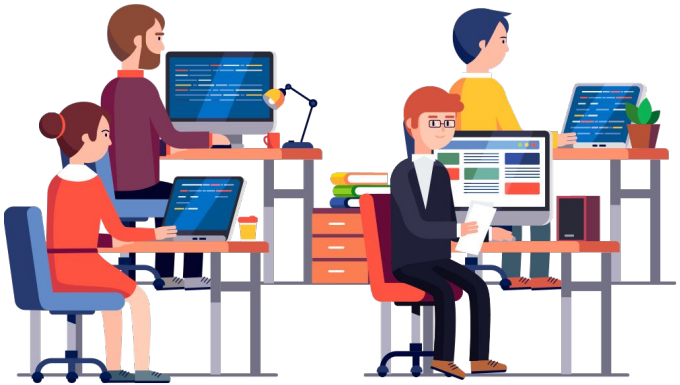
Learning to Update Natural Language Comments Based on Code Changes

Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, Raymond J. Mooney
The University of Texas at Austin

Source Code Comments



Developers communicate via comments:

- Usage
- Implementation
- Error cases
- ...

Source Code Comments

- Code is constantly evolving
- Failure to update comments upon code changes can lead to confusion and bugs

```
/**@return the highest value from the list of scores*/
```

```
  public int getScore() {
-       return Collections.max(scores);
+       return Collections.min(scores);
  }
```

Our Approach

```
/**@return double the roll euler angle.*/
public double getRotX() {
    return mOrientation.getRotationX();
}
```

Our Approach

```
/**@return double the roll euler angle.*/
public double getRotX() {
    return mOrientation.getRotationX();
}
```

```
public double getRotX() {
    return Math.toDegrees(mOrientation.getRotationX());
}
```

Our Approach

```
/**@return double the roll euler angle.*/
public double getRotX() {
    return mOrientation.getRotationX();
}
```

```
/**@return double the roll euler angle in degrees.*/
public double getRotX() {
    return Math.toDegrees(mOrientation.getRotationX());
}
```

Our Approach

C_{old}

```
/**@return double the roll euler angle.*/
```

M_{old}

```
public double getRotX() {
    return mOrientation.getRotationX();
}
```

C_{new}

```
/**@return double the roll euler angle in degrees.*/
```

M_{new}

```
public double getRotX() {
    return Math.toDegrees(mOrientation.getRotationX());
}
```

Given (C_{old}, M_{old}) and M_{new}, produce C_{new}.

Our Approach

C_{old}

```
/**@return double the roll euler angle.*/
```

M_{old}

```
public double getRotX() {
    return mOrientation.getRotationX();
}
```

C_{new}

```
/**@return double the roll euler angle in degrees.*/
```

M_{new}

```
public double getRotX() {
    return Math.toDegrees(mOrientation.getRotationX());
}
```

To address this task, we propose an edit model.

Given (C_{old}, M_{old}) and M_{new}, produce C_{new}.

Why Edits?

- When developers edit comments, they don’t delete the existing comment and start from scratch
- They edit only parts of the comment that are relevant to the code changes

Why Edits?

- When developers edit comments, they don't delete the existing comment and start from scratch
- They edit only parts of the comment that are relevant to the code changes

Learn to edit C_{old} ->C_{new} rather than generate C_{new}

Why Edits?

Comment edits

Implicitly learning these edits by directly generating C_{new} using C_{old} risks learning to copy, so we explicitly define NL edits.

Code edits

To better correlate code changes with NL edits and also prevent having the model implicitly learn these changes them from M_{old} and M_{new}, we explicitly define code edits.

Representing Edits

M_{old}

```
public double getRotX() {
    return mOrientation.getRotationX();
}
```

M_{new}

```
public double getRotX() {
    return Math.toDegrees(mOrientation.getRotationX());
}
```

M_{edit}

```
<Keep> public double getRotX() <KeepEnd>
<Insert> Math.toDegrees( <InsertEnd>
<Keep> mOrientation.getRotationX() <KeepEnd>
<Insert> ) <InsertEnd>
<Keep> ;} <KeepEnd>
```

Unifying M_{old} and M_{new} into a single diff sequence that explicitly identifies code edits, M_{edit}

Representing Edits

M_{old}

```
public double getRotX() {
    return mOrientation.getRotationX();
}
```

M_{new}

```
public double getRotX() {
    return Math.toDegrees(mOrientation.getRotationX());
}
```

M_{edit}

```
<Keep> public double getRotX() <KeepEnd>
<Insert> Math.toDegrees( <InsertEnd>
<Keep> mOrientation.getRotationX() <KeepEnd>
<Insert> ) <InsertEnd>
<Keep> ;} <KeepEnd>
```

Unifying M_{old} and M_{new} into a single diff sequence that explicitly identifies <u>code</u> edits, M_{edit}

C_{old}

double the roll euler angle.

C_{new}

double the roll euler angle in degrees.

C_{edit}

```
<InsertOldKeepBefore> angle
<InsertNewKeepBefore> angle in degrees
<InsertEnd>
```

Unifying C_{old} and C_{new} into a single diff sequence that explicitly identifies <u>comment</u> edits, C_{edit}

