بسم الله الرحمن الرحيم

دانشکده فنی و مهندسی
بخش مهندسی کامپیوتر

پایان نامه تحصیلی برای دریافت درجه کارشناسی ارشد

رشته مهندسی کامپیوتر گرایش هوش مصنوعی

# تشخیص قدرتمند بدافزارهای اندروید با استفاده از شبکه‌های عصبی ترنسفورمر

**مؤلف:**
علیرضا ایرانمنش

**استاد راهنما:**
دکتر حمید میروزیری

اردیبهشت ۱۴۰۴

به نام خدا

منشور اخلاق پژوهش

با استعانت از خدای سبحان و اعتقاد راسخ به این که عالم محضر خداست و او همواره ناظر بر اعمال ماست و به منظور انجام شایسته‌ی پژوهش‌های اصیل، تولید دانش جدید و بسازی زندگانی بشر، ما دانشجویان و اعضای هیات علمی دانشگاه‌ها و پژوهشگاه‌های کشور:

- ☐ تمام تلاش خود را برای کشف حقیقت و فقط حقیقت به کار خواهیم بست و از هرگونه جعل و تحریف در فعالیت‌های علمی پرهیز می‌کنیم.
- ☐ حقوق پژوهشگران، پژوهیدگان (انسان، حیوان، نبات و اشیاء)، سازمان‌ها و سایر صاحبان حق را به رسمیت می‌شناسیم و در حفظ آن می‌کوشیم.
- ☐ به مالکیت مادی و معنوی آثار پژوهشی ارج می‌نهیم، برای انجام پژوهش اصیل اهتمام ورزیده و از سرقت علمی و ارجاع نامناسب اجتناب می‌کنیم.
- ☐ ضمن پایبندی به انصاف و اجتناب از هرگونه تبعیض و تعصب در کلیه فعالیت‌های پژوهشی، رویکردی نقادانه اتخاذ خواهیم کرد.
- ☐ ضمن امانت‌داری، از منابع و امکانات اقتصادی، انسانی و فنی موجود، استفاده بهره‌ورانه خواهیم کرد.
- ☐ از انتشار غیراخلاقی نتایج پژوهش، نظیر انتشار موازی، هم‌پوشان و چندگانه (مکرر) پرهیز می‌کنیم.
- ☐ اصل محرمانه بودن و رازداری را محور تمام فعالیت‌های پژوهشی خود قرار می‌دهیم.
- ☐ در همه فعالیت‌های پژوهشی به منافع ملی توجه کرده و برای تحقق آن می‌کوشیم.
- ☐ خویش را ملزم به رعایت کلیه بخشنامه‌های علمی رشته خود، قوانین و مقررات، سیاست‌های حرفه‌ای، سازمانی، دولتی و رابطه‌های ملی در همه مراحل پژوهش می‌دانیم.
- ☐ رعایت اصول اخلاق در پژوهش را اقدامی فرهنگی می‌دانیم و به منظور ماندگی این فرهنگ، به ترویج و اشاعه آن در جامعه اهتمام می‌ورزیم.

**تعهدنامه**

اینجانب ناهید عبداللهی کرمانی به شماره دانشجویی ۴۰۱۱۵۶۰۰۵ دانشجوی مقطع کارشناسی ارشد رشته مهندسی کامپیوتر_هوش مصنوعی دانشکده فنی مهندسی دانشگاه شهید باهنر کرمان نویسنده پایان‌نامه با عنوان «خودکارسازی مهندسي پرامپت : تولید متا_پرامپت (پرامپت‌های دستوری) برای مدل های بزرگ زباني جهت حل مسائل پردازش زبان طبيعي» تحت راهنمایی دکتر مهدی افتخاری تأیید می‌کنم که این پایان‌نامه نتیجه پژوهش اینجانب می‌باشد و در عین حال که موضوع آن تکراری نیست، در صورت استفاده از منابع دیگران، نشانی دقیق و مشخصات کامل آن درج شده است. همچنین موارد زیر را نیز تعهد می‌کنم:

۱- برای انتشار تمام یا قسمتی از داده‌ها یا دستاوردهای خود در مجامع و رسانه‌های علمی اعم از همایش‌ها و مجلات داخلی و خارجی به صورت مقاله، کتاب، ثبت اختراع و .... به صورت مکتوب یا غیرمکتوب، با کسب مجوز از دانشگاه شهید باهنر کرمان و استاد(ان) راهنما اقدام نمایم.

۲- از درج اسامی افراد خارج از کمیته پایان‌نامه در جمع نویسندگان مقاله‌های مستخرج از پایان‌نامه، بدون مجوز استاد(ان) راهنما اجتناب نمایم و اسامی افراد کمیته پابان نامه را در جمع نویسندگان مقاله درج نمایم.

۳- از درج نشانی یا وابستگی کاری (affiliation) نویسندگان سازمان‌های دیگر (غیر از دانشگاه شهید باهنر کرمان) در مقاله‌های مستخرج از پایان‌نامه بدون تأیید استاد(دان) راهنما اجتناب نمایم [۱].

