (base, ep.=10, ES, US)	.88	.88	.88	.92	.92	.92
NoRBERT (base, ep.=10, ES, OS)	.91	.86	.88	.91	.94	.92
NoRBERT (base, ep.=16)	.89	.88	.89	.92	.93	.92
NoRBERT (large, ep.=10, OS)	**.92**	.88	.90	.92	**.95**	.93
p-fold						
NoRBERT (base, ep.=10)	.85	.51	.64	.74	.94	.82
NoRBERT (large, ep.=16)	**.89**	**.61**	**.73**	**.78**	**.95**	**.86**
loPo						
NoRBERT (base, ep.=10, ES)	**.88**	.50	.64	.73	**.95**	.83
NoRBERT (large, ep.=10, US)	.87	**.71**	**.78**	**.82**	.93	**.87**

T. Hey, J. Keim, A. Koziolek and W. F. Tichy: NoRBERT: Transfer Learning for Requirements Classification
2020 IEEE 28th International Requirements Engineering Conference (RE)

Traceability Link Recovery

Traceability Link Recovery (TLR)



Trace links are evidently useful for

- Software Maintenance
- Bug Localization
- Change Impact Analysis
- System Security
- …

Artifact
Relation

Fine-grained Traceability Link Recovery (FTLR)

Step 1

Division in fine-grained artifact elements

Step 2

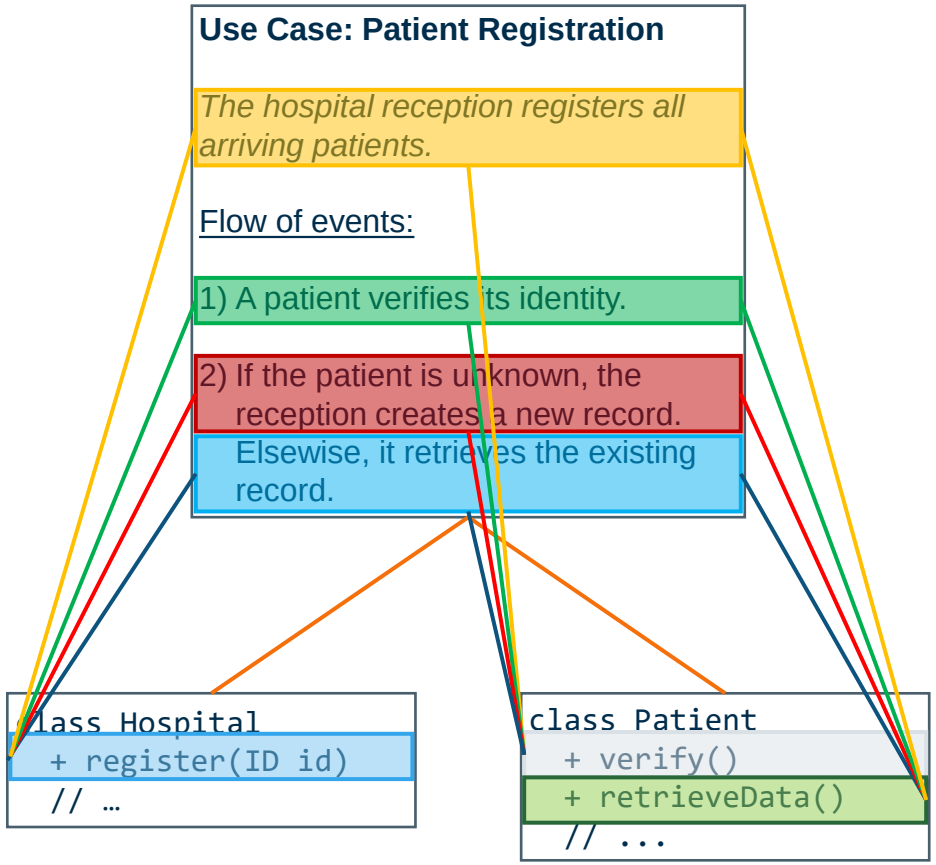
Representation with fastText embeddings

Step 3

Similarity comparisons using word movers distance

Step 4

Aggregation using majority voting

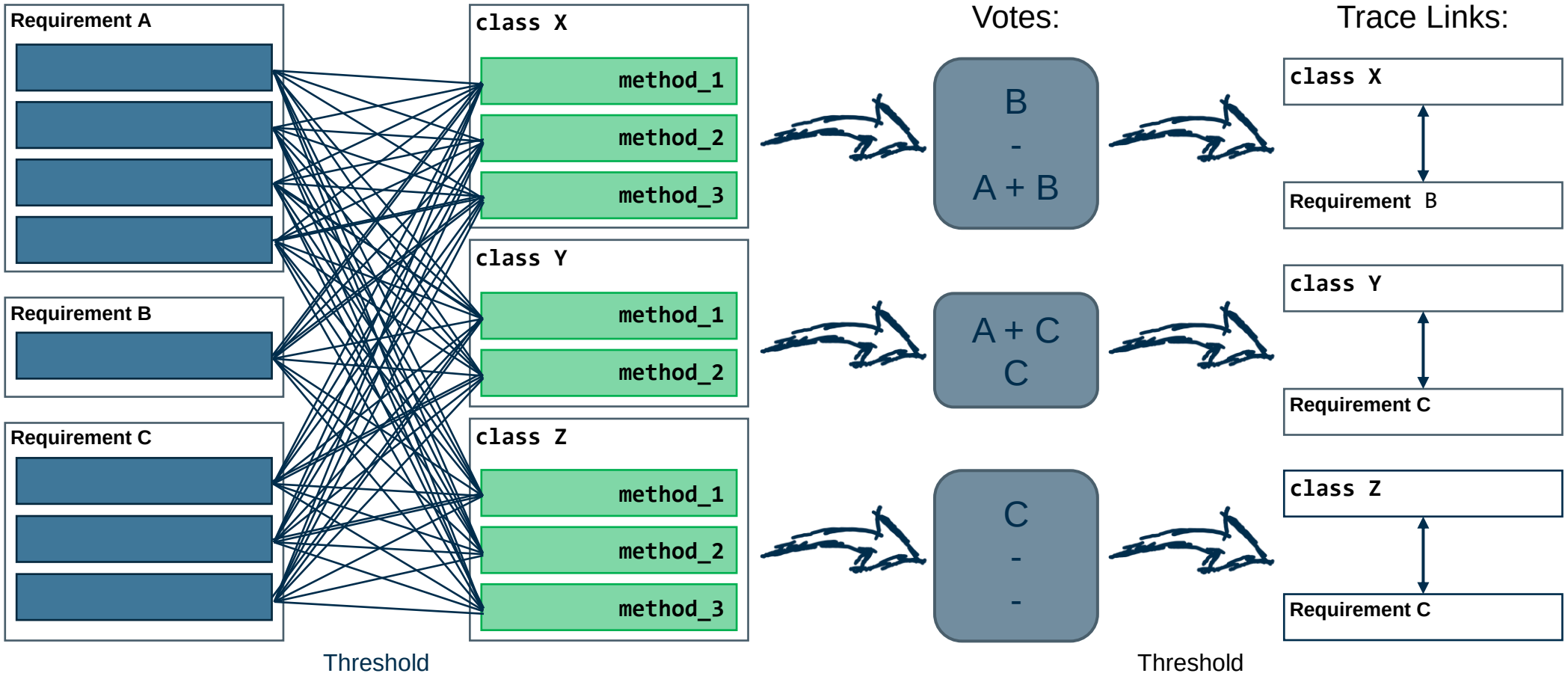


T. Hey, F. Chen, S. Weigelt, and W. F. Tichy: Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations
2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)

FTLR

Step 4

Aggregation using majority voting



T. Hey, F. Chen, S. Weigelt, and W. F. Tichy: Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations
2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)

FTLR: Further Improvements

- We try to map mostly functional requirements to code. How can we improve FTLR?

→ Filtering requirements using classification results of NoRBERT!

