



دانشکده‌ی مهندسی کامپیوتر

تشخیص بوی کد با استفاده از مدل های زبانی وسیع پیشنهادی

پروژه‌ی پایانی کارشناسی مهندسی کامپیوتر

محمدصادق پولائی موزیرجی

سایین اعلا

استاد راهنما

دکتر سعید پارسا

شهریور ۱۴۰۳



تأییدی هیأت داوران جلسه دفاع از پروژه

نام دانشکده: دانشکده مهندسی کامپیوتر

نام دانشجویان: محمدصادق پولائی موزیرجی، سایین اعلا

عنوان پروژه: تشخیص بوی کد با استفاده از مدل های زبانی وسیع پیشنهادی

تاریخ دفاع: شهریور ۱۴۰۳

رشته: مهندسی کامپیوتر

ردیف	سمت	نام و نام خانوادگی	مرتبه و نام دانشگاهی	دانشگاه یا مؤسسه	امضاء
۱	استاد راهنما	دکتر سعید پارسا	استاد	دانشگاه علم و صنعت ایران	
۲	استاد داور داخلی	دکتر	دانشگاه علم و صنعت ایران	

تأییدی صحت و اصالت نتایج

باسمه تعالی

اینجانبان محمدصادق پولائی موزیرجی به شماره دانشجویی ۹۹۵۲۱۱۴۵ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی و سایین اعلا به شماره دانشجویی ۹۹۴۰۰۰۲۳ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی تأیید می‌نماییم که کلیه نتایج این پروژه حاصل کار اینجانبان و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ایم. در صورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی) با اینجانبان رفتار خواهد شد و حق هرگونه اعتراض در خصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نماییم. در ضمن، مسئولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذیصلاح (اعم از اداری و قضایی) به عهده‌ی اینجانبان خواهد بود و دانشگاه هیچ‌گونه مسئولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: محمدصادق پولائی موزیرجی

تاریخ و امضا:

نام و نام خانوادگی: سایین اعلا

تاریخ و امضا:

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

- ☐ بهره‌برداری از این پایان‌نامه برای همگان بلامانع است.
- ☐ بهره‌برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.
- ☐ بهره‌برداری از این پایان‌نامه تا تاریخ ممنوع است.

استاد راهنما: دکتر سعید پارسا

تاریخ:

امضا:

قدردانی

سپاس خداوندگار حکیم را که با لطف بی‌کران خود، آدمی را زیور عقل آراست.
در آغاز وظیفه خود می‌دانم از زحمات بی‌دریغ استاد راهنمای خود، جناب آقای دکتر سعید پارسا، صمیمانه تشکر و قدردانی کنم که قطعاً بدون راهنمایی‌های ارزنده ایشان، این مجموعه به انجام نمی‌رسید.

از جناب آقای دکتر سعید پارسا که زحمت مطالعه و مشاوره این رساله را تقبل فرمودند و در آماده سازی این رساله، به نحو احسن اینجانب را مورد راهنمایی قرار دادند، کمال امتنان را دارم.
در پایان، بوسه می‌زنم بر دستان خداوندگاران مهر و مهربانی، پدر و مادر عزیزم و بعد از خدا، ستایش می‌کنم وجود مقدس‌شان را و تشکر می‌کنم از خانواده عزیزم به پاس عاطفه سرشار و گرمای امیدبخش وجودشان، که بهترین پشتیبان من بودند.

محمدصادق پولائی موزیرجی

سایین اعلا

شهریور ۱۴۰۳

چکیده

در این پژوهش، به بررسی و پیشنهاد مدلی برای تشخیص بوی کد با استفاده از مدل‌های زبانی وسیع پرداخته شده است. بوی کد به مفاهیم و ویژگی‌هایی در کد برنامه‌نویسی اشاره دارد که می‌تواند نشان‌دهنده مشکلات عمیق‌تری در طراحی و پیاده‌سازی نرم‌افزار باشد. این مشکلات ممکن است به کاهش کیفیت کد و افزایش پیچیدگی در نگهداری و توسعه منجر شوند. روش پیشنهادی این پژوهش شامل استفاده از مدل‌های زبانی وسیع است که با استفاده از مجموعه داده‌های برچسب‌خورده آموزش دیده و توانایی تشخیص ۲۸ نوع مختلف از بوی کد را دارد. مدل پیشنهادی با بهره‌گیری از روش‌های یادگیری عمیق و معماری‌های پیشرفته نظیر مبدل‌ها توسعه یافته است. نتایج ارزیابی‌ها نشان می‌دهد که مدل پیشنهادی قادر است بهبود قابل توجهی در دقت تشخیص بوی کد ایجاد کند و می‌تواند به عنوان ابزاری مؤثر برای توسعه‌دهندگان نرم‌افزار مورد استفاده قرار گیرد. همچنین در این پژوهش به چالش‌های موجود در آموزش و بهینه‌سازی مدل‌های زبانی وسیع پرداخته شده و راهکارهایی برای ارتقای عملکرد مدل ارائه شده است.

واژگان کلیدی: بوی کد، مدل‌های زبانی وسیع، یادگیری عمیق، تشخیص خودکار، مبدل، بهینه‌سازی مدل

فهرست مطالب

ح	فهرست تصاویر
خ	فهرست جداول
د	فصل ۱: مقدمه
۲	۱-۱ لزوم تشخیص بوی کد در صنعت نرم افزار
۵	فصل ۲: مفاهیم و کار های انجام شده
۶	۱-۲ مفاهیم اولیه
۱۶	۲-۲ کار های انجام شده با روش های پیشین
۱۹	فصل ۳: شرح مسئله
۲۳	فصل ۴: روش پیشنهادی
۲۴	۱-۴ پردازش و جمع آوری داده
۲۴	۲-۴ تعریف مدل
۲۸	۳-۴ آموزش مدل
۳۱	فصل ۵: ارزیابی و معیارهای سنجش عملکرد
۳۳	۱-۵ معیارهای ارزیابی
۳۵	۲-۵ نحوه ارزیابی مدل
۳۶	۳-۵ نتایج ارزیابی

فصل ۶: نتیجه گیری و کارهای آینده ۳۸

۱-۶ نتیجه گیری ۳۹

۲-۶ کارهای آینده ۴۰

واژه نامه فارسی به انگلیسی ۴۳

کتاب نامه ۵۳

فهرست تصاویر

۱-۲	مثالی از درخت نحو انتزاعی	۷
۲-۲	معماری شبکه عصبی بازگشتی	۹
۳-۲	معماری شبکه عصبی حافظه طولانی کوتاه مدت	۱۰
۴-۲	معماری شبکه عصبی (Attention)	۱۱
۵-۲	معماری (Transformer)	۱۳
۶-۲	معماری (LLaMA)	۱۴
۱-۴	نحوه بهبود دادن مدل	۲۷
۱-۵	نمونه خروجی مدل و برجسب های داده	۳۵

فهرست جداول

- ۱-۳ لیست بوی بد کد و توضیحات آن‌ها ۲۱
- ۱-۵ نتایج ارزیابی مدل‌های مختلف با روش شبکه عصبی چندلایه ۳۶
- ۲-۵ نتایج ارزیابی مدل‌های مختلف با روش مهندسی درخواست ۳۷

فصل اول

مقدمه

۱. مقدمه

تشخیص بوی کد^۱ یکی از مفاهیم اساسی در صنعت نرم‌افزار است که اهمیت بسیاری دارد. هر پروژه نرم‌افزاری از یک مجموعه بزرگ از کدها تشکیل شده است که ممکن است توسط چندین توسعه‌دهنده نوشته شده باشند. به همین دلیل، اهمیت تشخیص بوی کد به‌ویژه در پروژه‌های بزرگ بسیار چشمگیر است.

بوی کد به مفهومی گفته می‌شود که نشان‌دهنده کیفیت کد و ساختار آن است. یک کد خوب، کدی است که درک آن ساده است، قابل توسعه بوده و از جوانب مختلفی نظیر بهینه بودن، قابلیت خواندن، تست‌پذیری و قابلیت تغییر برخوردار است.

بوی کد نه تنها برای توسعه‌دهندگان فعلی بلکه برای توسعه‌دهندگان جدیدی که احتمالاً بعداً به پروژه می‌پیوندند نیز حائز اهمیت است. با تشخیص بوی کد، توسعه‌دهندگان به راحتی می‌توانند رفتار و عملکرد کد را درک کرده، مشکلات را پیدا کرده و بهبودهای لازم را اعمال کنند.

در مجموع، تشخیص بوی کد به توسعه‌دهندگان کمک می‌کند تا کدهای بهتری بنویسند، پروژه‌های بهتری ارائه دهند و در نهایت، هزینه‌های توسعه و نگهداری را کاهش دهند.[۱]

^۱code smell

۱-۱ دلایل لزوم تشخیص بوی کد

تشخیص و رفع بوی کد باید بخشی از فرآیند توسعه نرم افزار باشد. این کار کمک می کند تا نرم افزارهایی پایدارتر، قابل نگهداری تر و با کیفیت تر ایجاد شوند و در بلندمدت هزینه ها و زمان مورد نیاز برای توسعه و نگهداری را کاهش دهد.

۱-۱-۱ افزایش قابلیت نگهداری

کدهای نرم افزاری به مرور زمان پیچیده تر می شوند، به ویژه هنگامی که تغییرات یا ویژگی های جدید به آن اضافه می شوند. اگر بوی کد در مراحل اولیه تشخیص داده نشود، این پیچیدگی ها ممکن است به چنان حدی برسند که نگهداری و توسعه کد بسیار دشوار شود. تشخیص و رفع بوی کد به بهبود ساختار و معماری کد کمک می کند، از انباشت پیچیدگی های غیرضروری جلوگیری می کند و باعث می شود نگهداری و توسعه کد در آینده آسان تر شود. این مسئله به خصوص برای تیم هایی که روی پروژه های بزرگ کار می کنند یا توسعه دهندگانی که به کدهای قدیمی برمی گردند، اهمیت بیشتری پیدا می کند.

۲-۱-۱ بهبود خوانایی و درک کد

یکی از عوامل کلیدی برای کدی که بتوان آن را به خوبی نگهداری کرد، خوانایی و قابلیت درک آن است. بوی کد اغلب ناشی از کدهای پیچیده، متدهای بزرگ، یا ساختارهای نامنظم است که خواندن و درک آن ها دشوار می شود. این مشکل زمانی بحرانی تر می شود که توسعه دهندگان جدید به تیم بپیوندند یا اعضای تیم فعلی پس از مدتی به کدی برگردند که دیگر به یاد نمی آورند. با تشخیص بوی کد و بهبود طراحی، کد قابل فهم تر و منظم تر می شود. این به تیم کمک می کند تا سریع تر با کد کار کند و کمتر دچار اشتباهات ناشی از سوء تفاهم در ساختار کد شوند.

۳-۱-۱ کاهش خطر خطاهای آینده

بوی کد معمولاً نشانه‌ای از مشکلات پنهانی است که ممکن است هنوز به صورت خطا آشکار نشده باشند، اما توانایی آن را دارند که در آینده مشکل‌ساز شوند. برای مثال، کدهای تکراری ممکن است منجر به مشکلات سازگاری شوند، یا کلاس‌ها^۲ و متدهای^۳ پیچیده ممکن است منجر به بروز خطاهایی شوند که پیدا کردن و رفع آن‌ها سخت است. تشخیص بوی کد به شناسایی این مشکلات احتمالی کمک می‌کند و رفع آن‌ها باعث می‌شود که در آینده خطاهای کمتری در کد مشاهده شود. به عبارت دیگر، پیشگیری از مشکلات به جای رفع آن‌ها پس از بروز، از هزینه و زمان زیادی صرفه‌جویی می‌کند.

۴-۱-۱ افزایش بهره‌وری تیم توسعه

بوی کد می‌تواند باعث شود که اعضای تیم توسعه زمان بیشتری را برای درک و رفع مشکلات موجود در کد صرف کنند، به خصوص زمانی که این مشکلات به مرور زمان پیچیده‌تر و غیرقابل فهم‌تر می‌شوند. با تشخیص و رفع بوی کد، تیم می‌تواند روی توسعه ویژگی‌های جدید تمرکز کند به جای اینکه وقت زیادی را صرف رفع خطاهای ناشی از ساختار بد کد کند. در نتیجه، تیم توسعه می‌تواند با سرعت بیشتری پیشرفت کند و بهره‌وری آن به طور کلی افزایش یابد. علاوه بر این، کدهای تمیز و بهینه تعامل بهتری بین اعضای تیم ایجاد می‌کنند، زیرا ارتباطات و درک بهتر در مورد ساختار و مسئولیت‌های مختلف کد وجود دارد.

۵-۱-۱ پشتیبانی از تغییرات و ارتقاء سیستم

نرم‌افزارها معمولاً در طول زمان نیاز به تغییر، به‌روزرسانی یا ارتقاء دارند تا با نیازهای جدید کاربران، بازار، یا فناوری سازگار شوند. بوی کد می‌تواند این فرایند را دشوارتر و پرخطرتر کند. به عنوان مثال، وابستگی‌های سنگین بین کلاس‌ها و ماژول‌ها^۴ ممکن است باعث شود که تغییر در یک بخش از کد، منجر به مشکلات غیرمنتظره در بخش‌های دیگر شود. اما با تشخیص و رفع بوی کد، ساختار کد

^۲class

^۳method

^۴module

بهبود می‌یابد و مازول‌ها به صورت مستقل‌تر عمل می‌کنند. این امر به توسعه‌دهندگان اجازه می‌دهد که تغییرات و ارتقاءهای لازم را بدون ایجاد مشکلات ناخواسته اعمال کنند.