۴- کلیه ضوابط و اصول اخلاقی مربوط به استفاده از موجودات زنده یا بافتهای آنها را برای انجام پایان‌نامه رعایت نمایم. ۵- در صورت اثبات تخلف (در هر زمان) مدرک تحصیلی صادر شده توسط دانشگاه شهید باهنر کرمان از درجه اعتبار ساقط و اینجانب هیچ‌گونه ادعایی نخواهم داشت.

کلیه حقوق مادی و معنوی این اثر (مقالات مستخرج، برنامه های رایانه ای، نرم افزارها و تجهیزات ساخته شده) مطابق با آیین‌نامه مالکیت فکری، متعلق به دانشگاه شهید باهنر کرمان است و بدون اخذ اجازه کتبی از دانشگاه قابل واگذاری به شخص ثالث نیست. همچنین استفاده از اطلاعات و نتایج این پایان‌نامه بدون ذکر مرجع مجاز نمی باشد. چنانچه مبادرت به عملی خلاف این تعهدنامه

---

[۱] تنها آدرس مورد قبول برای دانشگاه به این صورت می باشد:
Shahid Bahonar University of Kerman، Kerman، Iran.
نام و آدرس واحدهای دانشگاه در تولیدات علمی محققان دانشگاه به تشخیص بخش و دانشکده به شرح زیر می باشد:
Department of Computer، Faculty of Engineering Shahid Bahonar University of Kerman، Kerman، Iran.
آدرس صحیح جهت درج در مقالات و سایر تولیدات علمی فارسی:
گروه (بخش) کامپیوتر، دانشکده فنی مهندسی، دانشگاه شهید باهنر کرمان، کرمان، ایران.

محرز گردد، دانشگاه شهید باهنر کرمان در هر زمان و به هر نحو مقتضی حق هرگونه اقدام قانونی را در استیفای حقوق خود دارد.

تاریخ و امضا:
ناهید عبداللهی کرمانی
فروردین ۱۴۰۴

## تقدیم به:

«این مجموعه را باکمال افتخار و احترام تقدیم می‌کنم به:
روح بلند بنیانگذار دانشگاه، مرحوم **افضلی‌پور** و همسر گرامیشان **بانو فاخره صبا**.
آنان که عاشقانه سوختند تا گرمابخش وجود ما و روشنگر راهمان باشند...
به پاس تعبیر عظیم و انسانی‌شان از کلمه ایثار.
به پاس عاطفه سرشار و گرمای امیدبخش وجودشان که در این سردترین روزگاران بهترین پشتیبان ماست.
به پاس قلب بزرگشان، و به پاس محبت‌های بی دریغشان که هرگز فروکش نمی‌کند»

# تشکر و قدردانی:

با سپاس از خداوند بزرگ که به من توانایی و انگیزه برای پیمودن این مسیر علمی را عطا نمود. این پایان‌نامه حاصل تلاش و کوشش‌های فراوان است و بدون حمایت و راهنمایی‌های ارزشمند افراد بسیاری به ثمر نمی‌نشست.
به مصداق شعر «به یاد کسی که در این راه بود، به یاد کسی که در این راه رفت»، شایسته می‌دانم مراتب سپاس و قدردانی صمیمانه خود را تقدیم نمایم به استاد فرهیخته و فرزانه، جناب آقای دکتر حمید میروزیری، که با دانش و تجربه‌ی خود همواره راهنمای من بودند و با صبر و شکیبایی به سوالات و ابهاماتم پاسخ دادند، صمیمانه تشکر می‌کنم.
همچنین از دوست عزیزم، محمدحسین شبانی، که با حمایت‌های بی‌دریغ و تشویق‌های همیشگی‌اش، انگیزه و انرژی مضاعفی به من بخشید، قدردانی می‌نمایم.
در پایان، از خانواده‌ی عزیزم که با عشق و محبت بی‌پایان خود همواره پشتیبان من بودند و در تمامی مراحل این مسیر پرچالش، همراه و همدل من بودند، بی‌نهایت سپاسگزارم.

