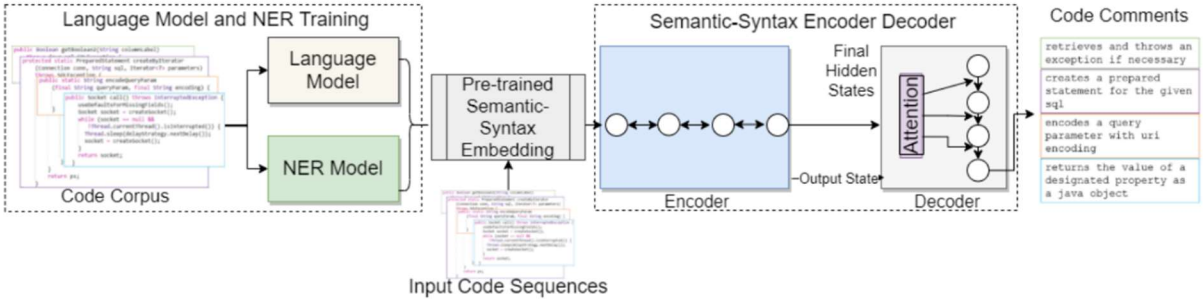
به نام خدا

صادق جعفری – امین متوسلی

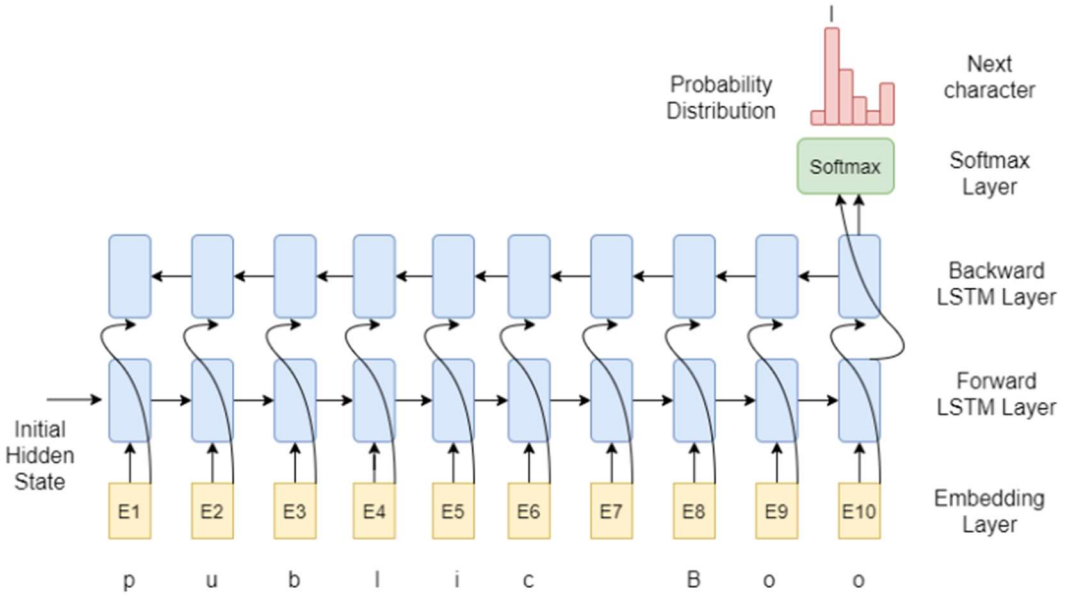
الگوریتم پیاده‌سازی:

به منظور انجام تسک ساخت کامنت در کد ها از شیوه‌ای جدید با بهره‌وری بیشتر استفاده خواهد شد. LAMNER شیوه‌ای شامل دو مدل زبانی و NER (Named Entity Recognition) می‌باشد. این الگوریتم در 3 مرحله کار میکند، نگاشت و آموزش ورودی بر اساس دو مدل یاد شده، نگاشت خروجی های دو مدل و ایجاد کامنت با استفاده از انکودر معنایی-نحوی (Semantic-Syntax). شکل 1.