ادامه‌ی پروژه به پنج فصل تقسیم شده است. در فصل ۲ پژوهش‌های مرتبط انجام شده را مرور می‌کنیم. در فصل ۳ به شرح مسئله‌ی توانایی شناسایی و دسته‌بندی بوی کدها می‌پردازیم. در فصل ۴ روش پیشنهادی برای توانایی شناسایی و دسته‌بندی بوی کدها را شرح می‌دهیم. در فصل ۵ نحوه‌ی ارزیابی مدل را ارائه می‌دهیم. در انتها در فصل ۶ یک جمع‌بندی کلی از تمامی مطالب ارائه می‌دهیم و پیشنهاداتی برای گسترش روش پیشنهادی عنوان می‌کنیم.

فصل دوم

مفاهیم و کارهای انجام شده

۲. مفاهیم و کارهای انجام شده

سیستم‌های قانون‌محور^۱ یکی از رویکردهای اصلی برای تشخیص بوی کد در صنعت نرم‌افزار هستند. این سیستم‌ها بر اساس تعدادی از قوانین یا قواعد برنامه‌نویسی شده توسط توسعه‌دهندگان عمل می‌کنند. این قواعد معمولاً بر اساس اصول نگارش کد، الگوهای طراحی خوب و استانداردهای برنامه‌نویسی تعریف شده‌اند.

۱-۲ مفاهیم اولیه

Code Smell ۱-۱-۲

بوی کد در برنامه‌نویسی کامپیوتر، به ویژگی‌ها یا نشانه‌هایی در کد یک برنامه که حاکی از وجود مشکلاتی در عمق برنامه باشند، بوی کد گفته می‌شود. تعیین اینکه چه چیزی کد بو محسوب می‌شود یا نه، وابسته به فرد توسعه‌دهنده، زبان برنامه‌نویسی و مدت توسعه می‌باشد. یک شیوه برای نگاه به کد بو در نظر گرفتن اصول و کیفیت طراحی است و بوهای کد ساختارهایی مشخص در کد هستند که اصول پایه‌ای برنامه‌نویسی را به گونه‌ای نقض کرده‌اند و کیفیت طراحی را پایین آورده‌اند.

در واقع بوهای بد کد خطا محسوب نمی‌شوند به این معنا که مانع از کارکرد صحیح برنامه نمی‌شوند. آنها ضعف‌هایی در طراحی را نمایان می‌کنند که باعث کند شدن روند توسعه هستند یا ریسک ایجاد خطاها یا خرابی در آینده را افزایش می‌دهند.

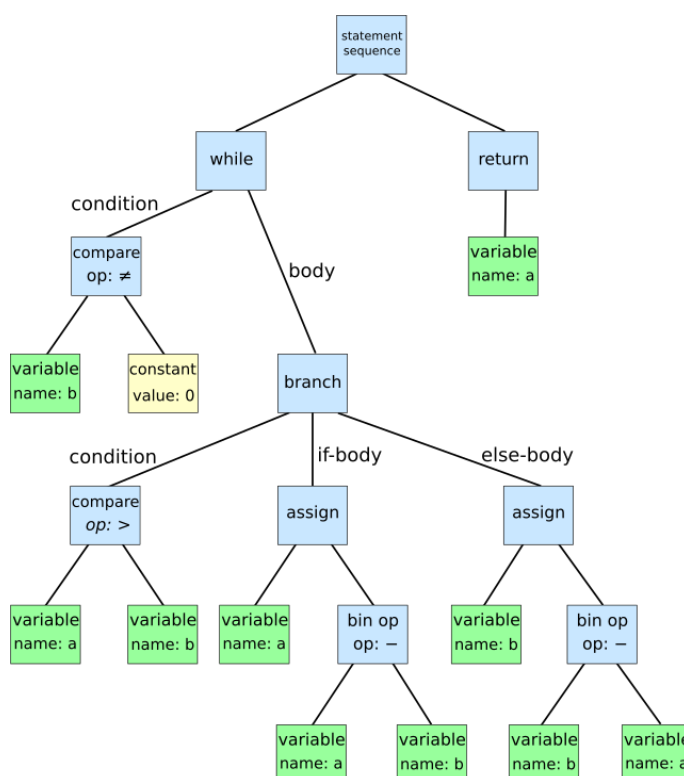
معمولاً مشکل عمیق‌تری که بوی بد کد به آن اشاره دارد توسط یک دور بازخورد کوتاه نمایان می‌شود، زمانی که کد در قدم‌هایی کوتاه و کنترل‌شده بازسازی می‌شود و نتیجه برای یافتن بوی بد و بازسازی دوباره، ارزیابی می‌شود. از دید یک برنامه‌نویس که مسئول بازسازی کد است، بوی بد راهنمای او در

^۱ Rule Based System

فرایند بازسازی و انتخاب روش^۲ بازسازی است.[۱]

AST ۲-۱-۲

درخت نحو انتزاعی^۳ نمایش درختی از ساختار نحوی انتزاعی متن و اغلب کد منبع نوشته شده به زبان رسمی است. هر گره درخت نشان‌دهنده یک ساختار در متن است. انتزاعی بودن نحو^۴ به این معناست که تمام جزئیات ظاهر شده در نحو واقعی را نشان نمی‌دهد، بلکه فقط جزئیات ساختاری یا مرتبط با محتوا را نشان می‌دهد. این امر درختان نحو انتزاعی را از درختان نحو انضمامی که به طور سنتی درختان تجزیه نامیده می‌شوند، متمایز می‌کند. درختان تجزیه معمولاً توسط یک تجزیه‌کننده در طول فرایند ترجمه و کامپایل کد منبع ساخته می‌شوند. پس از ساخته شدن، اطلاعات اضافی با پردازش بعدی، به عنوان مثال، تجزیه و تحلیل زمینه‌ای به AST اضافه می‌شود.



شکل ۲-۱: مثالی از درخت نحو انتزاعی

²technique

³Abstract Syntax Tree

⁴Syntax

یک درخت نحو انتزاعی معمولاً به دلیل مراحل متوالی تجزیه و تحلیل توسط کامپایلر^۵، حاوی اطلاعات اضافی در مورد برنامه است. درخت نحو انتزاعی ها به دلیل ماهیت ذاتی زبان‌های برنامه‌نویسی و مستندات آن‌ها مورد نیاز هستند. زبان‌ها معمولاً به طور ذاتی مبهم هستند. برای جلوگیری از این ابهام، زبان‌های برنامه‌نویسی اغلب به عنوان گرامر مستقل از متن^۶ مشخص می‌شوند. با این حال، اغلب جنبه‌هایی از زبان‌های برنامه‌نویسی وجود دارند که بخشی از زبان هستند و در مشخصات آن مستند شده‌اند ولی یک گرامر مستقل از متن نمی‌تواند آن‌ها را بیان کند. این‌ها جزییاتی هستند که برای تعیین اعتبار و رفتارشان نیاز به یک زمینه دارند. حتی اگر یک زبان دارای مجموعه‌ای از انواع از پیش تعریف شده باشد، اعمال استفاده مناسب اغلب به زمینه‌ای نیاز دارد.

درخت نحو انتزاعی به شدت در طول تجزیه و تحلیل معنایی^۷ که در آن کامپایلر استفاده صحیح از عناصر برنامه و زبان را بررسی می‌کند، استفاده می‌شود. کامپایلر همچنین جداول نماد را بر اساس درخت نحو انتزاعی در طول تجزیه و تحلیل معنایی تولید می‌کند. پیمایش کامل درخت اجازه می‌دهد تا صحت برنامه تأیید شود. پس از تأیید صحت، درخت نحو انتزاعی به عنوان پایه‌ای برای تولید کد عمل می‌کند. درخت نحو انتزاعی اغلب برای تولید یک نمایش میانی^۸ که گاهی اوقات زبان میانی نامیده می‌شود، برای تولید کد استفاده می‌شود. [۲]

۲-۱-۳ RNN

راه حل اولیه پژوهشگران برای پردازش متن، ارائه مدل شبکه عصبی بازگشتی^۹ بوده است. این مدل در واقع با استفاده از ساختار بازگشتی بودن خود راه حلی برای وابسته بودن ورودی ها در اثر زمان پیدا کرده است. به طوری که مدل های دیگر شبکه ی عصبی داده ها را به صورت ترتیبی نمی بینند و خروجی نوروں ها^{۱۰} به یکدیگر وابسته نیستند.

⁵ Compiler

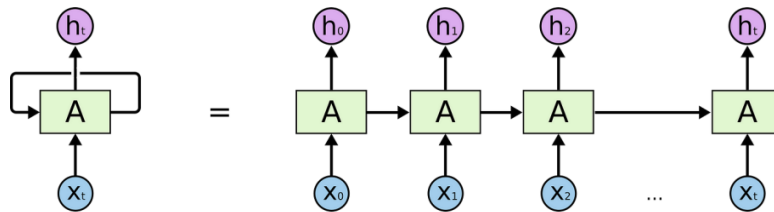
⁶ Context Free Grammar

⁷ Semantic Analysis

⁸ Information Representation

⁹ Recursive Neural Network

¹⁰ Neuron



شکل ۲-۲: معماری شبکه عصبی بازگشتی

در شکل ۲-۲ مشاهده می شود که هر ورودی در خروجی های بعدی خود تاثیر می گذارد که اینگونه حتی آخرین نورون هم قادر به اصلاح اولین نورون از طریق تابع ضرر^{۱۱} و محاسبه مشتق^{۱۲} می باشد. [۳]

LSTM ۴-۱-۲

به مرور زمان و با آموزش مدل های شبکه ی بازگشتی، محققان شاهد مشکل محوشدگی^{۱۳} و انفجار^{۱۴} مشتق در این نوع شبکه های عصبی بوده اند. یعنی به مرور زمان دچار فراموشی داده های قبلی و در نتیجه ساختار کلی متن را فراموش می کردند. سپس با ارائه مدل حافظه طولانی کوتاه مدت^{۱۵} توانستند بر این مشکل غلبه کنند به طوری که با تعریف دروازه های ورودی، فراموشی و خروجی داده های مورد نیاز را نگه می داشتند و داده های غیرقابل استفاده را از درون حافظه پاک می کردند

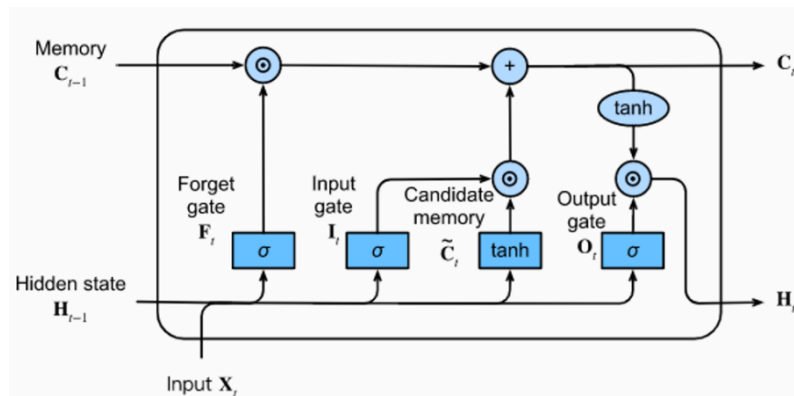
^{۱۱} Loss Function

^{۱۲} Gradient

^{۱۳} Vanishing Gradient

^{۱۴} Exploding Gradient

^{۱۵} Long Short Term Memory



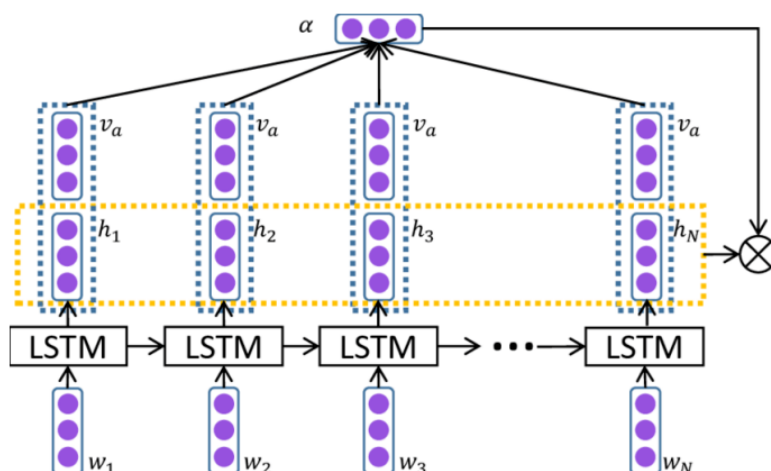
شکل ۲-۳: معماری شبکه عصبی حافظه طولانی کوتاه مدت

شکل ۲-۳ به خوبی ساختار درون هر لایه شبکه را نشان می دهد که در واقع نحوه ی دروازه ها را مشخص می کند. به واسطه تعریف دروازه های مجزا این شبکه چهار برابر شبکه ی عصبی بازگشتی ساده پارامتر دارد و در نتیجه از نظر محاسباتی چهار برابر کندتر از مدل های شبکه ی عصبی بازگشتی می باشد. [۴]

۲-۱-۵ Attention

مدل شبکه عصبی حافظه طولانی کوتاه مدت ارائه شده همچنان دچار فراموشی هایی به مرور زمان می شد و نمی توانست یک دنباله با طول بسیار زیاد را به خاطر بسپارد و همچنان مشکلی محوشدن مشتق مشاهده می شد. برای برطرف کردن مشکل ذکرشده پژوهشگران به ایده ی استفاده کردن از سازوکار توجه^{۱۶} رسیدند که به قدر خوبی تمام مشکلاتی که تا کنون مطرح شد را حل می کرد.