## چکیده:

تشخیص بدافزارهای اندرویدی با افزایش روزافزون تهدیدات سایبری، یکی از چالش‌های اصلی در حوزه امنیت اطلاعات به شمار می‌رود. روش‌های سنتی، به‌ویژه آن‌هایی که صرفاً بر تحلیل ویژگی‌های تک‌وجهی تکیه دارند، اغلب با محدودیت‌هایی نظیر ناتوانی در پردازش داده‌های پیچیده چندوجهی و تعمیم‌پذیری ضعیف در برابر تهدیدات جدید مواجه‌اند. این کاستی‌ها، نیاز به توسعه رویکردهای نوین و کارآمد را بیش از پیش آشکار می‌سازد. این پژوهش مدلی چندوجهی با عنوان تبدیل‌گر چندوجهی مبتنی بر جاسازی گراف دینامیک با توجه پویا (MAGNET) توسعه داد که از ترکیب داده‌های جدولی، گراف و ترتیبی، نظیر توالی فراخوانی‌های API، برای شناسایی بدافزارهای اندرویدی بهره برد. هدف اصلی، بهبود دقت و پایداری تشخیص با استفاده از معماری پیشرفته مبتنی بر یادگیری عمیق و ترنسفورمر بود. روش تحقیق شامل بهینه‌سازی هایپرپارامترها با الگوریتم‌های پیشرفته مانند PIRATES و ،Optuna آموزش مدل با مجموعه داده‌ای شامل ۴۶۴۱ نمونه آموزشی و ۱۴۵۱ نمونه آزمایشی، و اعتبارسنجی متقاطع ۵ـتایی شد. ویژگی‌های مورد استفاده شامل ویژگی‌های ایستا مانند مجوزها، فراخوانی‌های ،API مقاصد، و نام‌های مؤلفه و ویژگی‌های پویا مانند فعالیت شبکه و دسترسی به فایل‌ها بود. داده‌ها به صورت بردارهای عددی باینری یا نرمال‌سازی شده بودند. ابعاد ویژگی‌ها پس از پیش‌پردازش به ۴۳۰ ویژگی تنظیم شد. ابزارهای مورد استفاده شامل کتابخانه‌های یادگیری عمیق مانند ،PyTorch تکنیک‌های پیش‌پردازش داده‌ها نظیر استانداردسازی و نرمال‌سازی، و ساختارهای داده‌ای گرافی بودند. مواد اولیه شامل داده‌های واقعی از رفتار اپلیکیشن‌های اندرویدی، شامل ویژگی‌های ایستا و پویا، بود که با دقت آماده‌سازی شدند. نتایج نشان داد که مدل پیشنهادی عملکردی برجسته با دقت بالا، پایداری قابل‌توجه و قابلیت تعمیم‌پذیری خوب ارائه کرد و نسبت به روش‌های پیشین بهبود قابل‌ملاحظه‌ای داشت. این دستاوردها پتانسیل کاربرد مدل در سیستم‌های امنیتی واقعی را برجسته ساخت. پیشنهاد می‌شود تحقیقات آینده بر افزایش حجم داده‌ها، ادغام روش‌های خودنظارتی پیشرفته، آزمایش مدل در محیط‌های متنوع‌تر و بهینه‌سازی زمان اجرای آن متمرکز شوند تا کارایی مدل در سناریوهای پیچیده‌تر و واقعی‌تر ارتقا یابد. همچنین، بررسی تأثیر ترکیب داده‌های جدیدتر و توسعه الگوریتم‌های مقاوم در برابر حملات مخرب می‌تواند مسیرهای نوینی برای تحقیقات بعدی گشوده کند. این پژوهش، گامی مؤثر در راستای ارتقای سیستم‌های تشخیص خودکار بدافزارها برداشت و پایه‌ای محکم برای توسعه راه‌حل‌های امنیتی پیشرفته‌تر فراهم آورد.

**واژگان کلیدی: تشخیص بدافزار، ترنسفومر، یادگیری عمیق، داده‌های چندوجهی، امنیت اندروید، بدافزار اندروید.**

# فهرست مطالب

**عنوان** **صفحه**

# فهرست جداول

**عنوان** **صفحه**

# فهرست تصاویر

**عنوان** **صفحه**

# فهرست الگوریتم‌ها

**عنوان** **صفحه**

bookti- = booktitle title، = title author، = author @inbookID،
editor، = OPTeditor bookauthor، = OPTbookauthor date، = date tle،
editorc، = OPTeditorc editorb، = OPTeditorb editora، = OPTeditora
OPTcom- annotator، = OPTannotator translator، = OPTtranslator
OPT- introduction، = OPTintroduction commentator، = mentator
sub- = OPTsubtitle afterword، = OPTafterword foreword، = foreword
OPT- maintitle، = OPTmaintitle titleaddon، = OPTtitleaddon title،
maintitleaddon، = OPTmaintitleaddon mainsubtitle، = mainsubtitle
booktitlead- = OPTbooktitleaddon booksubtitle، = OPTbooksubtitle
OPTvol- origlanguage، = OPTorigLanguage language، = OPTlanguage don،
OPTvolumes edition، = OPTedition part، = OPTpart volume، = ume
= OPTnote number، = OPTnumber series، = OPTseries volumes، =
OPTisbn location، = OPTlocation publisher، = OPTpublisher note،
OPTaddendum pages، = OPTpages chapter، = OPTchapter isbn، =
= OPTeprint doi، = OPTdoi pubstate، = OPTpubstate addendum، =
eprinttype، = OPTeprinttype eprintclass، = OPTeprintclass eprint،
urldate، = OPTurldate url، = OPTurl

\]mmakefnmark\akefntext[

# فصل اول:
# کلیات پژوهش

## ۱ - ۱ مقدمه و بیان مسئله

در سال‌های اخیر، گسترش تلفن‌های همراه و به‌ویژه سیستم‌عامل اندروید[۱]، موجب افزایش وابستگی کاربران به این ابزارها شده است. این دستگاه‌ها نه تنها در زندگی روزمره، بلکه در حوزه‌های تجاری و نظامی نیز نقش مهمی ایفا می‌کنند. با این حال، محبوبیت و فراگیری اندروید، آن را به هدفی جذاب برای حملات بدافزاری[۲] تبدیل کرده است. عرضه نرم‌افزارهای غیرمعتبر و تهدیداتی مانند ویروس‌ها و بدافزارها، امنیت کاربران را به خطر انداخته است. مطالعات اخیر نشان می‌دهد که بیش از 70 درصد دستگاه‌های هوشمند از سیستم‌عامل اندروید استفاده می‌کنند و این امر باعث شده است که این پلتفرم به هدف اصلی حملات امنیتی تبدیل شود **[AndroidSecurity]**. با وجود پیشرفت‌های قابل توجه در روش‌های تشخیص بدافزار، همچنان چالش‌های جدی در شناسایی بدافزارهای جدید و پیچیده وجود دارد.