شکل 1 مدل الگوریتم LAMNER

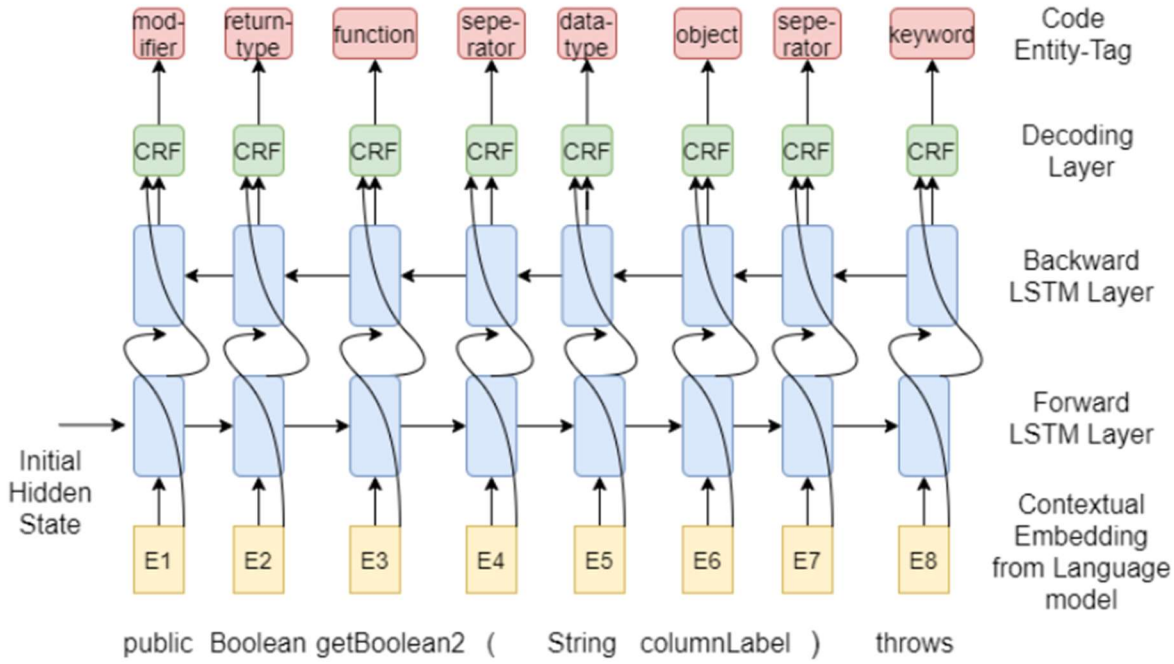
مدل زبانی در سطح کاراکتر فعالیت میکند. این مدل به منظور بررسی معنایی در کد می‌باشد تا به وسیله آن نگاشت های معنایی ایجاد گردند. ساختار این مدل زبانی با استفاده از یک لایه LSTM دو طرفه ایجاد شده است، با توجه به خروجی واحد قبلی در این ساختار، خروجی واحد بعدی بر اساس بیشترین احتمال تولید می‌گردد. در شکل 2 هر واحد محاسباتی در لایه پیش رونده، از واحد سمت چپ خود داده لایه پنهان خود را دریافت می‌کند و در لایه بازگشت هر واحد از واحد سمت راست خود این داده پذیرا خواهد بود. در لایه اول و دوم، به ترتیب آخرین و اولین واحد ها بر اساس واحد قبلی خود نگاشت هایی ایجاد می‌کنند که با الحاق این دو نگاشت، نگاشت خروجی این مدل تولید می‌گردد. همچنین هر واحد کاراکتر بعدی خود را در این مدل پیشبینی می‌کند.شکل 2.



شکل 2 الگوریتم LSTM در مدل زبانی

در تصویر بالا، کاراکتر l به ازای کاراکتر های وارد شده تا این لحظه یعنی public boo پیش‌بینی شده است.

در مدل NER تلاش بر تشخیص ساختار نحوی Token ها است. این مدل برای ایجاد نگاشت نحوی به یک نگاشت متنی از کد ورودی دارد که یک مدل زبانی می‌تواند آنرا تامین کند. با توجه به بهتر بودن نگاشت کاراکتری نسبت به نگاشت توکنی و عملکرد بهتر آن، از نگاشت هایی که در مدل زبانی بالا بدست آمده است برای این قسمت استفاده خواهیم کرد و با استفاده از یک LSTM دو طرفه توکن ها را تگ گذاری می‌کنیم.شکل 3.



شکل 3 الگوریتم مدل NER

در NER هدف استخراج نقش های نحوی توکن های وارد شده به مدل می‌باشد (نه پیش بینی). پس از آموزش مدل ویژگی های نحوی و نقش های توکن ها استخراج می‌گردد و نگاشت های نحوی بر اساس متن کد تشخیص داده می‌شود. دو نگاشت حاصل از مدل های بالا برای ایجاد کامنت مورد استفاده قرار خواهند گرفت.

در قسمت میانی شکل 1، دو نگاشت حاصل از این دو مدل با یکدیگر ترکیب می‌شوند (به صورتی که طول نگاشت ها دو برابر می‌گردد) و برای ایجاد کامنت مبتنی بر الگوریتم معنایی-نحوی به قسمت سوم در شکل 1 می‌رود.