¹⁶Attention



شکل ۲-۴: معماری شبکه عصبی (Attention)

شکل ۲-۴ نشان می‌دهد که در پردازش هر ورودی، آن لایه می‌تواند به کل بخش‌های ورودی توجه کند و با استفاده از میانگین‌گیری، مقادیر هر کدام را به نحوی استفاده کند. [۵]

۲-۱-۶ Transformer

در جدیدترین پژوهش، مدل‌های مبدل^{۱۷} مطرح شده‌اند که علم پردازش زبان‌های طبیعی^{۱۸} را متحول کردند. آن‌ها در مقاله‌ی خود ساختاری جدید را معرفی کردند که دیگر ساختار این مدل‌ها بر پایه شبکه‌های عصبی بازگشتی نمی‌باشند. راه حل نوینی برای بردارهای وابسته به متن ارائه کرده‌اند که نویسندگان این مقاله با ارائه سازوکار توجه به خود واقع به این معناست اگر دو کلمه‌ی هم‌شکل با معنای متفاوت درون متن قرار بگیرند، این سازوکار متوجه تفاوت این دو کلمه خواهد شد. آن‌ها همچنین فرآیند وابسته بودن هر بخش در شبکه‌های عصبی بازگشتی که منجر به کند بودن آن می‌شد را با استفاده از مدل جدید خود کاملاً به طور موازی درآوردند که بسیار به کار سرعت می‌بخشید. قدم بزرگ دیگر این ساختار، آموزش دیدن مدل‌های کارآمدی می‌باشند که در واقع با استفاده از این مدل‌ها که بر روی حجم بسیار عظیمی از متن‌ها آموزش دیده‌اند، می‌توانیم از وزن^{۱۹}‌های آموزش دیده‌ی آن‌ها در مسائل مختلف استفاده کنیم و مدل‌ها را برای انجام وظایف

¹⁷Transformer

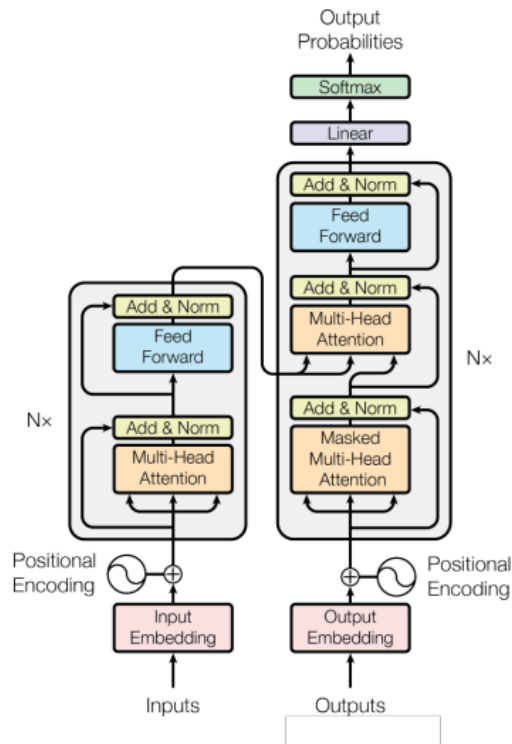
¹⁸Natural Language Processing

¹⁹weight

جدید بدون نیاز به آموزش دوباره از ابتدا به کار بگیریم. این ویژگی به‌ویژه در مواقعی که داده‌های آموزشی محدود هستند، بسیار مفید است. به عنوان مثال، مدل‌های مبدل که بر روی مقادیر عظیمی از داده‌های عمومی آموزش دیده‌اند، می‌توانند با تنظیمات اندک و استفاده از وزن‌های یادگیری‌شده، به سرعت به مسائل خاصی مانند ترجمه، خلاصه‌سازی متن، یا پاسخ به سوالات پاسخ دهند. این مدل‌ها همچنین به دلیل ساختار مقیاس‌پذیری که دارند، امکان پردازش موازی را فراهم می‌کنند که این امر منجر به کاهش زمان آموزش و پیش‌بینی می‌شود. علاوه بر این، مدل‌های مبدل به دلیل حذف محدودیت‌های مربوط به وابستگی‌های طولانی‌مدت در متن، قادر به درک بهتر و عمیق‌تری از توالی‌های طولانی هستند.

مدل‌های مبدل، از جمله معروف‌ترین آن‌ها یعنی BERT، GPT و T5، به عنوان پایه‌ای برای بسیاری از کاربردهای عملی در پردازش زبان طبیعی مورد استفاده قرار گرفته‌اند. این مدل‌ها توانسته‌اند در بسیاری از معیارهای استاندارد، عملکردی بهتر از مدل‌های پیشین ارائه دهند و در واقع، انقلابی در این حوزه به وجود آورده‌اند.

در نهایت، مبدل‌ها با ارائه رویکردی جدید به پردازش زبان طبیعی، امکان توسعه سیستم‌های هوشمندتر و کارآمدتر را فراهم کرده‌اند که می‌توانند در طیف وسیعی از کاربردها از جمله ترجمه ماشینی، تولید متن، تحلیل احساسات و بسیاری دیگر به کار گرفته شوند. این مدل‌ها به دلیل انعطاف‌پذیری و قدرت پیش‌بینی بالای خود، همچنان در حال پیشرفت و بهبود هستند و تحقیقات بیشتری در این زمینه در حال انجام است تا از توانایی کامل آن‌ها بهره‌برداری شود. [۵]



شکل ۲-۵: معماری (Transformer)

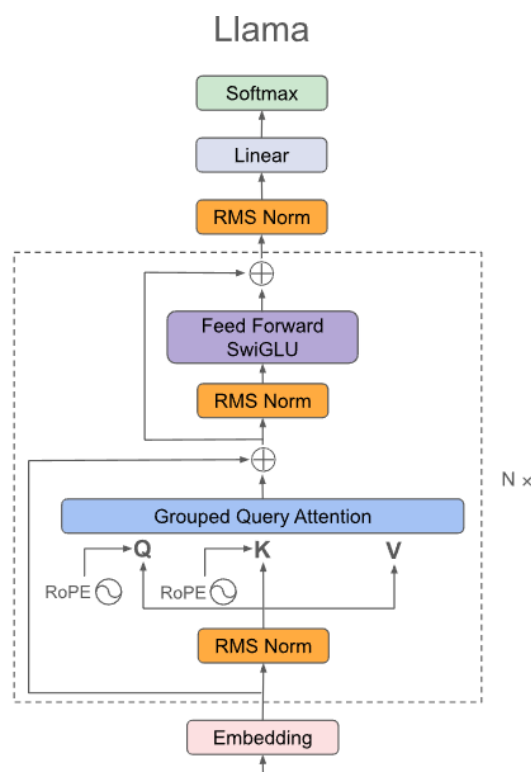
شکل ۲-۵ معماری مدل Transformer را نشان میدهد که لایه هر مرحله چگونه است.

۲-۱-۷ LLaMA

مدل LLaMA^{۲۰} به دلیل استفاده از مکانیسم توجه، قابلیت پردازش کارآمد و موازی داده‌های متنی را دارد. در ساختار، مکانیسم خودتوجهی^{۲۱} به مدل اجازه می‌دهد تا به تمامی بخش‌های ورودی توجه کند و روابط بین کلمات را درک کند. این امر به مدل امکان می‌دهد وابستگی‌های طولانی‌مدت در متن را شناسایی کند. محاسبات توجه با استفاده از انجام می‌شود که به بهبود پایداری عددی در پردازش داده‌های بزرگ کمک می‌کند.

^{۲۰}Large Language Model Meta AI

^{۲۱}Self-Attention



شکل ۲-۶: معماری (LLaMA)

شکل ۲-۶ معماری مدل LLaMA را نشان می‌دهد که از لایه‌های متوالی تشکیل شده که شامل بخش کدگذار^{۲۲} و کدگشا^{۲۳} است. در کاربردهایی مانند ترجمه، هر دو بخش استفاده می‌شوند، اما برای تولید متن معمولاً فقط کدگشا کافی است. هر لایه شامل مکانیزم خودتوجهی و لایه‌های شبکه عصبی کاملاً متصل است. برای تثبیت و تسریع آموزش مدل، از نرمال‌سازی لایه استفاده می‌شود. همچنین، اتصالات باقیمانده به جلوگیری از مشکل کمک می‌کنند و اطلاعات را به‌طور موثرتری در لایه‌های مدل منتقل می‌کنند. مدل LLaMA با اندازه‌های مختلف و تعداد پارامترهای متفاوت عرضه می‌شود، که این امر به کاربران اجازه می‌دهد مدل مناسب را بر اساس نیاز و منابع محاسباتی خود انتخاب کنند. این مدل‌ها به دلیل طراحی بهینه و استفاده از معماری، از کارایی محاسباتی بالایی برخوردارند و قادر به پردازش موازی داده‌ها هستند، که این امر باعث افزایش سرعت و کاهش زمان آموزش می‌شود. بهبودهای خاص LLaMA ممکن است شامل روش‌های کاهش پیچیدگی زمانی و مکانی و استفاده

^{۲۲} Encoder

^{۲۳} Decoder

از پیش‌آموزش‌های خاص برای افزایش دقت^{۲۴} در وظایف خاص باشد. این مدل‌ها به‌گونه‌ای طراحی شده‌اند که می‌توانند در محیط‌های مختلفی از جمله دستگاه‌های با قدرت محاسباتی محدود اجرا شوند، که این ویژگی آن‌ها را برای کاربردهای صنعتی و تحقیقاتی که نیاز به پردازش سریع و دقیق دارند، بسیار مفید می‌سازد. در مجموع، LLaMA با استفاده از ساختار و بهینه‌سازی‌های مختلف، یک ابزار قدرتمند برای پردازش زبان طبیعی ارائه می‌دهد که می‌تواند در طیف وسیعی از کاربردها مورد استفاده قرار گیرد.[۶]

²⁴Accuracy

۲-۲ کارهای انجام شده با روش‌های پیشین

۱-۲-۲ سیستم‌های قانون محور

سیستم‌های قانون محور یا یکی از رویکردهای اصلی برای تشخیص بوی کد در صنعت نرم‌افزار هستند. این سیستم‌ها بر اساس تعدادی از قوانین یا قواعد برنامه نویسی شده توسط توسعه‌دهندگان عمل می‌کنند. این قواعد معمولاً بر اساس اصول نگارش کد، الگوهای طراحی خوب و استانداردهای برنامه نویسی تعریف شده‌اند. [۷]

تعریف قوانین

در این مرحله، توسعه‌دهندگان قوانین مربوط به بوی کد را تعریف می‌کنند. این قوانین ممکن است شامل الگوهای طراحی، اصول نگارش کد، استفاده از متغیرهای مفهومی، اصول و سایر استانداردهای برنامه نویسی باشند.

پیاده‌سازی قوانین

قوانین تعریف شده باید به صورت قابل اجرا در سیستم پیاده‌سازی شوند. این به معنای تبدیل قوانین بوی کد به قواعد قابل اجرا توسط سیستم است تا بتواند کد را تحلیل کرده و مشکلات را شناسایی کند.

تحلیل کد و اعمال تغییرات

سیستم قوانین و قواعد تعریف شده را بر روی کد منبع اجرا می‌کند. سپس با توجه به این قوانین، بوی کد را تشخیص داده و مشکلات مربوط به کیفیت کد را شناسایی می‌کند. این تحلیل معمولاً شامل اعلام هشدارها، پیشنهادات بهبود و گزارش‌های مرتبط با بوی کد است. بر اساس نتایج حاصل از تحلیل، توسعه‌دهندگان می‌توانند تغییرات لازم را در کد اعمال کنند تا بهبودهای مورد نیاز را انجام دهند. این شامل اصلاحات نگارشی، بهبودهای ساختاری و سایر تغییراتی است که برای بهبود کیفیت کد لازم است.

۲-۲-۲ یادگیری ماشین

در طول زمان ابزار شناسایی بوی کد نیز پیشنهاد شده است که بیشتر آن‌ها به عنوان مبتنی بر اکتشافی در نظر گرفته می‌شوند. آن‌ها یک فرآیند دو مرحله‌ای را اجرا می‌کنند که در آن ابتدا مجموعه‌ای از متریک‌ها محاسبه می‌شود و سپس برخی آستانه‌ها^{۲۵} برای تمایز بین کلاس‌های با بو و بدون بو اعمال می‌شود. این ابزارها از نظر الگوریتم‌های خاص مورد استفاده برای شناسایی بوی کد و متریک‌های بهره‌بردار شده با یکدیگر تفاوت دارند. اگرچه نشان داده شده است که این ابزارها در زمینه دقت توصیه‌ها عملکرد معقولی دارند، ولی کارهای قبلی تعدادی محدودیت مهم را نشان داده‌اند که ممکن است استفاده عملی از این ابزارها را محدود کند. به خصوص، بوهای کدی که توسط ابزارهای موجود شناسایی می‌شوند می‌توانند به صورت ذهنی توسط توسعه‌دهندگان تفسیر شوند. به علاوه، توافق بین آن‌ها پایین است. مهم‌تر از همه، بیشتر آن‌ها نیاز به تعیین آستانه‌هایی دارند تا مؤلفه‌های بوی کد را از مؤلفه‌های بدون بوی تمایز بدهند و به طور طبیعی، انتخاب آستانه‌ها به شدت بر دقت آن‌ها تأثیر می‌گذارد.