Edit Model

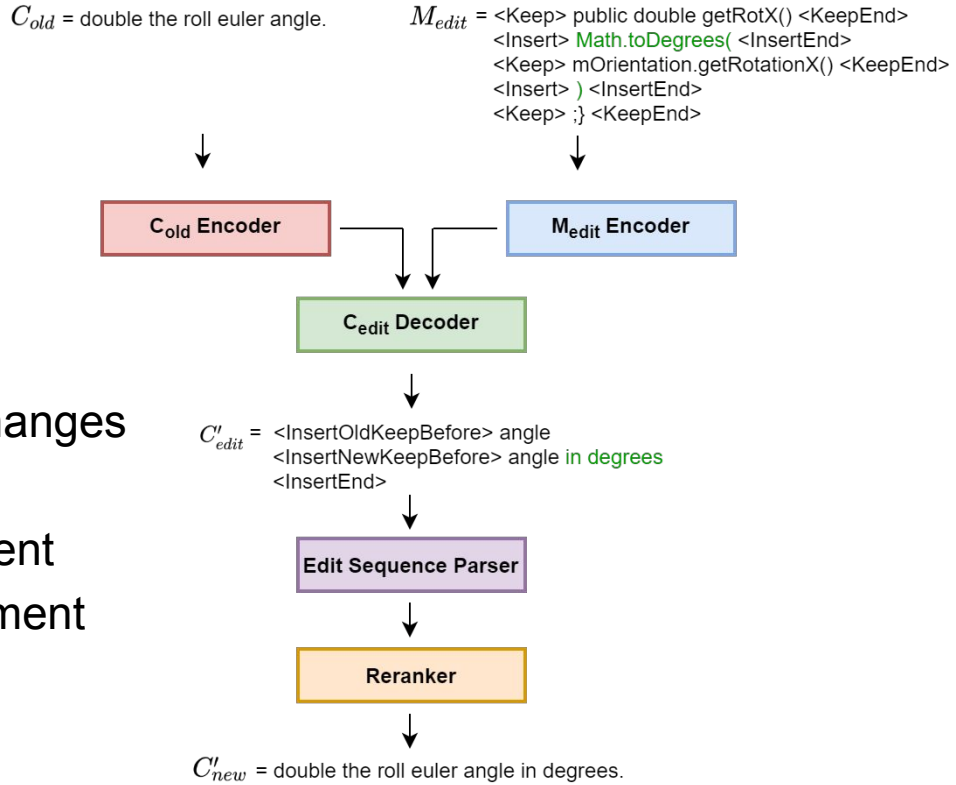
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

Step 3: Predict NL edits

Step 4: Apply NL edits to existing comment

Step 5: Rerank + produce updated comment



Edit Model

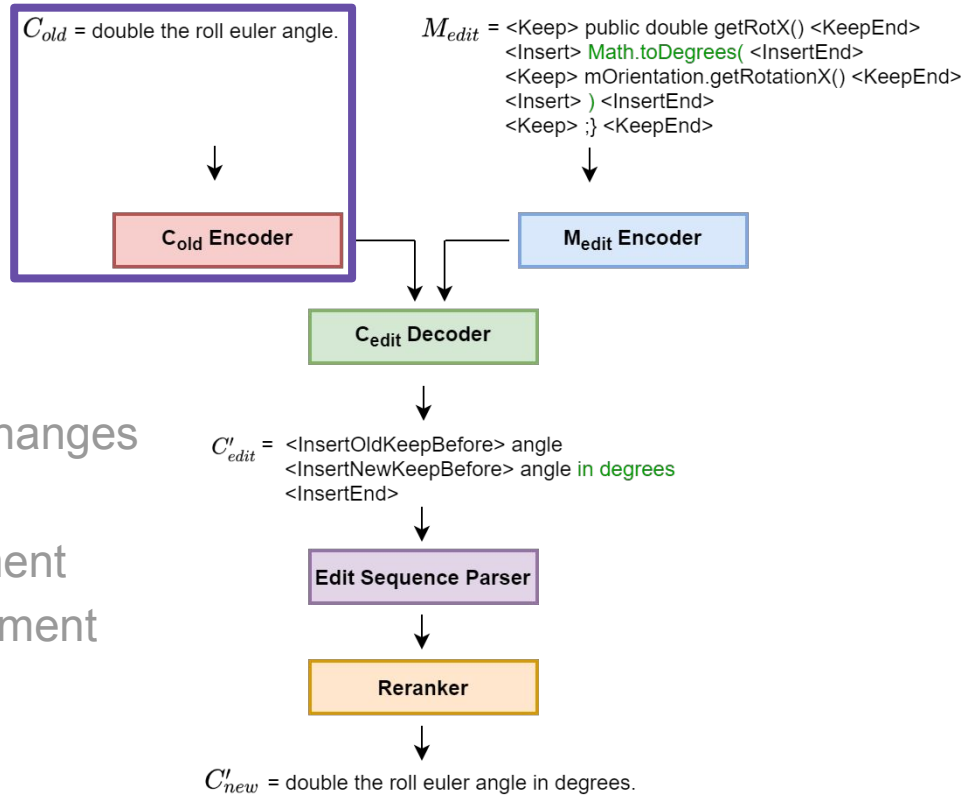
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