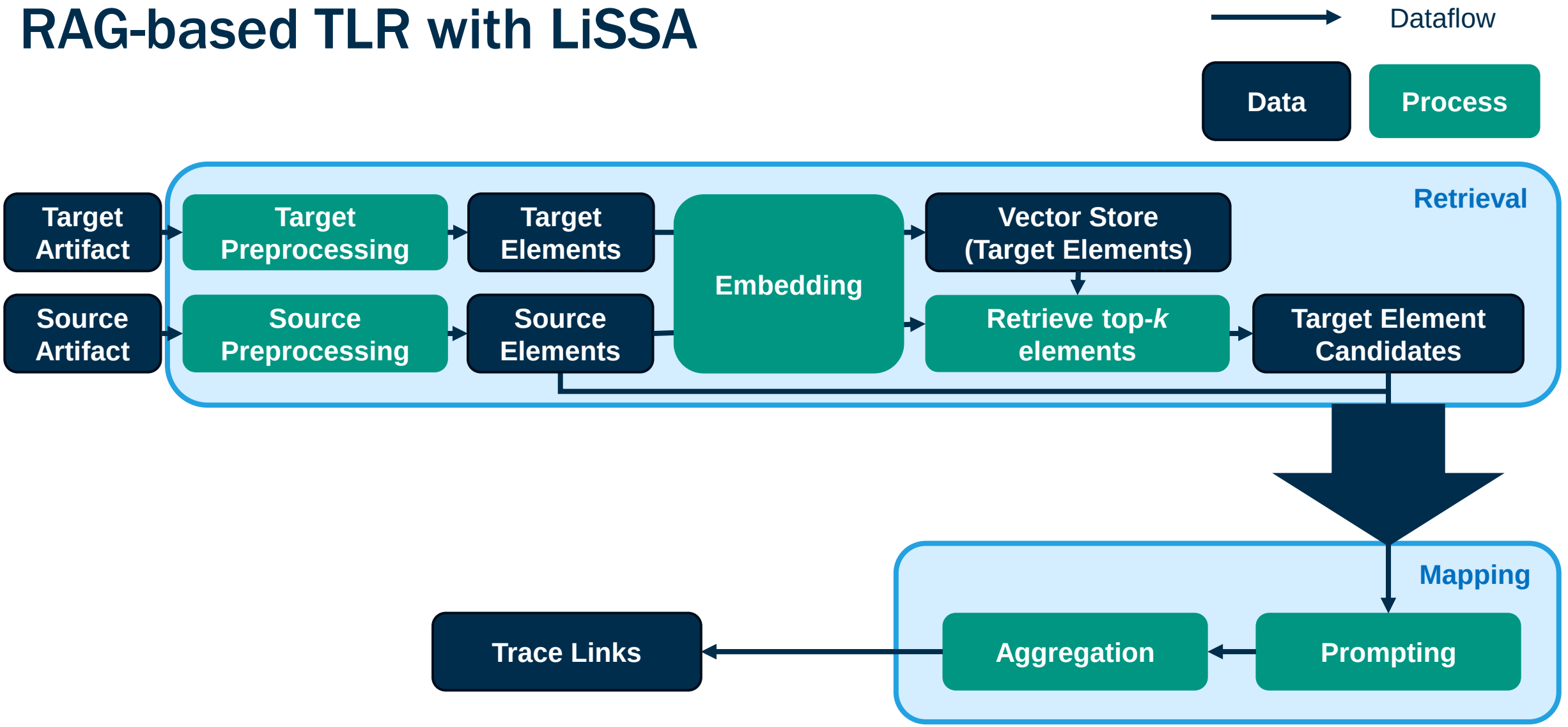
- Filtering for Function (F) concerns and/or User-Related (UR) requirement elements

Stands for using method comments (mc) and call dependencies (cd)