در ابتدا، روش‌های سنتی مبتنی بر تحلیل مجوزها[۳] و بازکردن فایل‌ها مورد استفاده قرار می‌گرفتند که به دلیل دقت پایین و ضعف در شناسایی بدافزارهای پیچیده، محدودیت‌هایی داشتند. پژوهش‌های اخیر نشان داده‌اند که روش‌های مبتنی بر یادگیری ماشین[۴] و یادگیری عمیق[۵] می‌توانند عملکرد بهتری در تشخیص بدافزارها داشته باشند **[DeepLearningMalware]**. با این حال، همچنان چالش‌های مهمی در زمینه تفسیرپذیری مدل‌ها[۶] و قابلیت تعمیم‌پذیری[۷] وجود دارد. این چالش‌ها به ویژه در مواجهه با بدافزارهای جدید و ناشناخته (Zero-Day)[۸] بیشتر خود را نشان می‌دهند.

مدل MAGNET[۹] که در این پژوهش معرفی شده است، با بهره‌گیری از معماری ترنسفورمر[۱۰] چندوجهی و ترکیب داده‌های جدولی، گراف و توالی، تلاش می‌کند تا این چالش‌ها را برطرف کند. این مدل با استفاده از مکانیزم‌های توجه پویا[۱۱] و تحلیل همزمان داده‌های مختلف، قادر به تشخیص دقیق‌تر بدافزارها خواهد بود. نتایج نشان می‌دهد که این رویکرد

[1] Android
[2] Malware
[3] Permissions
[4] Machine Learning
[5] Deep Learning
[6] Model Interpretability
[7] Generalization
[8] Zero-Day
[9] MAGNET(Multi-Modal Analysis for Graph and Network Threat Detection)
[10] Transformer
[11] Attention Mechanism

با دقت 97.24% ± 0.5%، معیار F1 = 0.9823 ± 0.002، و معیار AUC = 0.9932 ± 0.003 عملکرد بهتری نسبت به مدل‌های پایه مانند SVM (دقت 90.6%)، CNN (دقت 92.8%) و LSTM (دقت 91.5%) دارد.

### ۱ - ۱ - ۱ روش‌های تشخیص بدافزار

تشخیص بدافزارهای اندرویدی به دو روش کلی پویا[1] و ایستا[2] انجام می‌شود. در روش پویا، رفتار اپلیکیشن در زمان اجرا مانند مصرف باتری، پردازنده یا ترافیک شبکه بررسی می‌شود تا الگوهای غیرعادی شناسایی گردد. این روش به‌تنهایی کافی نیست و ممکن است برخی تهدیدات پنهان را نادیده بگیرد. روش ایستا با تحلیل ساختار و کد اپلیکیشن، مانند بررسی فراخوانی‌های API[3] و مجوزها، اطلاعات ارزشمندی ارائه می‌دهد که می‌تواند در تشخیص دقیق‌تر کمک کند. پژوهش‌های اخیر نشان داده‌اند که ترکیب این دو روش می‌تواند نتایج بهتری در تشخیص بدافزارها ارائه دهد [**AndroidMalwareSurvey**].

### ۱ - ۱ - ۲ مجموعه داده‌های مربوطه

در حوزه تشخیص بدافزار اندروید، مجموعه داده‌های متنوعی برای ارزیابی عملکرد مدل‌ها مورد استفاده قرار گرفته‌اند. از جمله این مجموعه داده‌ها می‌توان به موارد زیر اشاره کرد:

- مجموعه داده‌های Drebin [**Drebin**] و AndroZoo [**AndroZoo**] که شامل نمونه‌های گسترده‌ای از بدافزارها و برنامه‌های سالم اندرویدی هستند.
- مجموعه داده‌های CICMalDroid [**CICMalDroid**] و VirusShare که شامل نمونه‌های جدید و به‌روز از بدافزارها می‌باشند.
- مجموعه داده‌های خصوصی و صنعتی که توسط شرکت‌های امنیتی و مراکز تحقیقاتی گردآوری شده‌اند.

این مجموعه داده‌ها به عنوان شاخص‌های استاندارد، امکان ارزیابی دقیق و جامع عملکرد الگوریتم‌های تشخیص بدافزار را فراهم می‌کنند و نقش مهمی در اثبات قابلیت تعمیم و کارایی روش‌های پیشنهادی دارند.