به دلیل تمامی این دلایل، یک روند جدید به سمت استفاده از روش‌های یادگیری ماشین^{۲۶} برای مقابله با این مشکل پیش رفته است. در این سناریو، یک روش نظارت‌شده استفاده می‌شود که مجموعه‌ای از متغیرهای مستقل برای پیش‌بینی ارزش یک متغیر وابسته (یعنی بوی کد یک کلاس) با استفاده از یک طبقه‌بند یادگیری ماشین به کار می‌روند. مدل می‌تواند با استفاده از مقدار کافی داده‌های موجود از پروژه تحت بررسی، یعنی استراتژی درون‌پروژه‌ای، یا با استفاده از داده‌های پروژه‌های نرم‌افزاری دیگر، یعنی استراتژی بین‌پروژه‌ای، آموزش داده شود. این رویکردها به وضوح از روش‌های مبتنی بر اکتشافی متفاوت هستند، زیرا آن‌ها به طبقه‌بندها برای تمایز بوی کد کلاس‌ها تکیه می‌کنند نه بر آستانه‌های از پیش تعریف‌شده بلکه بر روی معیارهای^{۲۷} محاسبه‌شده است. [۸][۹]

²⁵ Threshold²⁶ Machine Learning²⁷ Metric

۳-۲-۲ یادگیری عمیق

در حوزه شناسایی بوی کد، یادگیری عمیق^{۲۸} به عنوان یک رویکرد نوآورانه و قدرتمند در حال ظهور است. یادگیری عمیق، به ویژه شبکه‌های عصبی عمیق، می‌تواند به طور خودکار ویژگی‌های پیچیده و غیرخطی را از داده‌های ورودی استخراج کند که این امر به بهبود دقت و کارایی در شناسایی بوی کد کمک می‌کند.

یادگیری عمیق، برخلاف روش‌های سنتی یادگیری ماشین که به ویژگی‌های دستی یا از پیش تعریف‌شده متکی هستند، قابلیت یادگیری و استخراج خودکار ویژگی‌ها را دارد. این امر می‌تواند به کاهش وابستگی به آستانه‌های ثابت و بهبود دقت مدل در شناسایی بوی کد کمک کند. شبکه‌های عصبی عمیق مانند شبکه‌های پیچشی^{۲۹} یا شبکه‌های بازگشتی به طور خودکار ویژگی‌های سطح بالا و پیچیده را از داده‌های خام استخراج می‌کنند. این فرآیند به مدل اجازه می‌دهد تا روابط پیچیده‌تر و الگوهای غیرخطی را که ممکن است توسط روش‌های سنتی کشف نشود، شناسایی کند. یادگیری عمیق، معماری شبکه شامل لایه‌های متعدد (لایه‌های پنهان) است که هر لایه به عنوان یک استخراج‌کننده ویژگی عمل می‌کند. لایه‌های اولیه ویژگی‌های ساده‌تر و لایه‌های عمیق‌تر ویژگی‌های پیچیده‌تر و انتزاعی‌تر را استخراج می‌کنند. یکی از مزایای اصلی یادگیری عمیق این است که نیاز به مهندسی ویژگی دستی را کاهش می‌دهد، چرا که مدل به طور خودکار ویژگی‌های مناسب را از داده‌ها استخراج و یاد می‌گیرد.

مدل‌های یادگیری عمیق نیاز به مقدار زیادی داده برای آموزش دارند تا بتوانند ویژگی‌های مفید را به خوبی یاد بگیرند. می‌توان از روش‌هایی مانند یادگیری انتقالی استفاده کرد که در آن مدل‌های از پیش آموزش‌دیده بر روی مجموعه داده‌های بزرگ‌تر به پروژه‌های خاص تطبیق داده می‌شوند. این روش به خصوص در شرایطی که داده‌های محدود موجود است، مفید است. ارزیابی مدل‌های یادگیری عمیق نیز مشابه روش‌های سنتی یادگیری ماشین است، اما می‌تواند به دقت بیشتری دست یابد. معیارهایی مانند دقت، فراخوان، و امتیاز F1 برای ارزیابی مدل‌ها استفاده می‌شوند. همچنین، استفاده از روش‌های اعتبارسنجی متقابل می‌تواند به ارائه ارزیابی دقیق‌تر کمک کند. [۱۰][۱۱]

²⁸ Deep Learning²⁹ Convolutional Neural Network

فصل سوم

شرح مسئله

۳. شرح مسئله

در پروژه حاضر، هدف اصلی توسعه یک سیستم هوشمند است که توانایی شناسایی و دسته‌بندی بوی کدها در نمونه‌های کد را داشته باشد. بوی کدها به نشانه‌هایی در کد اشاره دارند که معمولاً نشان‌دهنده مشکلات طراحی یا پیاده‌سازی هستند و می‌توانند منجر به کاهش کیفیت کد، افزایش پیچیدگی و دشواری در نگهداری و توسعه شوند. شناسایی بوی کدها به توسعه‌دهندگان کمک می‌کند تا پیش از آن که مشکلات عمده‌ای در کد بروز کنند، آنها را شناسایی و رفع کنند.

طبق جدول ۱-۳ در این پروژه، ۲۸ نوع بوی کد مختلف مورد شناسایی قرار می‌گیرند. هر یک از این بوها ویژگی‌ها و نشانه‌های خاص خود را دارند که باید توسط سیستم شناسایی شوند. برخلاف دسته‌بندی تک‌برچسبی^۱ که هر نمونه تنها یک برچسب دریافت می‌کند، در دسته‌بندی چندبرچسبی^۲ هر نمونه کد می‌تواند چندین برچسب را به صورت همزمان داشته باشد. به عبارت دیگر، هر نمونه کد ممکن است شامل چندین نوع بوی کد باشد و سیستم باید بتواند تمامی بوی کدهای موجود در یک نمونه را شناسایی و برچسب‌گذاری کند.

ورودی سیستم شامل مجموعه‌ای از نمونه‌های کد است که می‌تواند از زبان‌های برنامه‌نویسی مختلف باشد. هر نمونه کد باید مورد بررسی قرار گیرد تا مشخص شود که کدام یک از ۲۸ بوی کد در آن وجود دارد. خروجی سیستم شامل مجموعه‌ای از برچسب‌ها برای هر نمونه کد است که نشان می‌دهد کدام بوی کدها در آن نمونه وجود دارند. این خروجی می‌تواند به توسعه‌دهندگان اطلاعات ارزشمندی برای بهبود کیفیت کد و رفع مشکلات ارائه دهد.

^۱ Singel-Label Classification

^۲ Multi-Label Classification

جدول ۳-۱: لیست بوی بد کد و توضیحات آن‌ها

نام بوی بد کد	توضیحات
Missing Hierarchy	نداشتن سلسله‌مراتب مناسب در ساختار کد
Long Parameter List	لیست پارامترهای طولانی که درک و استفاده از متد را دشوار می‌کند
Unnecessary Abstraction	وجود انتزاعاتی که هیچ نیازی به آن‌ها نیست
Imperative Abstraction	استفاده از انتزاعاتی که از سبک برنامه‌نویسی دستوری پیروی می‌کنند
Empty Catch Clause	بلوک‌های کچ خالی که خطاها را به درستی مدیریت نمی‌کنند
Deficient Encapsulation	عدم مخفی‌سازی کافی داده‌ها و متدها در کلاس‌ها
Long Identifier	استفاده از شناسه‌های طولانی که خوانایی کد را کاهش می‌دهد
Multifaceted Abstraction	انتزاعاتی که وظایف متعددی را انجام می‌دهند
Wide Hierarchy	سلسله‌مراتب گسترده‌ای که مدیریت را پیچیده می‌کند
Complex Conditional	شرط‌های پیچیده که کد را سخت‌خوان و دشوار می‌سازد
Rebellious Hierarchy	سلسله‌مراتبی که از اصول طراحی پیروی نمی‌کند
Magic Number	استفاده از اعداد جادویی که معنا را در کد مخفی می‌کنند
Missing default	نداشتن مورد پیش‌فرض در ساختارهای انتخاب
Long Method	متدهای طولانی که فهم و نگهداری آن‌ها سخت است
Broken Modularization	ماژول‌های شکسته که وظایف را به درستی تقسیم نمی‌کنند
Broken Hierarchy	سلسله‌مراتبی که به درستی ساختار نیافته است
Unutilized Abstraction	انتزاعاتی که به درستی استفاده نمی‌شوند
Long Statement	جملات برنامه‌نویسی طولانی که خوانایی را کاهش می‌دهد
Cyclic-Dependent Modularization	ماژول‌هایی که به صورت چرخه‌ای به هم وابسته‌اند
Multipath Hierarchy	سلسله‌مراتبی که مسیرهای متعددی دارد

سلسله مراتب عمیقی که پیگیری آن دشوار است	Deep Hierarchy
ماژول‌هایی که به‌طور متمرکز عمل می‌کنند و وابستگی زیادی دارند	Hub-like Modularization
ماژول‌هایی که به اندازه کافی تقسیم نشده‌اند	Insufficient Modularization
سلسله مراتبی که به صورت چرخه‌ای به هم وابسته‌اند	Cyclic Hierarchy
کپسوله‌سازی که به درستی بهره‌برداری نشده است	Unexploited Encapsulation
فراخوانی توابع انتزاعی از سازنده	Abstract Function Call From Constructor
متدهای پیچیده که درک و نگهداری آن‌ها دشوار است	Complex Method

شناسایی بوی کدها به دلیل تنوع و پیچیدگی نشانه‌ها و ویژگی‌های هر بوی کد، چالش برانگیز است. برخی از بوها ممکن است به صورت ترکیبی از چندین نشانه ظاهر شوند یا در بخش‌های مختلف کد پراکنده باشند. علاوه بر این، دسته‌بندی چندبرچسبی نیازمند الگوریتم‌ها و مدل‌های پیچیده‌ای است که بتوانند به درستی و با دقت بالا برچسب‌های متعدد را به هر نمونه کد اختصاص دهند. [۱۲][۱۳]

فصل چهارم

روش پیشنهادی

۴. روش پیشنهادی

۴-۱ پردازش و جمع آوری داده

برای آموزش مدل، به داده‌های برچسب‌خورده نیاز بود. در این پروژه از داده‌های موجود در لینک زیر استفاده شده است:

<https://github.com/liyichen1234/HMML>

این مجموعه داده‌ها شامل ۳۹۶۴۷ پروژه جاوا می‌باشد که هر پروژه حاوی ۲ تا ۱۰ فایل کد جاوا است. برای هر پروژه بین یک تا پنج برچسب بوی کد تخصیص داده شده است. [۱۴]

در ابتدا، تمامی فایل‌های کد مرتبط با هر پروژه ادغام شدند و سپس برچسب‌های اختصاص داده شده به هر پروژه به صورت یک رشته ترکیبی از برچسب‌ها تبدیل شدند. در نهایت، یک جدول با فرمت فایل csv ایجاد شد. در ستون اول این جدول، آدرس فایل ادغام‌شده پروژه‌ها و در ستون دوم، نام بوی کدهای مربوط به آن پروژه قرار گرفت.

برای آموزش مدل، هر فایل کد باید با استفاده از آدرس موجود در جدول واژه‌بندی^۱ شده و در کارت گرافیک بارگذاری شود. همچنین، برچسب‌های هر پروژه به صورت یک آرایه به طول ۲۸ تبدیل می‌شوند که در آن هر خانه با مقدار ۰ یا ۱ مشخص می‌کند که آیا بوی کد مورد نظر در آن پروژه جاوا وجود دارد یا خیر. [۱۵]

۴-۲ تعریف مدل

در یادگیری عمیق مدل به عنوان ساختار محاسباتی اصلی عمل می‌کند که ورودی‌ها را بر اساس داده‌های مشاهده شده به خروجی‌ها نگاشت می‌کند. مدل با استفاده از یک مجموعه داده آموزش

^۱tokenize

می‌بیند و الگوها، روابط و قوانینی را یاد می‌گیرد که می‌تواند برای پیش‌بینی داده‌های جدید و نادیده گرفته‌شده تعمیم یابد. مدل‌های مدرن، به‌ویژه آن‌هایی که بر روی معماری‌های یادگیری عمیق مانند مبدل‌ها ساخته شده‌اند، در شناسایی الگوهای پیچیده در داده‌ها بسیار قدرتمند هستند. با این حال، اغلب نیاز به سفارشی‌سازی و توسعه دارند تا برای وظایف خاص مناسب شوند. این مقاله به فرآیند بارگذاری یک مدل از پیش‌آموزش‌دیده، به‌ویژه یک مدل زبان بزرگ (LLM)، و سپس تطبیق آن برای یک وظیفه طبقه‌بندی چندبرچسبی از طریق اتصال لایه نهایی آن به یک شبکه عصبی چندلایه^۲ برای طبقه‌بندی در ۲۸ برچسب مختلف می‌پردازد. [۱۶]

۱-۲-۴ بارگذاری مدل از پیش‌آموزش‌دیده

یک مدل زبان بزرگ، مانند GPT یا Llama، معمولاً بر روی حجم عظیمی از داده‌های متنی از پیش‌آموزش‌دیده تا بتواند زبان طبیعی را درک و تولید کند. این مدل‌ها با معماری‌های عمیق و چندلایه طراحی شده‌اند که می‌توانند ویژگی‌های پیچیده زبان‌شناسی را شناسایی کنند. فاز پیش‌آموزش به مدل اجازه می‌دهد تا وظایف مختلف زبانی مانند طبقه‌بندی متن، تحلیل احساسات و ترجمه زبان را با استفاده از بازنمایی‌های داخلی زبان یاد بگیرد.