در این قسمت انکودر نگاشت نهایی را دریافت می‌کند و بر اساس آنها مدل سازی معنایی-نحوی کرده (چراکه نگاشت معنایی-نحوی است) و دیکودر بر اساس داده های آموزشی از انکودر و کد ورودی کامنت مناسب را تولید خواهد نمود. انکودر ورودی را با استفاده از یک GRU دوطرفه یک لایه پردازش میکند. در این الگوریتم، به ازای هر توکن و نگاشت منتسب به آن، یک لایه پنهان با نام h_t ایجاد میکند. در نهایت لایه‌ی پنهان توکن نهایی h_{last}که اطلاعات تمام دنباله را تا این نقطه دارد از الحاق دو لایه پنهان h_{left} و h_{right} که توسط دولایه چپ و راست الگوریتم ساخته شده، ایجاد می‌گردد. سپس این لایه به یک لایه کاملا متصل خطی متصل شده که به صورت زیر است:

$$y_{fc} = h_{last} * W_t + b$$

خروجی این لایه توسط یک تابع tanh لایه نهفته‌ی نهایی را ایجاد می‌کند:

$$h_{final} = \tanh(y_{fc})$$

دیکودر نیز از مکانیزم Bahdanau استفاده میکند که در این مکانیزم بر اساس h_{final} حاصل وزن های توجه (attention) برای هر توکن تخصیص داده و به توکن با وزن بیشتر توجه بیشتری خواهد داشت. بر اساس نگاشت ها معنایی توکن های بعدی پیش بینی شده و کامنت ها ساخته می‌شوند، به صورتی که به ازای هر توکن پیش بینی نمی‌کند بلکه بر اساس وزن های توجه محاسبه شده و کلمات با اهمیت بالاتر سعی در ایجاد کامنت دارد.

منبع جمع آوری داده:

- منابع جمع آوری داده در این پروژه کدهای جاوا موجود در Github هستند, که می‌توان فایل zip هر یک را دانلود کرد و بعد از خارج کردن از حالت فشرده path آن را به برنامه بدهیم تا برنامه دیتا مورد نیاز را از آن استخراج کند.

روش جمع آوری داده:

- ابزار استفاده شده python + antlr4 است.
- ابتدا با استفاده از Parse-tree antlr هر فایل جاوا را به دست می‌آوریم و بعد از پیدا کردن محل methodها(با استفاده از listener) و کامنت‌های آن‌ها(با استفاده از lexer) آن‌ها را در یک فایل json ذخیره می‌کنیم.

فرمت داده‌ها:

- ساختار کلی داده‌ها به صورت یک فایل json است که هر سطر موجود در آن شامل یک تابع و کامنت آن تابع است.


```
{
    "method_text": "boolean match(ACLMessage msg) {\n\t\t\ttry {\n\t\t\t\tContentElement ce = ConsumerAgent.
    "method_tokens": "['boolean', 'match', '(', 'ACLMessage', 'msg', ')', '{', 'try', '{', 'ContentElement',
    "comment_text": "This method verifies the action of the Request ACLMessage.",
    "comment_tokens": "['This', 'method', 'verifies', 'the', 'action', 'of', 'the', 'Request', 'ACLMessage',
},
```


- همانطور که در شکل بالا می‌بینید method_text متن خام یک method است, method_tokens توکن‌ها آن method است که با استفاده از ابزار antlr و lexer آن توکن‌ها تولید شده است, comment_text متن خام کامنت هر method است و commet_tokens هم توکن‌های متن خام کامنت است که با استفاده از nltk tokenizer به دست آمده است.
- یک نکته قابل توجه در این پیش پردازش داده این است که از تابع get_cumulative_comments برای تجمیع کامنت‌های نزدیک به هم استفاده شده است که کاربرد آن را به صورت دقیق در مثال زیر توضیح می‌دهم:

```
// this is comment
/* this
is comment 2
*/
// this is comment 3
public String toString() {
    return image + ": is decoded";
}
```

- در کد بالا همانطور که می‌بینید 3 تا کامنت وجود دارد که یکی چند خطی و 2 مورد هم کامنت ساده است, در پیش پردازش توسط تابع get_cumulative_comments این سه کامنت به یک دیگر می‌چسبند و یک کامنت را تشکیل

می‌دهند و کامنت حاصل به method پایینش اختصاص داده می‌شود, علت این کار این است که بعضی از برنامه نویس‌ها توضیحات مربوط به یک قسمت را در چند کامنت بیان می‌کنند.

واحد برچسب گذاری:

- با توجه به این که موضوع این پروژه تولید کامنت است, ما در اینجا چیزی به نام برچسب گذاری نداریم و اینجا مسئله بیشتر شبیه به مسئله ترجمه ماشینی است که کد را به زبان انگلیسی ترجمه می‌کند, پس ما در این جا کد و ترجمه آن کد به زبان انگلیسی که همان کامنت است را داریم.