[1] Dynamic Analysis
[2] Static Analysis
[3] API

## ۱ - ۲ ضرورت تحقیق و اهداف

پلتفرم اندروید به دلیل محبوبیت گسترده و سهم عظیمش از بازار جهانی، به هدف اصلی بدافزارها و حملات امنیتی تبدیل شده است. این سیستم‌عامل، که بیش از 70 درصد دستگاه‌های هوشمند را پشتیبانی می‌کند، به دلیل ساختار باز و دسترسی‌پذیری بالا، با تهدیدات پیشرفته‌ای مواجه است. بدافزارهای اندرویدی، از جمله تروجان‌ها[۴]، جاسوس‌افزارها[۵] و باج‌افزارها[۶]، با روش‌های پیچیده‌ای طراحی شده‌اند و پیشرفت‌های چشمگیری داشته‌اند. این تهدیدات، از سرقت اطلاعات حساس گرفته تا ایجاد اختلال در عملکرد دستگاه‌ها، چالش‌های امنیتی جدی ایجاد کرده‌اند. از این رو، نیاز به سیستمی قدرتمند و کارآمد برای تشخیص بدافزارهای اندرویدی بیش از پیش احساس می‌شود. هدف اصلی این پژوهش، تمرکز بر شناسایی بدافزارهای ناشناخته و نادیده (Zero-Day)[۱] است که تا کنون شناسایی نشده‌اند و می‌توانند تهدیداتی پنهان برای کاربران ایجاد کنند.

با توجه به چالش‌ها و نیازهای مطرح شده، اهداف اصلی این تحقیق بدین شرح می‌باشد:

- توسعه یک مدل چندوجهی پیشرفته با نام MAGNET که قادر به تحلیل همزمان داده‌های جدولی، گرافی و ترتیبی باشد.
- بهبود دقت تشخیص بدافزارهای اندرویدی با استفاده از معماری ترنسفورمر و مکانیزم‌های توجه پویا[۲].
- کاهش نرخ خطای تشخیص و افزایش قابلیت تعمیم‌پذیری مدل در مواجهه با بدافزارهای جدید.
- بهینه‌سازی مصرف منابع محاسباتی و افزایش سرعت تشخیص با استفاده از الگوریتم‌های پیشرفته.
- ایجاد یک چارچوب استاندارد برای ارزیابی و مقایسه روش‌های مختلف تشخیص بدافزار.

---

[4] Trojan
[5] Spyware
[6] Ransomware
[1] Zero-Day
[2] Attention Mechanism

## ۱ - ۳ سازماندهی پایان نامه

در این پایان‌نامه، ساختار مطالب به گونه‌ای تدوین شده که مسیر پژوهش از مبانی نظری و معرفی مسئله تا ارائه نتایج تجربی به صورت پیوسته و منطقی دنبال شود. به عبارت دیگر، هدف از سازماندهی مطالب این است که خواننده بتواند به راحتی با مباحث پایه، چالش‌ها، روش‌های موجود و نوآوری‌های پیشنهادی آشنا شود و در نهایت به درک جامع از دستاوردهای تحقیق دست یابد. ساختار کلی پایان‌نامه به شرح زیر است:

- فصل 2 – پیشینه تحقیق و مفاهیم پایه:

در این فصل، ابتدا به بررسی کلی امنیت اندروید و اهمیت تشخیص بدافزار پرداخته می‌شود. سپس، چالش‌ها و محدودیت‌های روش‌های سنتی بیان شده و مسئله تحقیق به تفصیل معرفی می‌شود. هدف این فصل ایجاد زمینه نظری مناسب برای درک اهمیت تشخیص خودکار بدافزارهاست.

در ادامه به بررسی جامع مطالعات پیشین در حوزه تشخیص بدافزار اندروید پرداخته می‌شود. در این بخش، رویکردهای مختلف از جمله روش‌های مبتنی بر یادگیری ماشین و یادگیری عمیق مورد تحلیل قرار می‌گیرند. نقاط قوت و ضعف هر یک از این رویکردها همراه با چالش‌های موجود در هر کدام به تفصیل بررسی می‌شود.

- فصل 3 – روش پیشنهادی (MAGNET):

در این فصل، مدل پیشنهادی MAGNET به صورت کامل تشریح می‌شود. ابتدا معماری کلی مدل و اجزای اصلی آن معرفی می‌شوند. سپس، جزئیات پیاده‌سازی و الگوریتم‌های بهینه‌سازی مورد استفاده توضیح داده می‌شود. در نهایت، نوآوری‌های اصلی این روش نسبت به سایر روش‌ها برجسته می‌شود.

- فصل 4 – نتایج و بحث:

این فصل به ارائه نتایج آزمایش‌های انجام شده بر روی چندین مجموعه داده معتبر اختصاص دارد. عملکرد مدل MAGNET از نظر دقت، کارایی و صرفه‌جویی در منابع محاسباتی مورد مقایسه قرار گرفته و نتایج به دست آمده تحلیل می‌شوند.

- فصل 5 – نتیجه‌گیری و پیشنهادات آتی:

در فصل نهایی، یافته‌های اصلی تحقیق به طور خلاصه ارائه شده و به نتیجه‌گیری کلی از دستاوردهای پژوهش پرداخته می‌شود. در این بخش، چالش‌های باقی‌مانده، محدودیت‌های تحقیق و نیز پیشنهاداتی جهت تحقیقات آتی و بهبود رویکرد ارائه می‌شود.