برای شروع، مدل از پیش‌آموزش‌دیده در محیط محاسباتی بارگذاری می‌شود. این مرحله شامل وارد کردن معماری مدل و وزن‌های یادگرفته‌شده آن است. فرآیند بارگذاری می‌تواند با استفاده از فریم‌ورک‌های یادگیری عمیق محبوب مانند TensorFlow یا PyTorch انجام شود که API هایی برای بارگذاری آسان این مدل‌های از پیش‌آموزش‌دیده ارائه می‌دهند. به عنوان مثال، در PyTorch، این کار با یک فرمان ساده مانند `model = AutoModel.from_pretrained('model_name')` انجام می‌شود، که در آن `'model_name'` مدل پیش‌آموزش‌دیده مورد نظر را مشخص می‌کند.

پس از بارگذاری، مدل زبان بزرگ می‌تواند برای وظایف خاص تنظیم یا تطبیق داده شود. اگرچه مدل قبلاً قادر به درک و تولید زبان است، اما ممکن است به طور مستقیم برای وظایفی که نیاز به خروجی‌های ساختاریافته دارند، مانند طبقه‌بندی چندبرچسبی، مناسب نباشد. بنابراین، باید لایه‌های اضافی یا مولفه‌های دیگر اضافه شود.

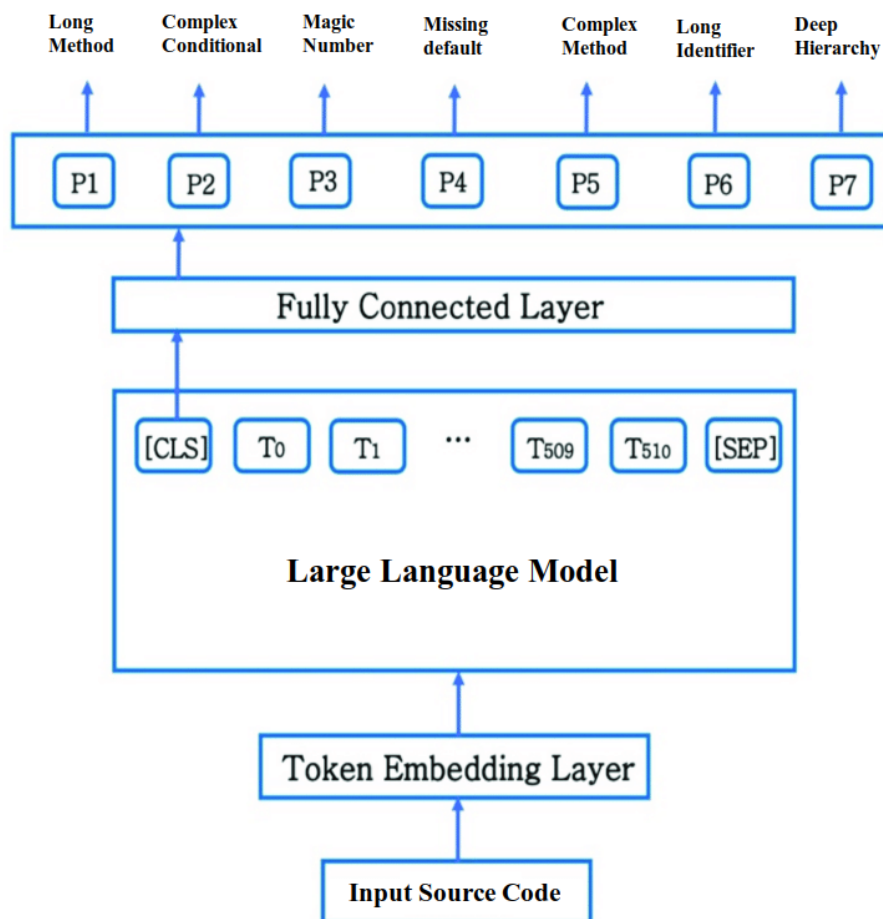
²Multi Layer Perceptron

۲-۲-۴ اتصال مدل زبان بزرگ به یک شبکه عصبی چندلایه

برای وظیفه مورد نظر طبقه‌بندی چندبرچسبی با ۲۸ برچسب لایه نهایی مدل زبان بزرگ باید تطبیق داده شود. به طور معمول، خروجی یک مدل زبان بزرگ یک بازنمایی با ابعاد بالا از متن ورودی است که اغلب به عنوان "وضعیت پنهان" یا "تعبیر" شناخته می‌شود. این بازنمایی شامل اطلاعات غنی است اما به طور مستقیم برای وظایف طبقه‌بندی قابل تفسیر نیست. برای تبدیل این بازنمایی به مجموعه‌ای از طبقه‌بندی‌ها، وضعیت پنهان نهایی مدل زبان بزرگ به یک شبکه عصبی چندلایه متصل می‌شود. شبکه عصبی چندلایه یک شبکه عصبی ساده اما قدرتمند است که از یک یا چند لایه کاملاً متصل تشکیل شده است. شبکه عصبی چندلایه بازنمایی با ابعاد بالا از مدل زبان بزرگ را دریافت کرده و آن را برای تولید خروجی مطلوب پردازش می‌کند. در اینجا نحوه انجام این فرآیند آمده است:

۱. استخراج بازنمایی: خروجی مدل زبان بزرگ، که اغلب یک بردار با هزاران بعد است، به عنوان ورودی به شبکه عصبی چندلایه ارسال می‌شود. این بازنمایی جوهره متن ورودی را به شکلی که شبکه عصبی چندلایه بتواند پردازش کند، ثبت می‌کند.
۲. طراحی شبکه عصبی چندلایه: شبکه عصبی چندلایه با یک یا چند لایه طراحی شده است. لایه نهایی شبکه عصبی چندلایه باید دارای ۲۸ نرون (یا گره) باشد که با ۲۸ برچسب در وظیفه طبقه‌بندی مطابقت دارد. هر نرون در لایه نهایی یک مقدار را خروجی می‌دهد که نمایانگر احتمال تعلق ورودی به آن برچسب خاص است. این خروجی‌ها معمولاً با استفاده از یک تابع فعال‌سازی sigmoid تفسیر می‌شوند.
۳. خروجی چندبرچسبی: برخلاف وظایف طبقه‌بندی سنتی که در آن تنها یک برچسب پیش‌بینی می‌شود، طبقه‌بندی چندبرچسبی امکان فعال بودن چندین برچسب به طور همزمان را فراهم می‌کند. لایه خروجی شبکه عصبی چندلایه به گونه‌ای طراحی شده است که با اعمال تابع sigmoid به هر یک از ۲۸ خروجی، این کار را انجام دهد. این تابع هر خروجی را به احتمالی بین ۰ و ۱ نگاشت می‌کند، که در آن یک آستانه (معمولاً 0.5) برای تعیین فعال بودن یا نبودن هر برچسب اعمال می‌شود.

۴. آموزش مدل توسعه یافته: کل مدل شامل مدل زبان بزرگ و شبکه عصبی چندلایه متصل شده باید بر روی یک مجموعه داده چندبرچسبی آموزش داده شود. در این فاز، مدل یاد می گیرد که پارامترهای خود را، به ویژه آنهایی که در شبکه عصبی چندلایه هستند، تنظیم کند تا اختلاف بین پیش بینی های خود و برچسب های واقعی در مجموعه داده را به حداقل برساند. این کار معمولاً با استفاده از یک تابع زیان طراحی شده برای طبقه بندی چندبرچسبی، مانند آنترپی^۳ متقاطع باینری، انجام می شود.



شکل ۴-۱: نحوه بهبود دادن مدل

³Entropy

همانطور که در شکل ۴-۱ ملاحظه میکنید، با بارگذاری یک مدل زبان بزرگ از پیش‌آموزش‌دیده و اتصال لایه نهایی آن به یک شبکه عصبی چندلایه، می‌توانیم مدل را به طور مؤثری برای وظایف پیچیده‌ای مانند طبقه‌بندی چندبرچسبی با ۲۸ برچسب تطبیق دهیم. این رویکرد از توانایی‌های قدرتمند درک زبان مدل‌های زبان بزرگ بهره می‌برد و در عین حال آن‌ها را برای مدیریت خروجی‌های ساختاریافته خاص گسترش می‌دهد. نتیجه یک مدل بسیار تخصصی است که می‌تواند چندین برچسب را برای یک ورودی پیش‌بینی کند و آن را به یک ابزار همه‌کاره برای وظایف مختلف پردازش زبان طبیعی تبدیل می‌کند. [۱۷]

۳-۴ آموزش مدل

توسعه و بهینه‌سازی مدل‌های زبانی به‌طور فزاینده‌ای پیچیده شده است. یکی از دستاوردهای برجسته در این مسیر، ظهور مدل‌های زبانی بزرگ (LLMs) است که توانایی فوق‌العاده‌ای در درک و تولید متن شبیه به انسان دارند. با این حال، آموزش این مدل‌ها چالش‌های زیادی را به همراه دارد، به‌ویژه از نظر نیاز به منابع محاسباتی و کارایی در بهینه‌سازی. در میان روش‌های مختلفی که برای مقابله با این چالش‌ها توسعه یافته‌اند، QLoRA^۴ (تطبیق ماتریس‌های کم‌رتبه با استفاده از فشرده‌سازی^۵) به‌عنوان یک رویکرد امیدوارکننده مطرح شده است. این مقاله به مفهوم تعریف مدل می‌پردازد و جزئیات آموزش مدل‌های زبانی بزرگ با استفاده از QLoRA را بررسی می‌کند.

۱-۳-۴ چالش‌های آموزش مدل‌های زبانی بزرگ

آموزش مدل‌های زبانی بزرگ یک وظیفه پیچیده است که نیازمند قدرت محاسباتی، حافظه و داده‌های عظیم است. با افزایش اندازه این مدل‌ها، نیاز به منابع سخت‌افزاری نیز افزایش می‌یابد و آموزش آن‌ها بر روی سخت‌افزارهای استاندارد دشوار می‌شود. علاوه بر این، بهینه‌سازی این مدل‌ها برای وظایف خاص یا حوزه‌های خاص به تلاش محاسباتی بیشتری نیاز دارد که اغلب به هزینه‌های بالایی منجر می‌شود.

^۴Quantized Low-Ranked Adaptation of Language Models

^۵Quantize

یکی از چالش‌های کلیدی در آموزش مدل‌های زبانی بزرگ، تعادل بین اندازه مدل و کارایی محاسباتی است. مدل‌های بزرگ‌تر معمولاً عملکرد بهتری دارند اما در عین حال منابع بیشتری را مصرف می‌کنند، که باعث می‌شود پیاده‌سازی آن‌ها در محیط‌هایی با منابع محدود دشوار شود. برای مقابله با این چالش‌ها، محققان روش‌های مختلفی را توسعه داده‌اند، از جمله فشردگی و تطبیق ماتریس‌های کم‌رتبه، که هدف آن‌ها کاهش بار محاسباتی بدون کاهش قابل توجه عملکرد مدل است.

۴-۳-۲ معرفی QLoRA

QLoRA (تطبیق ماتریس‌های کم‌رتبه با استفاده از فشردگی) یک روش است که برای بهبود کارایی آموزش مدل‌های زبانی بزرگ با ترکیب دو مفهوم قدرتمند طراحی شده است: فشردگی و تطبیق ماتریس‌های کم‌رتبه.

۱. فشردگی: این فرآیند شامل کاهش وزن‌های مدل از فرمت‌های با دقت بالا (مانند فلوتینگ‌پوینت ۳۲ بیتی) به فرمت‌های با دقت پایین‌تر (مانند اعداد صحیح ۸ بیتی) است. فشردگی به‌طور قابل توجهی حجم حافظه و نیازهای محاسباتی مدل را کاهش می‌دهد و به آن امکان می‌دهد تا به‌طور کارآمدتری بر روی سخت‌افزارهای کم‌قدرت اجرا شود. در حالی که فشردگی ممکن است منجر به کاهش جزئی در دقت مدل شود، این کاهش در مقایسه با مزایای به‌دست‌آمده از نظر کارایی منابع اغلب ناچیز است.