Thresh.	Filter	MC P	MC R	MC F_1	MC+CD P	MC+CD R	MC+CD F_1
ORG	—	.287	.423	.327	.291	.466	.344	.294	.459	.345
	F	.313	.399	.335	.311	.445	**.353**	.315	.436	**.353**
	UR	.373	.312	.302	.363	.334	.317	.367	.330	.318
	F+UR	.389	.302	.299	.365	.319	.308	.368	.315	.308
	F_{gold}	.336	.394	.349	.334	.443	.369	.339	.435	.370
	UR_{gold}	.371	.351	.342	.368	.389	.363	.370	.383	.360
	$F+UR_{gold}$	.379	.342	.342	.374	.385	.365	.377	.380	.363
OPT	—	.290	.532	.372	.316	.506	.378	.298	.527	.375
	F	.318	.509	**.388**	.337	.466	**.388**	.328	.473	.386
	UR	.311	.487	.371	.332	.452	.379	.329	.453	.378
	F+UR	.315	.472	.369	.329	.446	.375	.328	.445	.375
	F_{gold}	.359	.474	.405	.342	.515	.404	.344	.506	.402
	UR_{gold}	.346	.480	.394	.359	.462	.395	.364	.456	.392
	$F+UR_{gold}$	.362	.470	.396	.366	.444	.396	.358	.461	.394

Hey, T., Keim, J., & Corallo, S. (2024, June). Requirements Classification for Traceability Link Recovery. *2024 IEEE 32nd International Requirements Engineering Conference (RE)*

RAG-based TLR with LiSSA



Fuchβ *et al.*: LiSSA: Toward Generic Traceability Link Recovery Through Retrieval- Augmented Generation
2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

LiSSA: Evaluation

KISS (P1):
Question: Here are two parts of software development artifacts.
{source_type}: '''{source_content}'''
{target_type}: '''{target_content}'''
Are they related?
Answer with 'yes' or 'no'.

Chain-of-Thought (P2):
Below are two artifacts from the same software system. Is there a traceability link between (1) and (2)?
Give your reasoning and then answer with 'yes' or 'no' enclosed in <trace> </trace>.
(1) {source_type}: '''{source_content}'''
(2) {target_type}: '''{target_content}'''

COMPARISON OF THE AVERAGE AND WEIGHTED AVERAGE F_β-SCORES USING GPT-4O AND GPT-4O MINI FOR REQUIREMENT-TO-CODE TLR USING TOP-20 RETRIEVAL

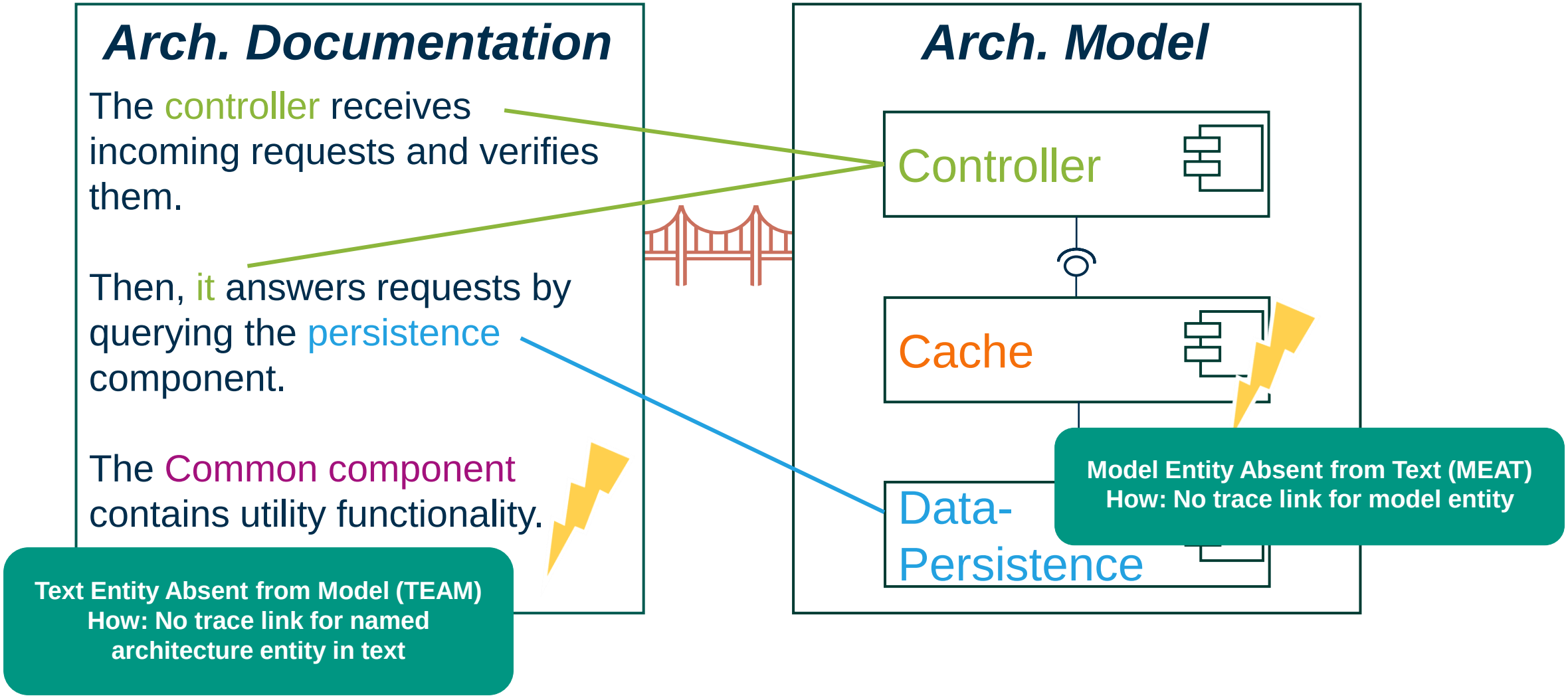
Approach		Avg.				w. Avg.			
		GPT-4o		GPT-4o mini		GPT-4o		GPT-4o mini	
Prep.	**Cls.**	F_1	F_2	F_1	F_2	F_1	F_2	F_1	F_2
None / None	No	.230	.332	.230	.332	.249	**.332**	.249	.332
	P1	.312	**.369**	.247	.345	.288	.319	.260	**.334**
	P2	**.322**	.368	**.295**	**.356**	**.299**	.319	**.282**	.315
Sentence / Chunk(200)	No	.203	.305	.203	.305	.235	**.332**	.235	.332
	P1	.242	.296	.210	.309	.256	.288	.241	.331
	P2	.242	.290	.236	.317	.257	.279	.261	.325
	+Art.	.261	.327	.243	.318	.277	.320	.263	.314
Sentence / Method	No	.198	.283	.198	.283	.243	.321	.243	.321
	P1	.257	.294	.211	.292	.285	.299	.256	.325
	P2	.257	.297	.232	.295	.289	.305	.268	.307
	+Art.	.260	.309	.238	.296	.288	.312	.267	.298