Step 3: Predict NL edits

Step 4: Apply NL edits to existing comment

Step 5: Rerank + produce updated comment



Edit Model

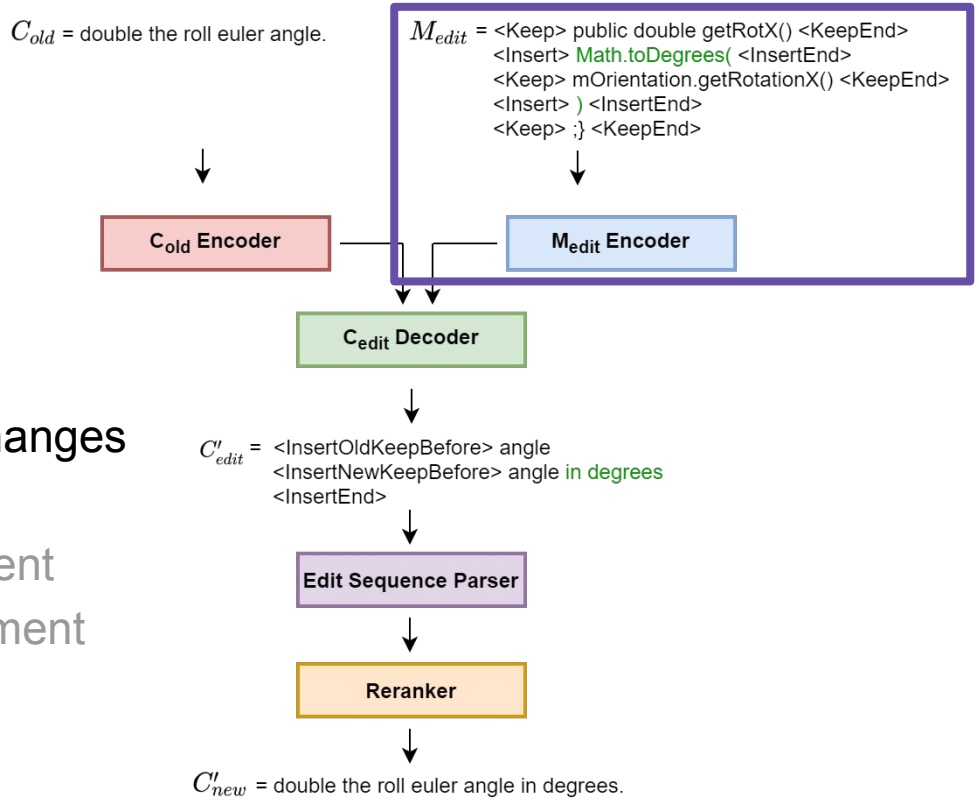
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

Step 3: Predict NL edits

Step 4: Apply NL edits to existing comment

Step 5: Rerank + produce updated comment



Edit Model

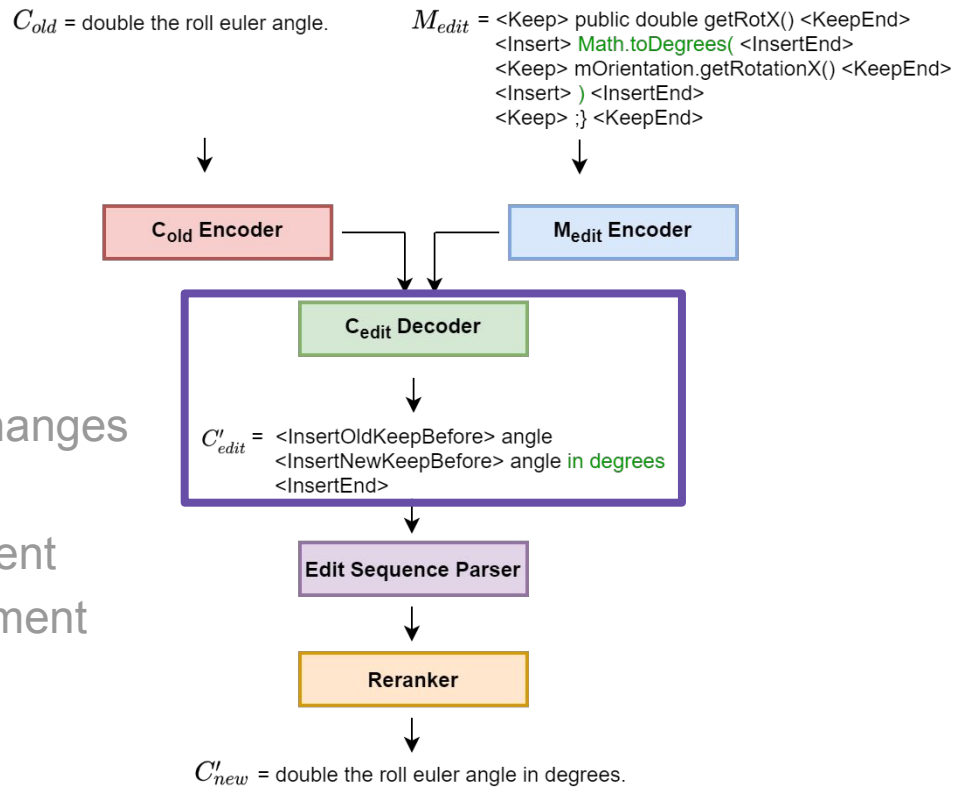
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

Step 3: Predict NL edits

Step 4: Apply NL edits to existing comment

Step 5: Rerank + produce updated comment



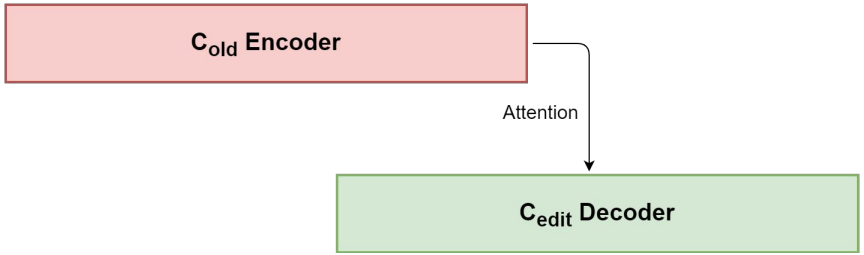
Step 3: Decoding C_{edit}

Generating C_{edit}, a sequence of NL edits, using a GRU decoder

At each decoding step:

(1) Identify edit locations in C_{old}

Step 3: Decoding C_{edit}



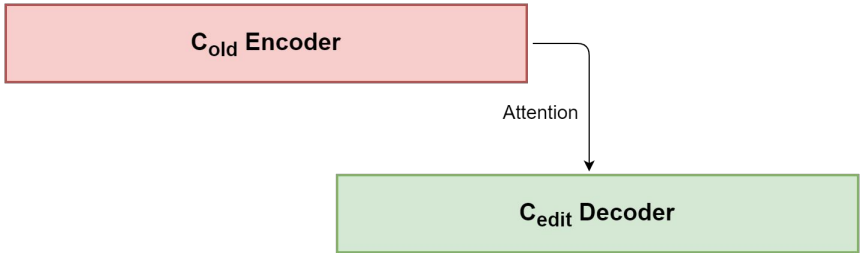
Generating C_{edit}, a sequence of NL edits, using a GRU decoder