۲. تطبیق ماتریس‌های کم‌رتبه: این روش شامل تقریب ماتریس‌های وزن مدل به‌عنوان حاصل ضرب دو ماتریس کوچک‌تر با رتبه کمتر است. با کاهش رتبه، تعداد پارامترها به‌طور قابل توجهی کاهش می‌یابد که به نوبه خود بار محاسباتی در طول آموزش را کاهش می‌دهد. تطبیق ماتریس‌های کم‌رتبه امکان بهینه‌سازی کارآمد مدل‌های زبانی بزرگ را فراهم می‌کند و بر روی پارامترهای با بیشترین اطلاعات تمرکز می‌کند، در نتیجه سرعت آموزش را افزایش داده و مصرف منابع را کاهش می‌دهد.

۴-۳-۳ آموزش مدل‌های زبانی بزرگ با QLoRA

QLoRA این دو روش را ترکیب می‌کند تا فرآیند آموزش مدل‌های زبانی بزرگ را به‌طور قابل توجهی

کارآمدتر سازد. در طول آموزش، وزن‌های مدل به دقت پایین‌تری فشرده می‌شوند، که نیازهای حافظه و محاسباتی را کاهش می‌دهد. به‌طور هم‌زمان، تطبیق ماتریس‌های کم‌رتبه به ماتریس‌های وزن مدل اعمال می‌شود، که تعداد پارامترهایی که نیاز به بهینه‌سازی دارند را بیشتر کاهش می‌دهد. این ترکیب امکان آموزش و بهینه‌سازی سریع‌تر مدل‌های زبانی بزرگ را فراهم می‌کند و آن‌ها را قابل اجرا بر روی سخت‌افزارهای با منابع محدود می‌سازد. QLoRA همچنین امکان پیاده‌سازی مدل‌های زبانی بزرگ در برنامه‌های واقعی را فراهم می‌کند، جایی که کارایی محاسباتی حیاتی است. با کاهش نیازهای منابع، QLoRA نه تنها هزینه آموزش مدل‌های زبانی بزرگ را کاهش می‌دهد، بلکه آن‌ها را برای طیف وسیع‌تری از محققان و توسعه‌دهندگان قابل دسترس‌تر می‌کند. [۱۸]

فصل پنجم

ارزیابی و معیارهای سنجش عملکرد

۵. ارزیابی و معیارهای سنجش عملکرد

ارزیابی مدل‌ها یکی از مراحل حیاتی در فرآیند یادگیری عمیق است چرا که تنها از طریق ارزیابی می‌توان عملکرد مدل را سنجید، نقاط قوت و ضعف آن را شناسایی کرد و در نهایت تصمیمات لازم برای بهبود مدل را اتخاذ نمود.

در این فصل، ابتدا به معرفی معیارهای مختلف برای ارزیابی مدل‌های دسته‌بندی چندبرچسبی می‌پردازیم. این معیارها به ما کمک می‌کنند تا از زوایای مختلفی عملکرد مدل را بسنجیم و درک عمیق‌تری از نحوه عملکرد مدل در مسائلی با چندین برچسب همزمان پیدا کنیم. سپس، با یک مثال عملی نحوه استفاده از این معیارها برای ارزیابی مدل را توضیح خواهیم داد. این مثال به شما کمک خواهد کرد تا بهتر بتوانید این معیارها را در مسائل واقعی پیاده‌سازی کنید و به طور مؤثرتری مدل خود را مورد ارزیابی قرار دهید.

معیارهایی که در این فصل معرفی می‌شوند، هر کدام به نوع خاصی از عملکرد مدل توجه دارند و می‌توانند در شرایط مختلفی مفید واقع شوند. برای مثال، برخی از این معیارها بر توانایی مدل در تشخیص تمام برچسب‌های مرتبط تمرکز دارند، در حالی که برخی دیگر به دقت پیش‌بینی‌های مدل اهمیت می‌دهند. با استفاده از این معیارها، می‌توان مدلی را که بهترین تعادل را بین دقت و بازخوانی برقرار می‌کند، شناسایی کرد.

در ادامه این فصل، با جزئیات بیشتری با این معیارها آشنا می‌شویم و یاد می‌گیریم که چگونه از آنها برای ارزیابی مدل‌های دسته‌بندی چندبرچسبی استفاده کنیم. این دانش به شما کمک خواهد کرد تا به عنوان یک محقق یا مهندس یادگیری عمیق، بتوانید مدل‌های کارآمدتر و مؤثرتری بسازید.

۱-۵ معیارهای ارزیابی

در دسته‌بندی چندبرچسبی، معیارهای ارزیابی مختلفی برای سنجش عملکرد مدل وجود دارد. در ادامه به برخی از این معیارها اشاره شده است.

در معیارهای زیر N تعداد نمونه‌ها، Y_i مجموعه برچسب‌های واقعی و \hat{Y}_i مجموعه برچسب‌های پیش‌بینی شده برای نمونه i است:

۱-۱-۵ دقت

دقت در دسته‌بندی چندبرچسبی به صورت زیر تعریف می‌شود: [۱۹]

$$\text{دقت} = \frac{1}{N} \sum_{i=1}^N \frac{|Y_i \cap \hat{Y}_i|}{|Y_i \cup \hat{Y}_i|} \quad (۱.۱-۵)$$

۲-۱-۵ دقت نمونه^۱

دقت نمونه نشان می‌دهد که از میان برچسب‌های پیش‌بینی شده، چه نسبتی از آنها واقعاً صحیح بوده‌اند. این معیار برای ارزیابی دقت پیش‌بینی‌های مدل مفید است. [۱۹]

$$\text{دقت نمونه} = \frac{1}{N} \sum_{i=1}^N \frac{|Y_i \cap \hat{Y}_i|}{|\hat{Y}_i|} \quad (۲.۱-۵)$$

۳-۱-۵ بازخوانی نمونه^۲

بازخوانی نمونه نشان می‌دهد که از میان برچسب‌های واقعی، چه نسبتی از آنها توسط مدل پیش‌بینی شده‌اند. این معیار برای ارزیابی توانایی مدل در شناسایی تمام برچسب‌های مرتبط مهم است. [۱۹]

^۱Example-Based Precision

^۲Example-Based Recall

$$\text{بازخوانی نمونه} = \frac{1}{N} \sum_{i=1}^N \frac{|Y_i \cap \hat{Y}_i|}{|Y_i|} \quad (3.1-5)$$

۴-۱-۵ امتیاز F1 نمونه^۳

امتیاز F1 نمونه میانگین هارمونیک^۴ دقت و بازخوانی نمونه است. این معیار تعادلی بین دقت و بازخوانی ایجاد می‌کند و برای ارزیابی کلی عملکرد مدل مفید است. [۱۹]

$$\text{امتیاز F1 نمونه} = \frac{1}{N} \sum_{i=1}^N \frac{2 \times |Y_i \cap \hat{Y}_i|}{|Y_i| + |\hat{Y}_i|} \quad (4.1-5)$$

۵-۱-۵ دقت خرد^۵

دقت خرد، دقت کلی مدل را در تمام نمونه‌ها و برچسب‌ها محاسبه می‌کند. این معیار برای ارزیابی عملکرد کلی مدل در تمام کلاس‌ها مفید است. [۱۹]

$$\text{دقت خرد} = \frac{\sum_{i=1}^N |Y_i \cap \hat{Y}_i|}{\sum_{i=1}^N |\hat{Y}_i|} \quad (5.1-5)$$

۶-۱-۵ بازخوانی خرد^۶

بازخوانی خرد، بازخوانی کلی مدل را در تمام نمونه‌ها و برچسب‌ها محاسبه می‌کند. این معیار نشان می‌دهد که مدل تا چه حد توانسته است تمام برچسب‌های مرتبط را در کل مجموعه داده شناسایی

³Example-Based F1 Score

⁴Harmonic

⁵Micro-Averaged Precision

⁶Micro-Averaged Recall

کند. [۱۹]

$$\text{بازخوانی خرد} = \frac{\sum_{i=1}^N |Y_i \cap \hat{Y}_i|}{\sum_{i=1}^N |Y_i|} \quad (۶.۱-۵)$$

۷-۱-۵ امتیاز F1 خرد^۷


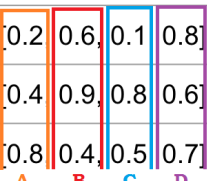


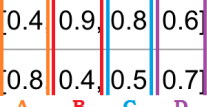




امتیاز F1 خرد، میانگین هارمونیک دقت خرد و بازخوانی خرد است. این معیار یک ارزیابی متعادل از عملکرد کلی مدل در تمام کلاس‌ها ارائه می‌دهد و برای مقایسه مدل‌های مختلف مفید است. [۱۹]

$$\text{امتیاز F1 خرد} = \frac{\text{بازخوانی خرد} \times \text{دقت خرد} \times 2}{\text{بازخوانی خرد} + \text{دقت خرد}} \quad (۷.۱-۵)$$

این معیارها برای سنجش عملکرد مدل در دسته‌بندی چندبرچسبی و شناسایی نقاط قوت و ضعف آن موثر اند.

۲-۵ نحوه ارزیابی مدل

برای ساده تر کردن موضوع یه مسئله چهار کلاسه را در نظر بگیرید، خروجی مدل به صورت یک آرایه چهار تایی می‌باشد.

	Actual Labels	predicted scores (after sigmoid)	Predictions for threshold = 0.5
sample 1			
sample 2			
sample 3			

شکل ۵-۱: نمونه خروجی مدل و برچسب‌های داده

⁷Micro-Averaged F1 Score

با توجه به شکل ۵-۱، لایه انتهایی مدل دارای ۴ نورون می‌باشد. خروجی هر نورون به معنای امتیازی هست که مدل به آن کلاس مدنظر می‌دهد. خروجی‌ها ابتدا وارد تابع فعال‌ساز sigmoid می‌شوند تا به احتمالی بین ۰ تا ۱ تبدیل شوند. در آخر با استفاده از حد آستانه‌ای مانند ۰/۵ احتمال تبدیل به ۰ و ۱ می‌شوند. حال می‌توان برای هر کلاس مدنظر تعداد مثبت صحیح^۸، منفی صحیح^۹، مثبت غلط^{۱۰} و منفی غلط^{۱۱} را شمارش کرد و با استفاده از روابط ۵-۱۰ تا ۵-۷ مدل را ارزیابی کرد. [۱۵]

۳-۵ نتایج ارزیابی

نتایج ارزیابی تشخیص بوی کد بر روی مدل‌های مختلف در جدول ۵-۱ قابل مشاهده است که این مدل‌ها به اندازه یک دوره بر روی ۱۰۰۰۰ داده و بیشینه طول ورودی ۳۰۰۰ توکن و با استفاده از AdamW به عنوان بهینه‌گر و از binary cross-entropy به عنوان تابع ضرر آموزش دیده‌اند.

Model	Precision	Recall	Accuracy	F1
LLaMA 3.1-8B	88.78%	61.39%	85.95%	72.59%
gemma 2-9B	89.03%	56.67%	84.15%	69.25%
LLaMA 3-8B	83.57%	52.08%	83.25%	64.17%
LLaMA 2-7B	84.43%	50.74%	82.15%	63.38%
mistral 7B	86.08%	50.00%	82.22%	63.26%
phi 3.5 mini 3.8B	89.01%	49.34%	80.67%	63.49%
smoLM 2B	85.85%	47.62%	82.93%	61.26%
GPT2-large	82.17%	47.51%	82.95%	60.21%
Random-Forest [14]	90%	30%	39%	45%
ASTNN [14]	76%	52%	45%	62%
HMML- GCN [14]	86%	65%	59%	74%
HMML- LSTM [14]	75%	78%	67%	75%
HMML- Control and data flow edges [14]	71%	72%	68%	71%

جدول ۵-۱: نتایج ارزیابی مدل‌های مختلف با روش شبکه عصبی چندلایه

⁸True Positive

⁹False Positive

¹⁰True Negative

¹¹False Negative

Model	Precision	Recall	Accuracy	F1
LLaMA 3.1-8B	33.73%	49.56%	66.60%	40.14%
gemma 2-9B	30.70%	53.09%	67.65%	38.90%
LLaMA 3-8B	29.29%	56.86%	63.20%	38.67%

جدول ۵-۲: نتایج ارزیابی مدل های مختلف با روش مهندسی درخواست

در این ارزیابی عملکرد ۸ مدل زبانی وسیع با ۵ روش در تشخیص بوی کد مقایسه شده است. نتایج نشان می دهد که مدل های زبانی وسیع به طور کلی عملکرد بهتری نسبت به روش های قبلی دارند. این برتری عمدتاً به دلیل دقت بالاتر، انعطاف پذیری بیشتر، و توانایی درک عمیق تر از متن توسط مدل های زبانی وسیع است.

مدل های زبانی وسیع به دلیل آموزش بر روی داده های وسیع و استفاده از معماری های پیچیده، قادر به تشخیص دقیق تر و جامع تر بوی کد هستند. این مدل ها نه تنها در معیار دقت عملکرد برجسته ای دارند، بلکه توانایی بالایی در تعمیم به داده های جدید و متنوع نیز از خود نشان می دهند. در مقایسه با روش های سنتی که ممکن است در یک یا دو معیار خاص عملکرد خوبی داشته باشند، مدل های زبانی وسیع تعادلی مناسب در تمامی معیارها از خود نشان می دهند.