# فصل دوم:
# پیشینه تحقیق و مفاهیم پایه

## ۲-۱ مقدمه

در سال‌های اخیر، با گسترش روزافزون استفاده از دستگاه‌های هوشمند مبتنی بر سیستم‌عامل اندروید، امنیت سایبری به یکی از دغدغه‌های اصلی کاربران و سازمان‌ها تبدیل شده است. محبوبیت گسترده اندروید، که طبق آمار Statista در سال ۲۰۲۴ بیش از ۷۰ درصد بازار جهانی دستگاه‌های هوشمند را در اختیار دارد [**Statista۲۰۲۴**]، آن را به هدف اصلی حملات بدافزاری، از جمله تروجان‌ها[۱]، جاسوس‌افزارها[۲] و باج‌افزارها[۳]، مبدل ساخته است [**AndroidSecurity**]. این تهدیدات، با بهره‌گیری از روش‌های پیچیده و نوظهور، چالش‌های جدی در شناسایی و خنثی‌سازی ایجاد کرده‌اند، به‌ویژه در مورد بدافزارهای ناشناخته (Zero-Day)[۴] که تشخیص آن‌ها با روش‌های سنتی مبتنی بر امضا دشوار است [**AndroidMalwareSurvey**]. از این رو، توسعه مدل‌های پیشرفته مبتنی بر یادگیری عمیق و داده‌های چندوجهی، به‌عنوان راهکاری نوین برای مقابله با این تهدیدات، مورد توجه فزاینده‌ای قرار گرفته است [**DeepLearningMalware**].

فصل حاضر به‌عنوان پایه‌ای برای درک روش پیشنهادی این پژوهش، به بررسی مفاهیم پایه و مرور مطالعات پیشین در حوزه تشخیص بدافزارهای اندرویدی می‌پردازد. ابتدا، مفاهیم کلیدی و اصطلاحات فنی مورد نیاز معرفی خواهند شد. سپس، تکنیک‌های رایج یادگیری ماشین و یادگیری عمیق که در این حوزه کاربرد دارند، تشریح می‌شوند. در ادامه، ابزارها و روش‌های بهینه‌سازی مانند ترنسفورمرها[۵]، الگوریتم‌های PIRATES و Optuna[۶]، که در طراحی مدل MAGNET (روش پیشنهادی این پژوهش) به کار رفته‌اند، معرفی خواهند شد. همچنین، تکنیک‌های آموزشی و معیارهای ارزیابی عملکرد، نظیر اعتبارسنجی متقاطع[۷] و معیارهای دقت[۸]، F1 Score[۹] و AUC[۱۰]، به‌منظور تبیین چارچوب ارزیابی مدل توضیح داده می‌شوند.

در بخش بعدی، پیشینه تحقیق با تمرکز بر روش‌های تحلیل ایستا[۱۱] و پویا[۱۲]، که

[1] Trojan
[2] Spyware
[3] Ransomware
[4] Zero-Day
[5] Transformer
[6] Optuna
[7] Cross-Validation
[8] Accuracy
[9] و
[10] Area Under Curve (AUC)
[11] Static Analysis
[12] Dynamic Analysis

پایه‌های اصلی تشخیص بدافزار را تشکیل می‌دهند، مورد بررسی قرار می‌گیرد. این مرور، با تحلیل نقاط قوت و ضعف روش‌های پیشین، زمینه‌ای برای درک نوآوری‌های مدل پیشنهادی فراهم می‌کند. هدف این فصل، ارائه دیدگاهی جامع و منسجم از مفاهیم و مطالعات مرتبط است تا خواننده را برای درک بهتر روش‌شناسی و نتایج پژوهش آماده سازد. این ساختار، نه‌تنها به تبیین مسائل نظری کمک می‌کند، بلکه راه را برای مقایسه و ارزیابی عملکرد مدل MAGNET هموار می‌سازد.

## ۲-۲ مفاهیم پایه

در این بخش، مفاهیم بنیادی مرتبط با موضوع پژوهش، از جمله تاریخچه مختصری از یادگیری ماشین، انواع بدافزارها، ساختار اپلیکیشن‌های اندروید، داده‌های چندوجهی و مدل‌های یادگیری عمیق مرتبط، تشریح می‌شوند.

### ۲-۲-۱ تکامل روش‌های یادگیری ماشین

در دهه‌های ۱۹۵۰ و ۱۹۶۰، اولین تلاش‌ها برای پردازش زبان طبیعی به وسیله‌ی روش‌های نمادین انجام شد. پژوهشگران در آن زمان سعی می‌کردند ساختارهای دستوری و قوانین زبان را به صورت صریح و دستی تعریف کنند. این رویکردها با وجود تلاش‌های ارزشمند، به دلیل محدودیت‌های محاسباتی و عدم وجود داده‌های کافی، نتوانستند به دقت و کارایی مورد انتظار دست یابند.

با گذر زمان و ورود به دهه‌های ۱۹۶۰ و ۱۹۷۰، رویکردهای آماری جایگزین بخش‌هایی از روش‌های نمادین شدند. در این دوران، مدل‌های n-gram که بر مبنای احتمال وقوع یک کلمه با توجه به کلمات قبلی محاسبه می‌شدند، به عنوان اولین قدم‌های موفق در مدلسازی زبان مطرح شدند. اگرچه این مدل‌ها ساده بودند، اما توانستند برخی از پیچیدگی‌های اولیه‌ی پردازش زبان را کاهش دهند.