At each decoding step:

(1) Identify edit locations in C_{old}

Attend to C_{old} encoder hidden states

Step 3: Decoding C_{edit}



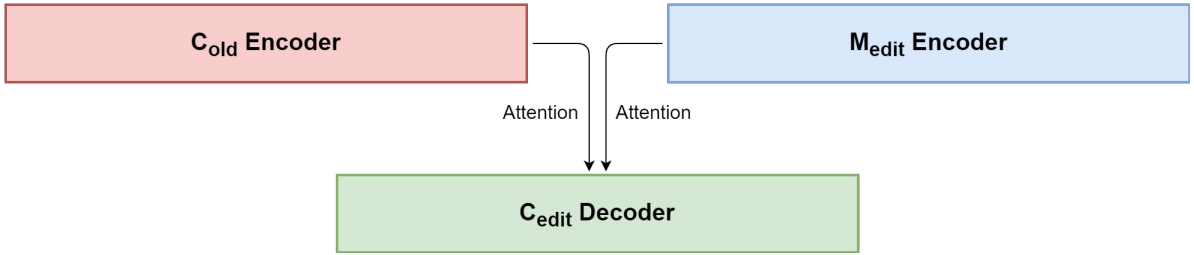
Generating C_{edit}, a sequence of NL edits, using a GRU decoder

At each decoding step:

(1) Identify edit locations in C_{old}
Attend to C_{old} encoder hidden states

(2) Determine parts of M_{edit} that pertain to making edits

Step 3: Decoding C_{edit}



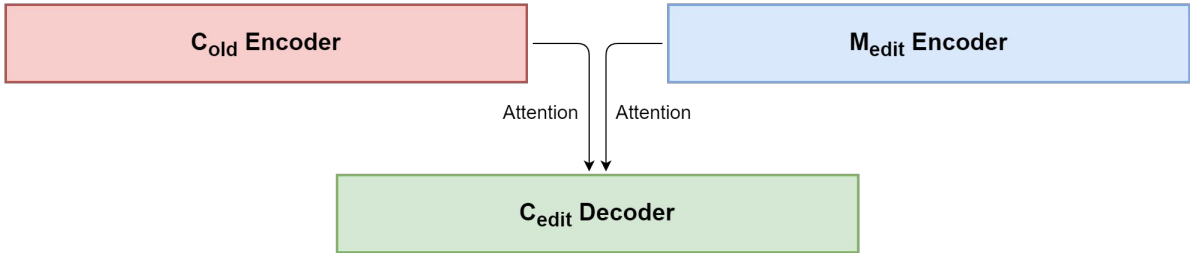
Generating C_{edit}, a sequence of NL edits, using a GRU decoder

At each decoding step:

(1) Identify edit locations in C_{old}
Attend to C_{old} encoder hidden states

(2) Determine parts of M_{edit} that pertain to making edits
Attend to M_{edit} encoder hidden states

Step 3: Decoding C_{edit}



Generating C_{edit}, a sequence of NL edits, using a GRU decoder

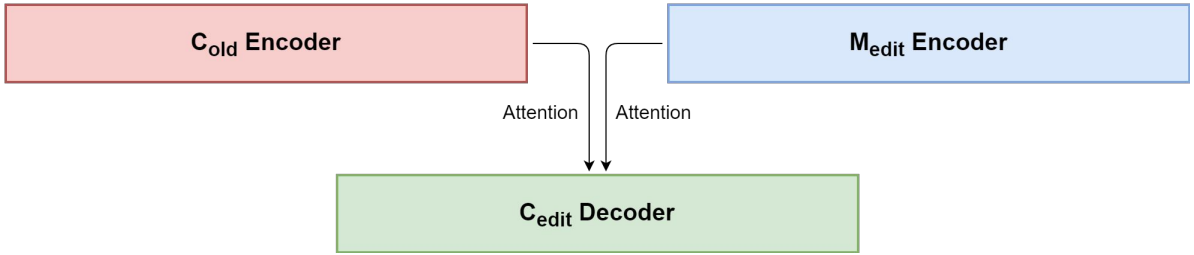
At each decoding step:

(1) Identify edit locations in C_{old}
Attend to C_{old} encoder hidden states

(2) Determine parts of M_{edit} that pertain to making edits
Attend to M_{edit} encoder hidden states

(3) Apply updates at edit locations based on the relevant code edits:

Step 3: Decoding C_{edit}



Generating C_{edit}, a sequence of NL edits, using a GRU decoder

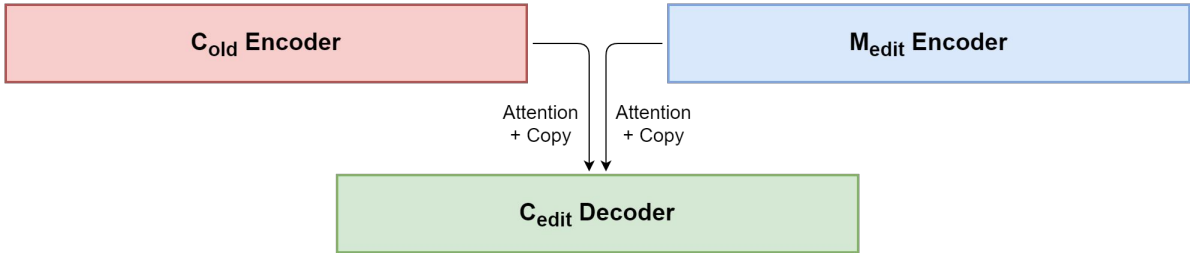
At each decoding step:

(1) Identify edit locations in C_{old}
Attend to C_{old} encoder hidden states

(2) Determine parts of M_{edit} that pertain to making edits
Attend to M_{edit} encoder hidden states

(3) Apply updates at edit locations based on the relevant code edits: start/end action or continue by generating/copying comment token

Step 3: Decoding C_{edit}



Generating C_{edit}, a sequence of NL edits, using a GRU decoder

At each decoding step:

(1) Identify edit locations in C_{old}
Attend to C_{old} encoder hidden states

(2) Determine parts of M_{edit} that pertain to making edits
Attend to M_{edit} encoder hidden states

(3) Apply updates at edit locations based on the relevant code edits: start/end action or continue by generating/copying comment token
Pointer network over C_{old} and M_{edit} encoder hidden states

Vinyals et al., 2015

Edit Model

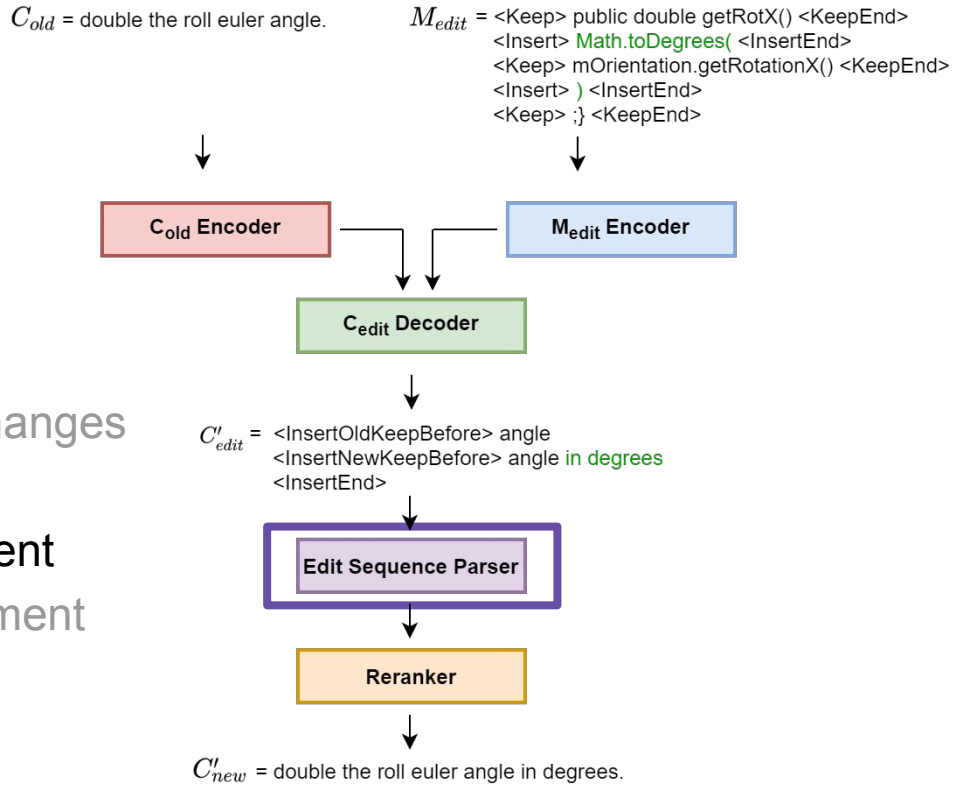
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

Step 3: Predict NL edits

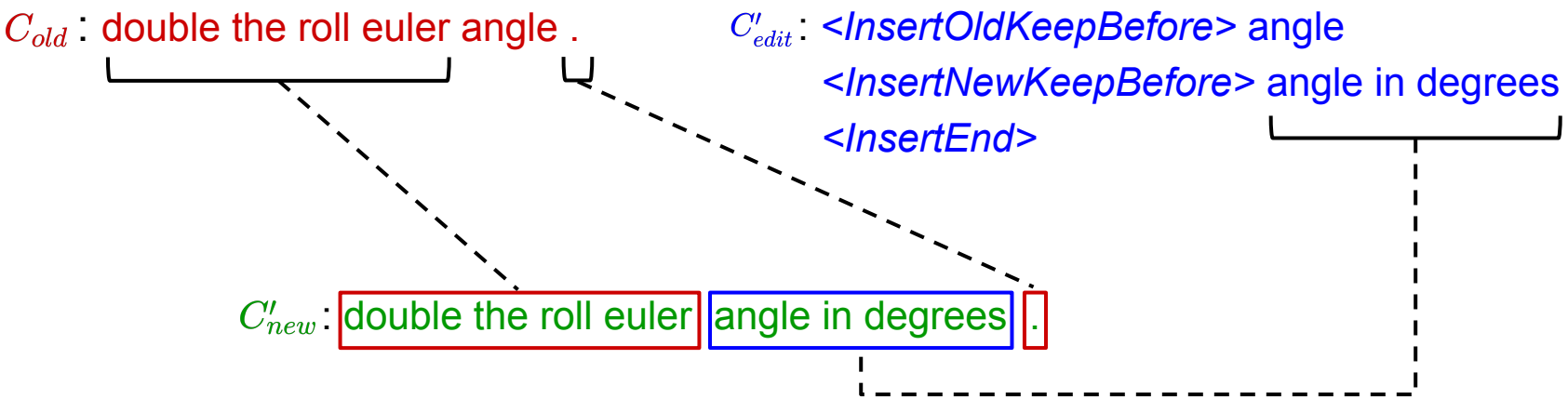
Step 4: Apply NL edits to existing comment

Step 5: Rerank + produce updated comment



Step 4: Parsing Edit Sequence

Aligning predicted edit sequence, C'_{edit}, with C_{old} and copying unchanged tokens to form predicted C'_{new}