به طور کلی، استفاده از مدل های زبانی وسیع به دلیل دقت بالاتر، انعطاف پذیری بیشتر، و توانایی درک عمیق تر از نشانه های بوی کد، گزینه ای مناسب تر و موثرتر نسبت به روش های سنتی است. این مدل ها می توانند به طور قابل توجهی به بهبود کیفیت کد و کاهش هزینه های نگهداری در پروژه های نرم افزاری کمک کنند.

فصل ششم

نتیجه گیری و کارهای آینده

۶. نتیجه گیری و کارهای آینده

۱-۶ نتیجه گیری

در این پایان نامه، به بررسی و توسعه یک مدل هوشمند برای تشخیص بوی کد با استفاده از مدل های زبانی وسیع پرداخته شد. بوی کد به عنوان یک مفهوم کلیدی در مهندسی نرم افزار، نشان دهنده مشکلات پنهانی در کدهای برنامه نویسی است که می تواند منجر به کاهش کیفیت، افزایش پیچیدگی در نگهداری و توسعه نرم افزار، و ایجاد مشکلات عملکردی در آینده شود. هدف اصلی این پژوهش ارائه روشی نوین برای شناسایی این مشکلات با بهره گیری از توانایی های مدل های زبانی وسیع و روش های یادگیری عمیق بود.

نتایج این پژوهش نشان داد که استفاده از مدل های زبانی وسیع، به ویژه مدل هایی که با داده های گسترده و متنوع آموزش دیده اند، می تواند به طور چشمگیری دقت و کارایی در تشخیص بوی کد را بهبود بخشد. این مدل ها قادر به تشخیص ۲۸ نوع مختلف از بوی کد بودند و در مقایسه با روش های سنتی، نتایج بهتری را ارائه دادند. همچنین، ارزیابی های انجام شده بر روی مدل پیشنهادی نشان داد که این روش می تواند به عنوان ابزاری مؤثر و کارآمد برای توسعه دهندگان نرم افزار به کار گرفته شود، و به آنها در بهبود کیفیت کد و کاهش هزینه های نگهداری کمک کند.

در عین حال، این پژوهش به چالش های موجود در آموزش و بهینه سازی مدل های زبانی وسیع نیز پرداخت. از جمله این چالش ها می توان به نیاز به منابع محاسباتی بالا و پیچیدگی های مرتبط با تنظیم دقیق مدل ها اشاره کرد. با این وجود، راهکارهای ارائه شده در این پایان نامه، نظیر استفاده از روش های بهینه سازی و فشرده سازی، می تواند به کاهش این چالش ها کمک کرده و امکان استفاده از مدل های زبانی وسیع را در محیط های محدودتر فراهم سازد.

علاوه بر این، نتایج نشان می دهد که مدل های زبانی وسیع به طور کلی در مقایسه با روش های پیشین

عملکرد بهتری دارند. این برتری عمدتاً ناشی از دقت بالاتر، انعطاف پذیری بیشتر، و توانایی این مدل‌ها در درک بهتر و عمیق‌تر از متون است. به دلیل استفاده از داده‌های گسترده و معماری‌های پیچیده در آموزش این مدل‌ها، آن‌ها قادر به تشخیص دقیق‌تر و جامع‌تر بوی کد هستند. این مدل‌ها در تمامی معیارهای ارزیابی عملکرد، تعادلی مناسب از خود نشان می‌دهند و به طور کلی در شناسایی و پیشگیری از مشکلات کد بسیار موثرتر از روش‌های سنتی عمل می‌کنند. به همین دلیل، استفاده از مدل‌های زبانی وسیع می‌تواند به طور قابل توجهی به بهبود کیفیت کد و کاهش هزینه‌های نگهداری در پروژه‌های نرم‌افزاری کمک کند.

در نهایت، این پژوهش نشان داد که ترکیب مدل‌های زبانی وسیع با دانش مهندسی نرم‌افزار می‌تواند به طور قابل توجهی به بهبود فرآیند توسعه نرم‌افزار و کاهش خطاها و مشکلات کد منجر شود. امید است که نتایج این تحقیق بتواند زمینه‌ساز تحقیقات و توسعه‌های بیشتر در این حوزه گردد و به ارتقاء کیفیت و کارایی نرم‌افزارها کمک کند. همچنین، به پژوهشگران و توسعه‌دهندگان توصیه می‌شود که با ادامه تحقیقات در این زمینه و بررسی روش‌های نوین، به بهبود و تکامل ابزارهای تشخیص بوی کد بپردازند تا بتوانند نرم‌افزارهایی پایدارتر، قابل نگهداری‌تر و با کیفیت‌تر ارائه دهند.

۲-۶ کارهای آینده

در سال‌های اخیر، ادغام روش‌های یادگیری ماشین در حوزه مهندسی نرم‌افزار پیشرفت‌های قابل توجهی داشته است، به ویژه در ارزیابی کیفیت کد. یکی از حوزه‌های مهم تمرکز، تشخیص بوی بد کد است. نشانه‌هایی از مشکلات بالقوه در کد که ممکن است به اشکالات یا دشواری‌های نگهداری منجر شوند. اگرچه مدل‌های کنونی نویدبخش هستند، اما هنوز زمینه‌های زیادی برای بهبود وجود دارد. در ادامه این پروژه به سه حوزه کلیدی زیر اشاره کرد:

۱-۲-۶ گسترش مجموعه داده‌ها

یک عنصر اساسی در هر پروژه یادگیری ماشین، مجموعه داده است. تنوع و حجم داده‌ها به طور مستقیم بر توانایی مدل در تعمیم به سناریوهای مختلف تأثیر می‌گذارد. مجموعه داده‌های فعلی برای تشخیص بوی بد کد، هرچند مؤثر هستند، اما اغلب از گستردگی لازم برای پوشش کامل تنوع

سبک‌های کدنویسی، زبان‌ها و محیط‌های مختلف برخوردار نیستند. گسترش مجموعه داده‌ها به منظور شامل کردن طیف وسیع‌تری از زبان‌های برنامه‌نویسی، چارچوب‌های کدنویسی بسیار حیاتی خواهد بود. این گسترش می‌تواند شامل جمع‌آوری پروژه‌های متن‌باز^۱ بیشتر، ادغام کد از حوزه‌های مختلف و حتی تولید مثال‌های مصنوعی از بوی بد کد باشد. با تنوع‌بخشی به مجموعه داده، مدل می‌تواند برای شناسایی طیف گسترده‌تری از بوهای بد آموزش ببیند و در نتیجه دقت و استحکام خود را در زمینه‌های مختلف بهبود بخشد.

۲-۲-۶ بهینه‌سازی مدل

پس از گسترش مجموعه داده، گام بعدی بهینه‌سازی خود مدل است. اگرچه مدل‌های کنونی در تشخیص بوی بد کد عملکرد مناسبی دارند، ولی همیشه فضایی برای بهبود وجود دارد. کارهای آینده می‌توانند شامل بررسی معماری‌های جایگزین یادگیری ماشین، مانند مدل‌های مبتنی بر مبدل یا شبکه‌های عصبی گرافی باشد که ممکن است با توجه به ساختار کد، عملکرد بهتری داشته باشند. علاوه بر این، روش‌های تنظیم دقیق مانند یادگیری انتقالی، که در آن یک مدل پیش‌آموزش‌دیده بر روی یک مجموعه داده بزرگ و عمومی، به صورت خاص‌تر بر روی یک مجموعه داده کوچک‌تر آموزش داده می‌شود، می‌تواند مفید باشد. تنظیم ابر متغیرها^۲ و ترکیب مدل‌های مختلف نیز ممکن است بهبود عملکرد را به همراه داشته باشد. هدف از این بهینه‌سازی‌ها افزایش هر دو معیار دقت و یادآوری تشخیص بوی بد کد است که کاهش مثبت‌های کاذب و منفی‌های کاذب برای پذیرش عملی بسیار مهم است.

۳-۲-۶ تشخیص در زمان واقعی

هدف نهایی ابزارهای تشخیص بوی بد کد، یکپارچه‌سازی بدون مشکل در جریان کاری توسعه‌دهنده است، به نحوی که بازخورد فوری ارائه دهند و به حفظ کیفیت بالای کد از همان ابتدا کمک کنند. توسعه یک سیستم تشخیص در زمان واقعی، مانند یک افزونه برای ویرایشگرهای کد محبوب مانند Visual Studio Code، IntelliJ IDEA یا Eclipse، یک مسیر هیجان‌انگیز برای کارهای آینده است. این

^۱ Open Source

^۲ hyper-parameter

افزونه می‌تواند کد را در حین نوشتن تحلیل کند، بوی بد کد احتمالی را بلافاصله پرچم‌گذاری کند و پیشنهاداتی برای بهبود ارائه دهد. دستیابی به عملکرد در زمان واقعی نیازمند بهینه‌سازی مدل برای سرعت بدون از دست دادن دقت است و همچنین اطمینان از اینکه افزونه بهینه باشد. چنین ابزاری نه تنها بهره‌وری توسعه‌دهنده را افزایش می‌دهد، بلکه با شناسایی مشکلات در اوایل چرخه توسعه، به نگهداری طولانی‌مدت پروژه‌های نرم‌افزاری کمک می‌کند.

آینده تشخیص بوی بد کد در گسترش مجموعه داده‌ها، بهینه‌سازی مدل‌ها و قابلیت‌های تشخیص در زمان واقعی نهفته است. هر یک از این حوزه‌ها چالش‌های منحصر به فردی را به همراه دارد، اما همچنین توانایی قابل توجهی برای بهبود نحوه تشخیص و رسیدگی به مسائل کیفیت کد ارائه می‌دهد. با تمرکز بر این جنبه‌ها، می‌توانیم به توسعه ابزارهایی نزدیک شویم که نه تنها دقیق و کارآمد هستند، بلکه به طور یکپارچه در فرآیند توسعه نرم‌افزار ادغام می‌شوند و در نهایت به کدهایی با کیفیت بالاتر و قابل نگهداری‌تر منجر می‌شوند.

واژه‌نامه فارسی به انگلیسی

Code Smell	بوی کد
Class	کلاس
Method	تابع
Module	ماژول
Rule Based System	سیستم قانون محور
Technique	روش
Abstract Syntax Tree	درخت نحو انتزاعی
Syntax	نحو
Compiler	کامپایلر
Context Free Grammar	گرامر مستقل از متن
Semantic Analysis	تجزیه و تحلیل معنایی
Information Representation	نمایش میانی
Recursive Neural Network	شبکه عصبی بازگشتی
Neuron	نورون
Loss Function	تابع ضرر
Gradient	مشتق
Vanishing Gradient	محوشدگی مشتق
Exploding Gradient	انفجار مشتق
Long Short Term Memory	مدل حافظه طولانی کوتاه مدت
Attention	سازوکار توجه
Transformer	مبدل
Natural Language Processing	پردازش زبان های طبیعی
Weight	وزن
Self-Attention	مکانیسم خودتوجهی
Encoder	کدگذار
Decoder	کدگشا

Machine Learning	یادگیری ماشین
Threshold	حد آستانه
Metric	معیار
Deep Learning	یادگیری عمیق
Convolutional Network	شبکه‌های پیچشی
Singel-Label Classification	دسته بندی تک برچسبی
Multi-Label Classification	دسته بندی چند برچسبی
Tokenize	واژه‌بندی
Multi Layer Perceptron	شبکه عصبی چندلایه
Entropy	آنتروپی
Quantize	فشرده‌سازی
Accuracy	دقت
Example-Based Precision	دقت نمونه
Example-Based Recall	بازخوانی نمونه
Example-Based F1 Score	امتیاز F1 نمونه
Micro-Averaged Precision	دقت خرد
Micro-Averaged Recall	بازخوانی خرد
Micro-Averaged F1 Score	امتیاز F1 خرد
Activation Function	تابع فعال‌ساز
True Positive	مثبت صحیح
False Positive	منفی صحیح
True Negative	مثبت غلط
False Negative	منفی غلط
Open Source	متن‌باز
Hyper-Parameter	ابر متغیر

پیوست ۱ - آماده سازی داده ها

تمامی کد های پروژه و فایل های Latex در لینک زیر موجود است:

<https://github.com/MSPoulaei/code-smell-detection-with-LLM>

لینک مدل آموزش داده شده:

https://huggingface.co/mspoulaei/Code_Smell_Detection_SmoLLM

ثابت ها

```
CODE_OUTPUT_PATH="output_code"
SOURCE_CODE_NAME="source<counter>.java"
code_dir = "code"
label_dir = "label"
prefix = 'dataset'
counter_code=0
```

تابع ترکیب کردن کد های پروژه جاوا

```
def combine_and_save_source_code(folder_dir):
    global counter_code
    combined_contents = []
    for dir, _, files in os.walk(folder_dir):
        for file in files:
            file_path = os.path.join(dir, file)
            with open(file_path, 'r') as f:
                content = f.read()
                combined_contents.append(content)
            counter_code+=1
    source_code_name_with_counter = SOURCE_CODE_NAME.replace("<counter>",str(counter_code))
    output_path = os.path.join(CODE_OUTPUT_PATH,source_code_name_with_counter)
    with open(output_path, 'w') as f:
        f.write("\n\n".join(combined_contents))
    return source_code_name_with_counter
```