در دهه‌های ۱۹۸۰ و ۱۹۹۰، با پیشرفت‌های چشمگیر در فناوری‌های محاسباتی و افزایش دسترسی به داده‌های متنی، روش‌های یادگیری ماشین وارد عرصه شدند. الگوریتم‌های یادگیری نظارت‌شده و غیرنظارتی به منظور تشخیص الگوهای زبانی به کار گرفته شدند. با این حال، محدودیت‌های موجود همچنان مانع از دستیابی به درک عمیق‌تر و تولید متن‌های طبیعی به سطح امروزی می‌شدند.

ورود به قرن ۲۱ و به‌ویژه دهه ۲۰۱۰، با ظهور شبکه‌های عصبی عمیق مانند شبکه‌های

عصبی کانولوشنی [1] ، شبکه‌های عصبی بازگشتی [2] و شبکه‌های حافظه بلندمدت_ کوتاه‌مدت [3] همراه بود. این مدل‌ها توانستند وابستگی‌های زمانی، الگوهای تصویری و روابط بلندمدت موجود در داده‌ها را بهتر مدل‌سازی کنند [**Vinayakumar۲۰۱۹**]. با این حال، چالش‌هایی همچنان در زمینه بهبود کیفیت و کارایی تحلیل داده‌های پیچیده مانند کدهای بدافزار وجود داشت که منجر به توسعه معماری‌های پیشرفته‌تری مانند ترنسفورمرها شد.

### ۲-۲-۲ انواع بدافزار

بدافزار (Malware) اصطلاحی کلی برای هر نوع نرم‌افزار مخرب است که با هدف آسیب رساندن به سیستم‌ها، سرقت اطلاعات یا اختلال در عملکرد طراحی می‌شود. در ادامه، برخی از رایج‌ترین انواع بدافزارها معرفی می‌شوند:

#### ۲-۲-۲-۱ باج‌افزار

باج‌افزار[1] گونه‌ای از بدافزارها است که با سرعت زیادی گسترش یافته و امروزه به شکل فراگیری در حال شیوع است. باج‌افزارها عمدتاً به دو نوع قفل‌کننده (کریپتور)[2] و مسدودکننده[3] تقسیم می‌شوند. پس از آلوده‌سازی رایانه، نوع قفل‌کننده داده‌های ارزشمند کاربر از جمله اسناد، تصاویر و پایگاه‌های داده را رمزنگاری می‌کند و تا زمانی که رمزگشایی نشوند، هیچ فایلی قابل دسترسی نخواهد بود. مهاجمان در ازای ارائه کلید رمزگشا برای بازگرداندن فایل‌های قفل‌شده، درخواست باج می‌کنند. نوع مسدودکننده نیز دسترسی به کل سیستم آلوده را مسدود می‌کند و معمولاً میزان باج‌خواهی آن کمتر از نوع قفل‌کننده است.

باج‌افزارها می‌توانند سیستم‌عامل‌های مختلفی مانند ویندوز، مک‌اواس[4]، لینوکس و اندروید را هدف قرار دهند و هم رایانه‌های دسکتاپ و هم دستگاه‌های موبایل را آلوده سازند [**AndroidSecurity**]. آلودگی معمولاً از طریق باز کردن فایل‌های ضمیمه مخرب، کلیک روی لینک‌های مشکوک یا نصب برنامه‌ها از منابع غیررسمی رخ می‌دهد. حتی وب‌سایت‌های رسمی نیز گاهی به واسطه شبکه‌های تبلیغاتی آلوده می‌شوند. حذف باج‌افزارها معمولاً دشوار است و در صورت رمزنگاری فایل‌ها، کاربر باید برای بازیابی

[1] Convolutional Neural Networks (CNNs)
[2] Recursive Neural Networks (RNNs)
[3] Long Short-Term Memory (LSTM)
[1] Ransomware
[2] Cryptor
[3] Locker
[4] Mac OS X

دسترسی، رمزگشایی انجام دهد که پرداخت باج نیز تضمینی برای بازگشت اطلاعات نیست و توصیه نمی‌شود.

### ۲-۲-۲-۲ تروجان

تروجان[۵] یا اسب تروآ، نوعی برنامه مخرب است که خود را به عنوان نرم‌افزاری مشروع و بی‌خطر جلوه می‌دهد تا کاربر را فریب داده و به سیستم نفوذ کند. نام‌گذاری این بدافزار برگرفته از داستان اسب چوبی تروای یونانیان است که با حیله وارد شهر شد. انواع مختلفی از تروجان‌ها وجود دارد، از جمله:

- **تروجان دسترسی از راه دور (Backdoor):** به مهاجم امکان کنترل سیستم قربانی از راه دور را می‌دهد.
- **تروجان ارسال داده:** اطلاعات حساس مانند رمزهای عبور و داده‌های واردشده توسط کاربر را به مهاجم ارسال می‌کند.
- **تروجان مخرب:** فایل‌های سیستمی را حذف یا تخریب می‌کند.
- **تروجان DoS:** باعث کاهش سرعت سیستم و اینترنت کاربر می‌شود (حملات منع سرویس).
- **تروجان پراکسی:** به مهاجم اجازه می‌دهد از طریق سیستم قربانی به سرورهای پراکسی حمله کند.
- **تروجان FTP:** پورت FTP را باز کرده و کنترل سیستم را از این طریق به مهاجم می‌دهد.
- **تروجان غیرفعال‌سازی امنیت:** نرم‌افزارهای امنیتی را غیرفعال می‌کند تا حمله آسان‌تر انجام شود.