Edit Model

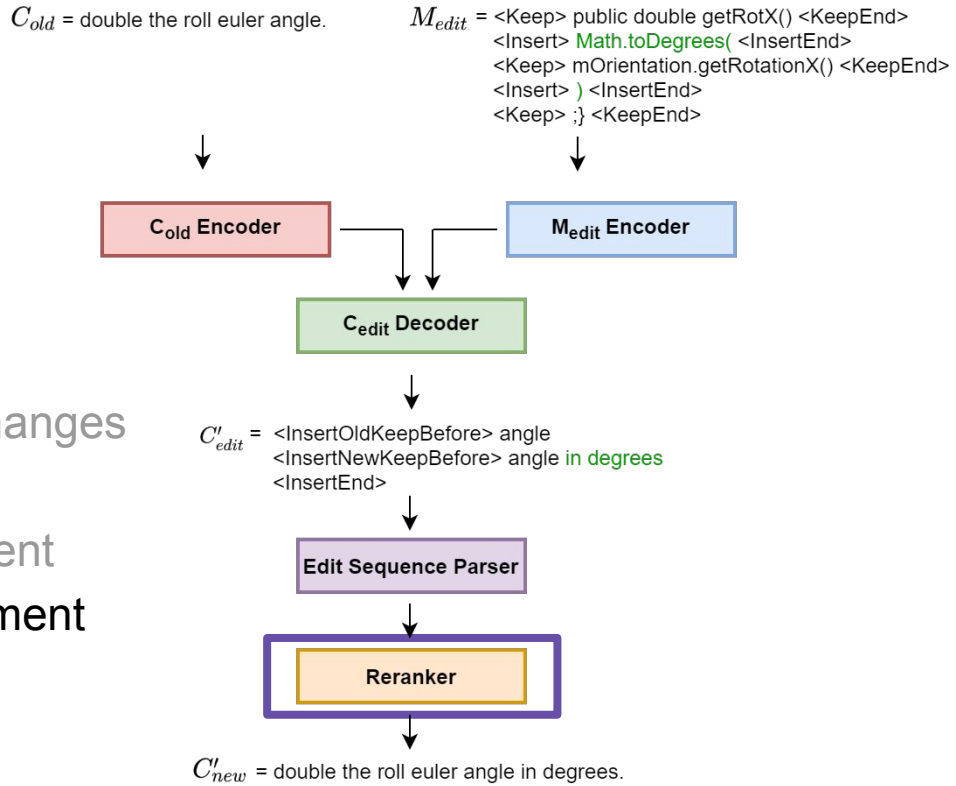
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

Step 3: Predict NL edits

Step 4: Apply NL edits to existing comment

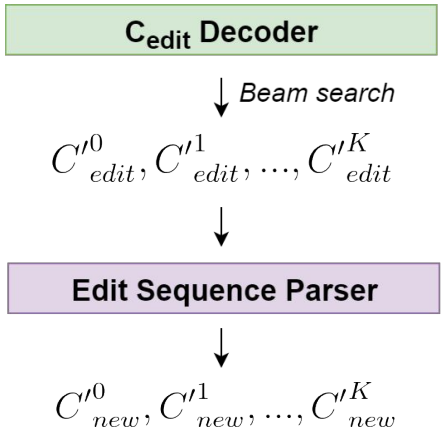
Step 5: Rerank + produce updated comment



Step 5: Selecting Best C'_{new} Candidate

Reranking candidate predictions

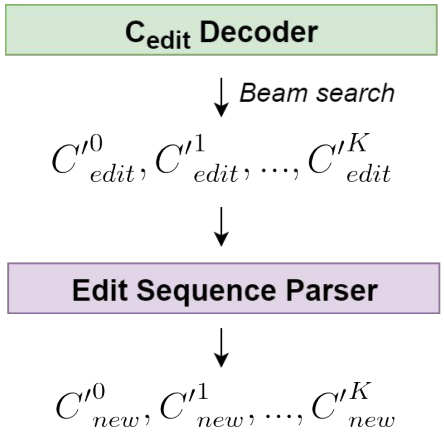
(1) Accurately update C_{old} with minimal modifications
(2) Be suitable for M_{new}
(3) Conform to comment style conventions



Step 5: Selecting Best C'_{new} Candidate

Reranking candidate predictions

(1) Accurately update C_{old} with minimal modifications
(2) Be suitable for M_{new}
(3) Conform to comment style conventions



Decoder trained to generate edits, and so has no notion of these global characteristics

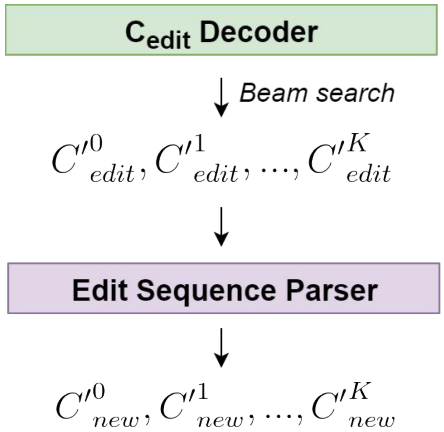
Step 5: Selecting Best C'_{new} Candidate

Reranking candidate predictions

(1) Accurately update C_{old} with minimal modifications
Similarity to C_{old}: $METEOR(C_{old}, C'^{i}_{new})$

(2) Be suitable for M_{new}

(3) Conform to comment style conventions



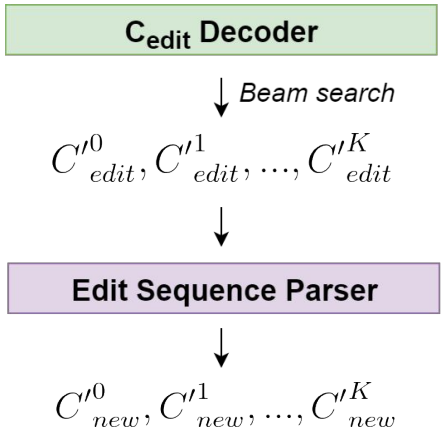
Step 5: Selecting Best C'_{new} Candidate