آمار داده:

- چون ممکن است برای فاز بعدی نیاز به داده بیشتری باشد این آماری که در زیر می‌بینید صرفا برای داده‌هایی است که در فاز یک در کوئرا آپلود کردیم:

تعداد کل داده‌ها	تعداد جملات در کامنت‌ها	تعداد توکن‌های کامنت	تعداد توکن‌های کد	تعداد نوع کلمات کامنت	تعداد نوع کلمات کد
68	92	1182	2801	271	331

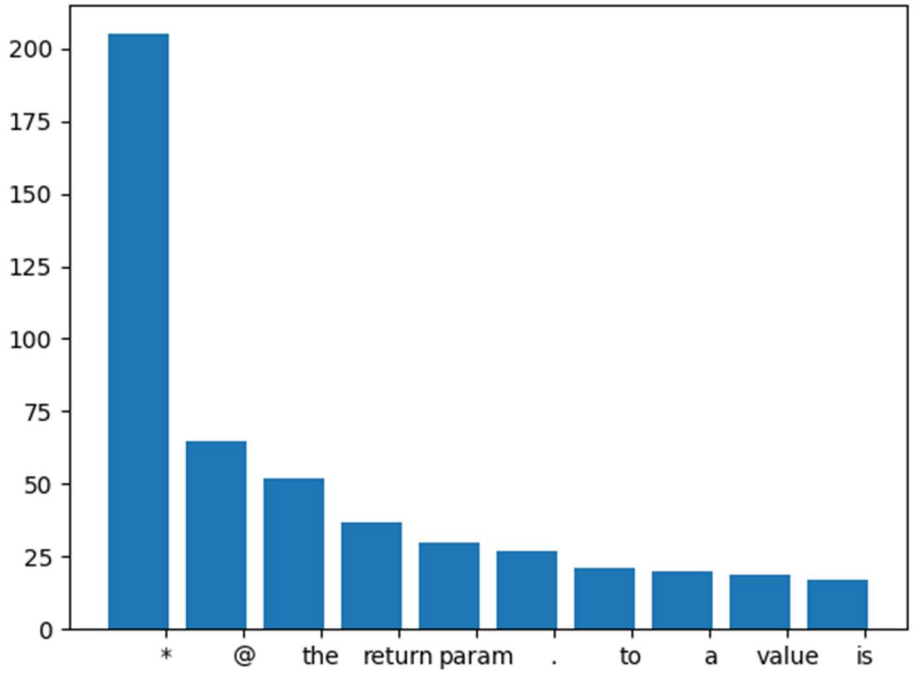


Figure1 : comment histogram

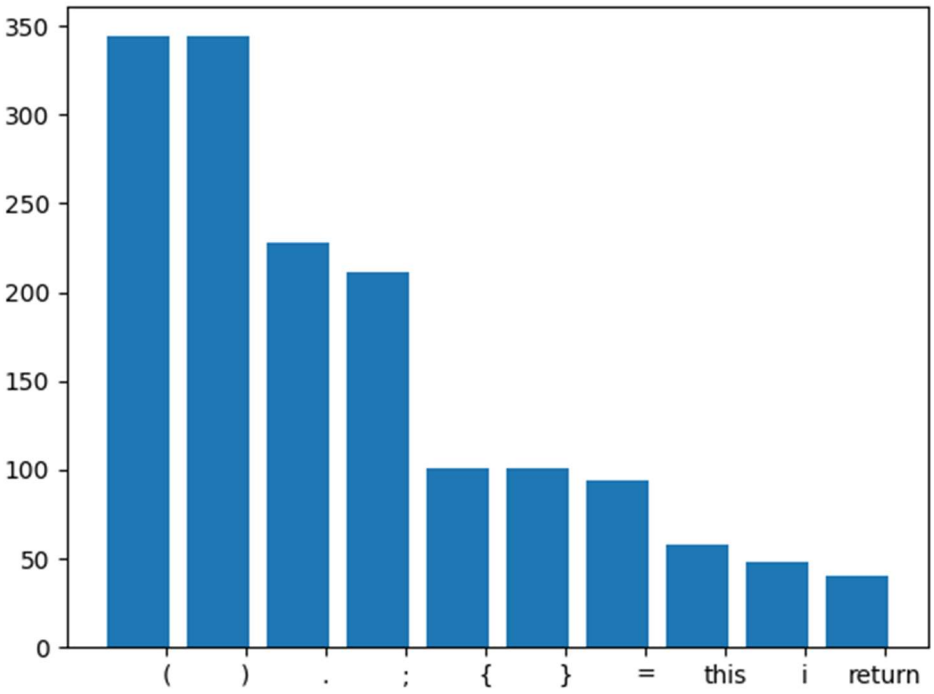


Figure2 : code histogram

منابع استفاده شده:

https://arxiv.org/abs/2204.09654