Baselines	F_1	F_2	F_1	F_2
VSM_{OPT}	.282	.256	.283	.257
LSI_{OPT}	.285	.270	.285	.268
FTLR	.278	.300	.273	.277
$FTLR_{OPT}$	.303	.327	.294	.329

Fuchβ *et al.*: LiSSA: Toward Generic Traceability Link Recovery Through Retrieval- Augmented Generation
2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

Inconsistency Detection

Inconsistency Detection using Trace Links with ArDoCo



J. Keim, S. Corallo, D. Fuchß and A. Koziolek: Detecting Inconsistencies in Software Architecture Documentation Using Traceability Link Recovery, *2023 IEEE 20th International Conference on Software Architecture (ICSA)*

ArDoCo: Evaluation

Model Entity Absent from Text (MEAT)

Text Entity Absent from Model (TEAM)

	Weighted Avg.		
	Precision	Recall	F1
MEAT	.87	.88	.86
TEAM	.39	.64	.34

J. Keim, S. Corallo, D. Fuchß and A. Koziolek: Detecting Inconsistencies in Software Architecture Documentation Using Traceability Link Recovery, *2023 IEEE 20th International Conference on Software Architecture (ICSA)*

Summary

- Natural Language Processing is key for automated Software Engineering
  - Many artifacts are natural language texts
  - Automatically processing these artifacts gives us a big lever
- There are plenty chances and opportunities to improve SE with NLP

- Moreover, (software engineering) research can be improved (Meta-research)

- Focuses at our chairs: Traceability Link Recovery, Inconsistency Detection, Knowledge extraction, Inter-symbolic AI, Meta-research, and more!

Interested in NLP & SE?

- The lecture **Natural Language Processing and Software Engineering** looks into this topic more thoroughly
- There are also topics in **seminars** and **practical courses**
- **Bachelor‘s** and **Master‘s theses** topics
- Some open positions for student workers