Reranking candidate predictions

(1) Accurately update C_{old} with minimal modifications
Similarity to C_{old}: $METEOR(C_{old}, C'^{i}_{new})$

(2) Be suitable for M_{new}

(3) Conform to comment style conventions
Generation likelihood: $P(C'^{i}_{new}|M_{new})$



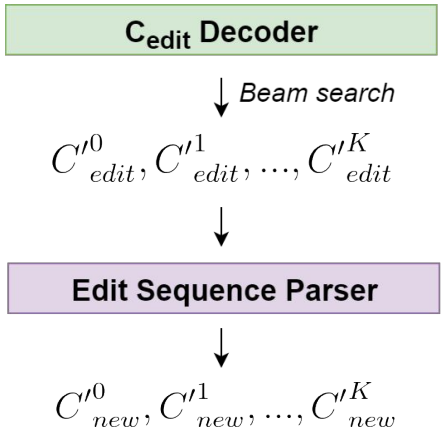
Step 5: Selecting Best C'_{new} Candidate

Reranking candidate predictions

(1) Accurately update C_{old} with minimal modifications
Similarity to C_{old}: $METEOR(C_{old}, C'^{i}_{new})$

(2) Be suitable for M_{new}

(3) Conform to comment style conventions
Generation likelihood: $P(C'^{i}_{new}|M_{new})$



$$Rerank(i) = 0.5Beam(i) + 0.2Sim(i) + 0.3GenLikelihood(i)$$

Edit Model

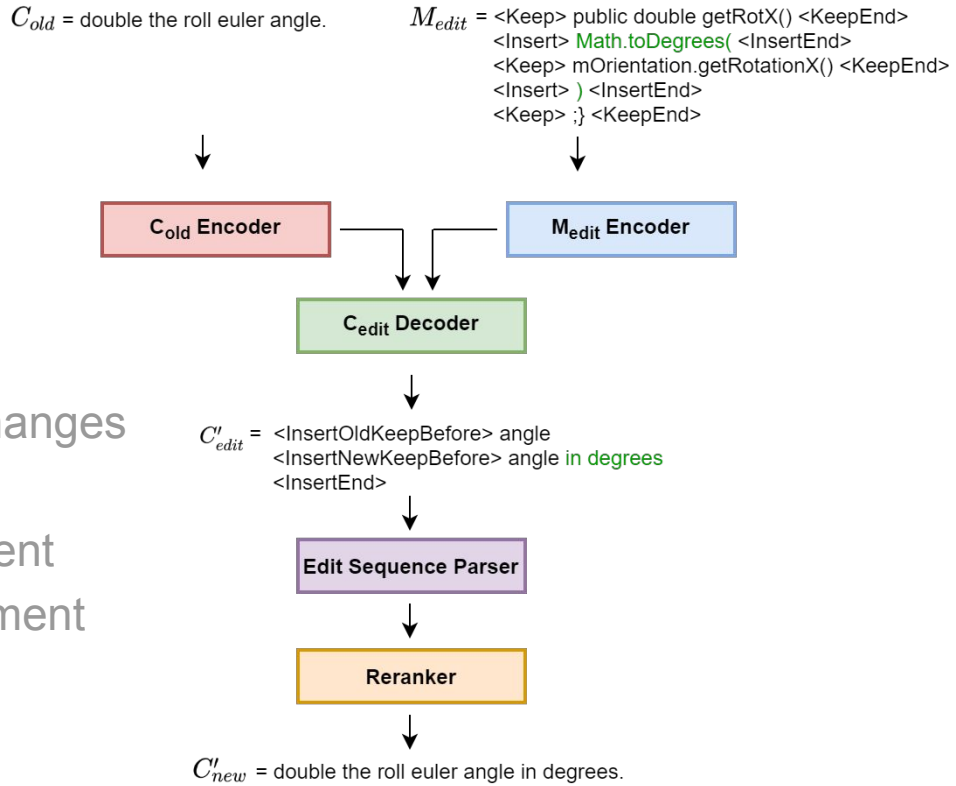
Step 1: Learn representation for C_{old}

Step 2: Learn representation for code changes

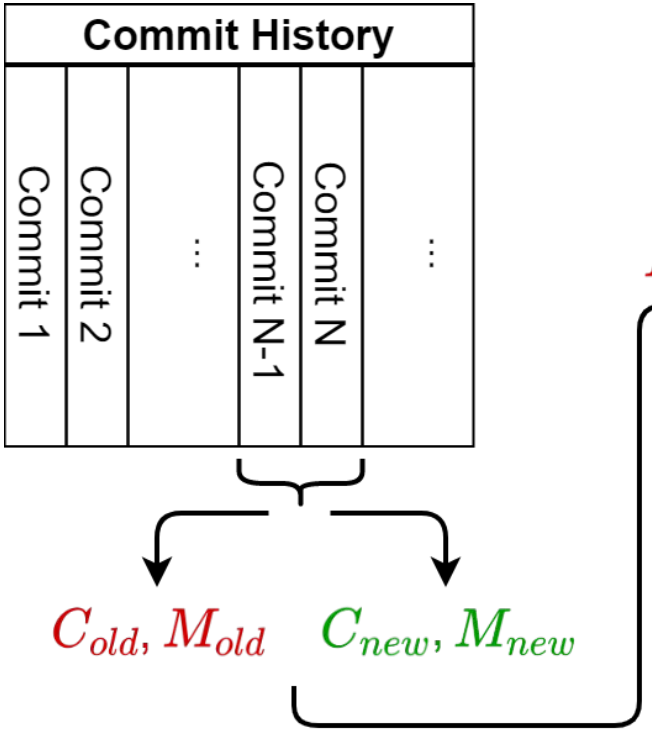
Step 3: Predict NL edits

Step 4: Apply NL edits to existing comment

Step 5: Rerank + produce updated comment



Data Collection



$C_{old} \neq C_{new}$
and
$M_{old} \neq M_{new}$

	Train	Validation	Test
Examples	5,791	712	736
Projects	526	274	281

Mining simultaneous updates to (comment, method) pairs from consecutive commits of open-source Java projects on GitHub