گردهم آوردن لیبل ها به صورت یکجا برای آموزش

```
dataset = []
for dir, folders, files in os.walk(os.path.join(prefix,label_dir)):
    name = dir.split("\\")[-1]
    for file in files:
        last_name = None
        if file == 'designCodeSmells.csv' or file=='implementationCodeSmells.csv':
            file_path = os.path.join(dir, file)
            df = pd.read_csv(file_path)
            i=0
            while i < len(df):
                row=df.iloc[i]
                codesmells=[row[-1]]
                last_name=row[-2]
                for j in range(i+1,len(df)):
                    row2=df.iloc[j]
                    if row2[-2]!=last_name:
                        i=j-1
                        break
                codesmells.append(row2[-1])
                output_path = combine_and_save_source_code(
```

```
        os.path.join(prefix,code_dir,str(row[0]),
str(row[0]),
str(row[1]),
str(row[2])))
dataset.append({'file_path':output_path,
'codesmells':"".join(set(codesmells))})
i+=1
dataset = pd.DataFrame(dataset)
dataset.to_csv('dataset.csv', index=False)
```

پیوست ۲ - آموزش مدل با روش QLORA

بارگذاری داده ها و ایجاد لیبل ها بصورت one hot

```
df = pd.read_csv('prepared_dataset/dataset.csv', header=None,
names=['file_path', 'codesmells'])
df.rename(columns={'codesmells': 'labels'}, inplace=True)
df['labels'] = df['labels'].apply(lambda x: x.split(','))

all_labels = set(label for sublist in df['labels'] for label in sublist)
label_to_idx = {label: idx for idx, label in enumerate(all_labels)}
id2label = {i: label for label, i in label_to_idx.items()}

def encode_labels(labels):
    encoded = [0] * len(label_to_idx)
    for label in labels:
        encoded[label_to_idx[label]] = 1
    return encoded
df['encoded_labels'] = df['labels'].apply(encode_labels)
```

جدا کردن داده های آزمون و آموزش و ساخت دیتاست

```
train_df, test_df = train_test_split(df, test_size=0.1, random_state=42, shuffle=True)
class CodeDataset(Dataset):
    def __init__(self, dataframe, tokenizer):
        self.dataframe = dataframe
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        code_path = os.path.join("prepared_dataset", "output_code",
self.dataframe.iloc[idx]['file_path'])
        with open(code_path, 'r') as file:
            code = file.read()
        labels = torch.tensor(self.dataframe.iloc[idx]['encoded_labels'],
dtype=torch.float).to(device)
        inputs = self.tokenizer(code, return_tensors='pt',
truncation=True, padding='max_length', max_length = MAX_LEN,
add_special_tokens = True).to(device)

        #squeeze inputs:
        inputs = {key: val.squeeze() for key, val in inputs.items()}
        return {**inputs, 'labels': labels}
train_dataset = CodeDataset(train_df, tokenizer)
test_dataset = CodeDataset(test_df, tokenizer)
```

بارگذاری توکنایزر و مدل و فشرده کردن مدل

```
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
)
model = AutoModelForSequenceClassification.from_pretrained(
MODEL_NAME, num_labels=len(all_labels),
quantization_config=quantization_config,
problem_type="multi_label_classification",
```

```
low_cpu_mem_usage=True)
```

استفاده از حالت مصرف بهینه رم برای پارامترها با استفاده از PEFT

```
model.train() # model in training mode (dropout modules are activated)

# enable gradient check pointing
model.gradient_checkpointing_enable()

# enable quantized training
model = prepare_model_for_kbit_training(model)
# LoRA config
config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["q_proj"],
    lora_dropout=0.1,
    bias="none",
    task_type="SEQ_CLS",
)
# LoRA trainable version of model
model = get_peft_model(model, config)

# trainable parameter count
model.print_trainable_parameters()
```

آموزش دادن مدل

```
training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    warmup_steps=500,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    num_train_epochs=NUM_EPOCHS,
    weight_decay=0.01,
    gradient_accumulation_steps=4,
    # warmup_steps=2,
    fp16=True,
    optim="paged_adamw_8bit",
)
trainer = Trainer(
    model=model,
    tokenizer=tokenizer,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics,
)
model.config.use_cache = False # silence the warnings. Please re-enable for inference!
trainer.train()
model.config.use_cache = True
```

پیوست ۳ - نحوه‌ی ارزیابی مدل

تابع بدست آوردن معیارهای ارزیابی از روی خروجی مدل و برچسب‌های داده

```
def compute_metrics(p):  
    # Convert predictions to sigmoid and then to binary  
    preds = torch.sigmoid(torch.tensor(p.predictions))  
    preds = (preds > 0.5).int()  
    labels = torch.tensor(p.label_ids)  
  
    # Accuracy  
    accuracy = (preds == labels).float().mean().item()  
  
    # Precision, Recall, F1 Score  
    true_positive = (preds * labels).sum(dim=0).float()  
    predicted_positive = preds.sum(dim=0).float()  
    actual_positive = labels.sum(dim=0).float()  
  
    # Adding a small epsilon to avoid division by zero  
    epsilon = 1e-7  
  
    precision = (true_positive / (predicted_positive + epsilon)).mean().item()  
    recall = (true_positive / (actual_positive + epsilon)).mean().item()  
    f1_score = (2 * precision * recall / (precision + recall + epsilon))  
  
    return {  
        "accuracy": accuracy,  
        "precision": precision,  
        "recall": recall,  
        "f1_score": f1_score  
    }
```

ارزیابی مدل با استفاده از تابع ارزیابی تعریف شده

```
results = trainer.evaluate()  
print(results)
```

پیوست ۴ - تشخیص بوی کد با استفاده از مهندسی پرامپت

بارگزاری مدل زبانی برای کار تولید متن بصورت فشرده

```
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
)
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=quantization_config,
    device_map="auto",
)
```

تعریف پایپلاین تولید متن

```
gen = pipeline('text-generation', model=model, tokenizer=tokenizer)
generation_args = {
    "max_new_tokens": 100,
    "return_full_text": False,
    "temperature": 0.0,
    "do_sample": False,
}
```

تعریف پرامپت سیستم

```
system_prompt=f"""
you are an agent should output the correct code smells for multi-classification problem.
these are the classes we have:
{label_to_idx}"""
```

تولید پرامپت کاربر بصورت one-shot

```
def generate_user_prompt(code, oneshot_code, oneshot_labels):
    user_prompt = f"""
    output an array with the correct classes as 1 comma separated for the code below:
    code1:
    """
    {oneshot_code}
    """
    labels1:
    {oneshot_labels}
    code2:
    """
    {code}
    """
    labels2:
    """
    return user_prompt
```

تولید پرامپت ها با استفاده از داده ها

```

prompts=[]
for i,entry in tqdm(enumerate(dataset)):
    last_entry=dataset[i-1]
    user_prompt=generate_user_prompt(entry['code'],last_entry['code'],dataset[i-1]["labels"])
    messages=[
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]
    prompts.append(messages)

```

ورودی دادن پرامپت ها به پایپلاین تولید متن

```

prompts_to_gen=[]
count=0
while count<10:
    for i in range(len(prompts)):
        if len(prompts[i][0]["content"])+len(prompts[i][1]["content"])<2000:
            prompts_to_gen.append(prompts[i])
            count+=1
inference_results=gen(prompts_to_gen, **generation_args)

```

تابع تجزیه متن خروجی مدل بصورت آرایه برچسب ها

```

def parse_generated_text(generated_text):
    # Extract class names from the generated text
    lines = generated_text.split('\n')
    class_names = [line.split('.')[1].strip('\"') for line in lines if '.' in line]
    # Convert class names to indices
    class_indices = [label_to_idx[name] for name in class_names if name in label_to_idx]
    return class_indices

```

تابع ارزیابی مدل

```

def compute_individual_metrics(preds, labels):
    # Accuracy
    preds = preds.cpu()
    labels = labels.cpu()
    accuracy = (preds == labels).float().mean().item()

    # Precision, Recall, F1 Score
    true_positive = (preds * labels).sum().float()
    predicted_positive = preds.sum().float()
    actual_positive = labels.sum().float()

    # Adding a small epsilon to avoid division by zero
    epsilon = 1e-7

    precision = (true_positive / (predicted_positive + epsilon)).item()
    recall = (true_positive / (actual_positive + epsilon)).item()
    f1_score = (2 * precision * recall / (precision + recall + epsilon))

    return {
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1_score
    }

```

تابع بدست آوردن ارزیابی مدل به ازای تمام نمونه ها

```
def compute_overall_metrics(inference_results, ground_truths):
    # Initialize accumulators for metrics
    total_accuracy = 0
    total_precision = 0
    total_recall = 0
    total_f1_score = 0

    for i, result in enumerate(inference_results):
        # Parse the generated text to get predicted class indices
        generated_text = result[0]['generated_text']
        predicted_indices = parse_generated_text(generated_text)

        # Create a binary tensor for predictions
        preds = torch.zeros(len(label_to_idx), dtype=torch.int)
        preds[predicted_indices] = 1

        # Ground truth labels
        labels = torch.tensor(ground_truths[i]["labels"], dtype=torch.int)

        # Compute metrics for this instance
        metrics = compute_individual_metrics(preds, labels)

        # Accumulate metrics
        total_accuracy += metrics["accuracy"]
        total_precision += metrics["precision"]
        total_recall += metrics["recall"]
        total_f1_score += metrics["f1_score"]

    # Average metrics over all instances
    num_instances = len(inference_results)
    overall_metrics = {
        "accuracy": total_accuracy / num_instances,
        "precision": total_precision / num_instances,
        "recall": total_recall / num_instances,
        "f1_score": total_f1_score / num_instances
    }

    return overall_metrics
```

ارزیابی خروجی های مدل

```
metrics = compute_overall_metrics(inference_results, dataset)
print(metrics)
```


- [1] F. AL-Jibory, "Code smells in software: Review," *International Journal of Nonlinear Analysis and Applications*, vol. 14, no. 1, pp. 1339–1345, 2023. [Online]. Available: https://ijnaa.semnan.ac.ir/article_7029.html
- [2] M. Ilyas, S. Adnan, and W. Ahmad, "Code smell detection and refactoring using astvisitor," 04 2020.
- [3] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167278919305974>
- [4] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia CIRP*, vol. 99, pp. 650–655, 2021, 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2212827121003796>
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.
- [7] M. A. A. Hilmi, A. Puspaningrum, Darsih, D. O. Siahaan, H. S. Samosir, and A. S. Rahma, "Research trends, detection methods, practices, and challenges in code smell: Slr," *IEEE Access*, vol. 11, pp. 129 536–129 551, 2023.
- [8] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, vol. 108, pp. 115–138, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584918302623>
- [9] A. Kaur, S. Jain, and S. Goel, "A support vector machine based approach for code smell detection," in *2017 International Conference on Machine Learning and Data Science (MLDS)*. IEEE, 2017, pp. 9–14.
- [10] H. Liu, J. Jin, Z. Xu, Y. Zou, Y. Bu, and L. Zhang, "Deep learning based code smell detection," *IEEE transactions on Software Engineering*, vol. 47, no. 9, pp. 1811–1837, 2019.
- [11] A. K. Das, S. Yadav, and S. Dhal, "Detecting code smells using deep learning," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. IEEE, 2019, pp. 2081–2086.

- [12] T. Guggulothu, "Code smell detection using multilabel classification approach," pp. 8–9, 2019. [Online]. Available: <https://arxiv.org/abs/1902.03222>
- [13] T. Guggulothu and S. A. Moiz, "Code smell detection using multi-label classification approach," *Software Quality Journal*, vol. 28, pp. 1063–1086, 2020.
- [14] Y. Li and X. Zhang, "Multi-label code smell detection with hybrid model based on deep learning," 07 2022, pp. 42–47.
- [15] J. Bogatinovski, L. Todorovski, S. Džeroski, and D. Kocev, "Comprehensive comparative study of multi-label classification methods," *Expert Systems with Applications*, vol. 203, p. 117215, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417422005991>
- [16] Z. Y, D. CH, L. H, and G. CY, "Code smell detection approach based on pre-training model and multi-level information," *Journal of Software*, vol. 33, no. 5, p. 1551, 05 2022.
- [17] M. R. J, K. VM, H. Warriar, and Y. Gupta, "Fine tuning llm for enterprise: Practical guidelines and recommendations," 2024. [Online]. Available: <https://arxiv.org/abs/2404.10779>
- [18] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.
- [19] M. Hossin and S. M.N, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining and Knowledge Management Process*, vol. 5, pp. 01–11, 03 2015.

Abstract:

This research proposes a model for detecting code smells using large language models. Code smells refer to concepts and features in programming code that may indicate deeper issues in software design and implementation. These issues can lead to reduced code quality and increased complexity in maintenance and development. The proposed method leverages large language models trained on labeled datasets, capable of detecting 28 different types of code smells. The model is developed using advanced deep learning techniques and architectures, such as Transformers. Evaluation results show that the proposed model significantly improves the accuracy of code smell detection and can serve as an effective tool for software developers. The research also addresses the challenges in training and optimizing large language models and provides solutions to enhance the model's performance.

Keywords: Code Smells, Large Language Models, Deep Learning, Automatic Detection, Transformer, Model Optimization



Iran University of Science and Technology
Computer Engineering Department

Code Smell Detection with Suggested Large Language Models

Bachelor of Computer Engineering Final Project

By:

Mohammad Sadegh Poulaei Moziraji

Sayin Ala

Supervisor:

Dr. Saeid Parsa

August 2024