Baselines

- **Copy**

  $C'_{new} = C_{old}$

- **Generation w/ reranking**

  Given M_{new}, generate C'_{new} and rerank

- **Rule-based**

$$C'_{new} = \begin{cases} C_{old}\text{.replace}(\texttt{RetType}(M_{old}), \texttt{RetType}(M_{new})) + \text{“}\textit{or null if null}\text{”} & \text{if } \texttt{null} \text{ added to } \texttt{return} \text{ statement or } \texttt{if} \text{ statement in } M_{new} \\ C_{old}\text{.replace}(\texttt{RetType}(M_{old}), \texttt{RetType}(M_{new})) & \text{otherwise} \end{cases}$$

Automatic Evaluation Results

		Copy	Gen w/ reranking	Rule-based	Edit
	xMatch (%)	0.000	2.083	13.723	**18.433**
Generation	**METEOR** (Banerjee and Lavie, 2005)	34.611	18.170	43.359	**44.698**
Generation	**BLEU-4** (Papineni et al., 2002)	46.218	18.891	**51.160**	50.717
Editing	**SARI** (Xu et al., 2016)	19.282	25.641	32.109	**45.486**
Editing	**GLEU** (Napoles et al., 2015)	35.400	22.685	42.627	**46.118**

Automatic Evaluation Results

		Copy	Gen w/ reranking	Rule-based	Edit
	xMatch (%)	0.000	2.083	13.723	**18.433**
Generation	**METEOR** (Banerjee and Lavie, 2005)	34.611	18.170	43.359	**44.698**
Generation	**BLEU-4** (Papineni et al., 2002)	46.218	18.891	**51.160**	50.717
Editing	**SARI** (Xu et al., 2016)	19.282	25.641	32.109	**45.486**
Editing	**GLEU** (Napoles et al., 2015)	35.400	22.685	42.627	**46.118**

Despite being trained on more more data, the generation baseline substantially underperforms the edit model.

Automatic Evaluation Results

		Copy	Gen w/ reranking	Rule-based	Edit
	xMatch (%)	0.000	2.083	13.723	**18.433**
Generation	**METEOR** (Banerjee and Lavie, 2005)	34.611	18.170	43.359	**44.698**
Generation	**BLEU-4** (Papineni et al., 2002)	46.218	18.891	**51.160**	50.717
Editing	**SARI** (Xu et al., 2016)	19.282	25.641	32.109	**45.486**
Editing	**GLEU** (Napoles et al., 2015)	35.400	22.685	42.627	**46.118**

Rule-based baseline achieves a slightly higher BLEU-4 score; however the difference is NOT statistically significant.

Automatic Evaluation Results

		Copy	Gen w/ reranking	Rule-based	Edit
	xMatch (%)	0.000	2.083	13.723	**18.433**
Generation	**METEOR** (Banerjee and Lavie, 2005)	34.611	18.170	43.359	**44.698**
Generation	**BLEU-4** (Papineni et al., 2002)	46.218	18.891	**51.160**	50.717
Editing	**SARI** (Xu et al., 2016)	19.282	25.641	32.109	**45.486**
Editing	**GLEU** (Napoles et al., 2015)	35.400	22.685	42.627	**46.118**

Based on edit-specific metrics, our model appears to be better at editing comments.

Human Evaluation

- Given C_{old} and the diff of M_{old} and M_{new}:
  - Select the most suitable C'_{new} from **up to 3 suggestions**:
    - Generation model w/ reranking
    - Rule-based baseline
    - Edit model
  - Select **None** if all options are bad or if C_{old} does not need to be updated
- 10 participants w/ 2+ years Java experience
- Each participant annotated 50 examples
- Each example was annotated by 2 participants

500 evaluations across 250 distinct examples

Human Evaluation Results

Percentage of annotations for which users selected comment suggestions produced by each model

Gen w/ reranking	Rule-based	Edit	None
12.4%	18.4%	30.2%	55.0%

Inter-annotator agreement: 0.64 (Krippendorff's α with MASI distance)

Human Evaluation Results

Percentage of annotations for which users selected comment suggestions produced by each model

Gen w/ reranking	Rule-based	Edit	None
12.4%	18.4%	30.2%	55.0%

Inter-annotator agreement: 0.64 (Krippendorff's α with MASI distance)

The edit model outperforms the generation and rule-based baselines.

Human Evaluation Results

Percentage of annotations for which users selected comment suggestions produced by each model

Gen w/ reranking	Rule-based	Edit	None
12.4%	18.4%	30.2%	55.0%

Inter-annotator agreement: 0.64 (Krippendorff's α with MASI distance)

We found many cases in which the comment did not need to be updated.

Summary

- Formulated task of automatically updating comments based on code changes
- Introduced architecture for this task:
    (1) Generates a sequence of NL edits based on learned representations of the existing comment and code edits
    (2) Transforms this edit sequence into an updated comment by parsing and reranking based on global heuristics
- Evaluated approach against rule-based and generation baselines with automated metrics and a user study

Code and data available: https://github.com/panthap2/LearningToUpdateNLComments
Contact: Sheena Panthaplackel <spantha@cs.utexas.edu>