### ۲-۲-۲-۳ جاسوس‌افزار

جاسوس‌افزار[۱] نوعی نرم‌افزار مخرب است که بدون اطلاع کاربر، اطلاعاتی درباره او یا فعالیت‌هایش را جمع‌آوری و برای مهاجم ارسال می‌کند. کاربردهای رایج جاسوس‌افزارها

[5] Trojan Horse
[1] Spyware

شامل تبلیغات هدفمند، جمع‌آوری اطلاعات شخصی (مانند اطلاعات بانکی یا رمزهای عبور) و ایجاد تغییرات ناخواسته در تنظیمات سیستم قربانی است. این بدافزارها می‌توانند باعث کاهش کارایی سیستم، نصب نوارابزارهای ناخواسته، تغییر صفحه خانگی مرورگر و باز شدن صفحات پاپ‌آپ تبلیغاتی شوند.

#### ۲-۲-۲-۴ تبلیغ‌افزار

تبلیغ‌افزار[۲] نرم‌افزاری است که با هدف نمایش تبلیغات ناخواسته روی سیستم قربانی طراحی شده است. این تبلیغات معمولاً به صورت بنرها یا صفحات پاپ‌آپ ظاهر می‌شوند و گاهی صفحات اینترنتی متعددی را به طور متوالی باز می‌کنند. اگرچه همه تبلیغ‌افزارها ذاتاً مخرب نیستند (برخی نرم‌افزارهای رایگان از طریق نمایش تبلیغ درآمدزایی می‌کنند)، اما انواع تهاجمی آن‌ها می‌توانند فعالیت‌های آنلاین کاربر را دنبال کرده و تجربه کاربری را مختل کنند. همچنین، برخی تبلیغ‌افزارها اطلاعات مربوط به رفتار کاربر را جمع‌آوری و برای اهداف تبلیغاتی یا حتی فروش به اشخاص ثالث ارسال می‌کنند که می‌تواند حریم خصوصی را نقض کند.

### ۲-۲-۳ مفاهیم مرتبط با اپلیکیشن‌های اندرویدی

#### ۲-۲-۳-۱ مجوزهای دسترسی

اندروید از یک سیستم تخصیص مجوز[۳] برخوردار است که برای هر عملیات یا وظیفه (Task) حساس، مجوزها و سطوح دسترسی خاصی را تعیین می‌کند. هر اپلیکیشن می‌تواند برای انجام عملیات خاص، مجوزهای لازم را از این سامانه درخواست نماید. به عنوان مثال، یک برنامه کاربردی ممکن است برای دسترسی به اینترنت، دوربین، یا لیست مخاطبین، مجوزهای مربوطه را درخواست کند. این سیستم دارای سطوح مختلفی برای مجوزها است که به آن‌ها سطح حفاظت (Protection Level) گفته می‌شود.

در نسخه‌های قدیمی‌تر اندروید (قبل از نسخه ۰.۶)، اپلیکیشن‌ها هنگام نصب، لیست تمامی مجوزهای مورد نیاز خود را به کاربر اعلام می‌کردند و کاربر یا همه را می‌پذیرفت یا نصب را لغو می‌کرد. اما از نسخه ۰.۶ اندروید (Marshmallow، API سطح ۲۳) به بعد، مدل مجوزدهی پویا (Runtime Permissions) معرفی شد که طی آن، مجوزهای حساس تنها در زمان اجرای برنامه و هنگامی که اپلیکیشن برای اولین بار به آن قابلیت نیاز دارد،

[2] Adware
[3] Permission System

از کاربر درخواست می‌شوند. اپلیکیشن‌های اندرویدی مجوزهای مورد نیاز خود را در فایل مانیفست اندروید (AndroidManifest.xml) اعلام می‌کنند. همچنین، اپلیکیشن می‌تواند مجوزهای اضافی را در این فایل تعریف کند که دسترسی به برخی اجزای داخلی نرم‌افزار توسط سایر برنامه‌ها را محدود می‌سازد.

در اندروید، مجوزها عمدتاً به دو سطح دسترسی و امنیت تقسیم می‌شوند:

- **معمولی (Normal):** مجوزهایی که ریسک پایینی برای حریم خصوصی کاربر یا عملکرد سایر اپلیکیشن‌ها دارند (مانند مجوز دسترسی به اینترنت یا تنظیم منطقه زمانی). این مجوزها معمولاً به صورت خودکار هنگام نصب یا به‌روزرسانی به اپلیکیشن اعطا می‌شوند و نیازی به تأیید صریح کاربر ندارند.
- **خطرناک (Dangerous):** مجوزهایی که به اطلاعات خصوصی کاربر (مانند مخاطبین، موقعیت مکانی، پیامک‌ها) دسترسی دارند یا می‌توانند بر عملکرد دستگاه یا سایر اپلیکیشن‌ها تأثیر بگذارند (مانند دسترسی به دوربین یا میکروفون). این مجوزها باید حتماً در زمان اجرا و به صورت صریح از کاربر درخواست شوند.

تحلیل الگوهای درخواست مجوز یکی از روش‌های مهم در تشخیص بدافزارهای اندرویدی است، زیرا بدافزارها اغلب مجوزهای خطرناک بیشتری نسبت به کاربرد ظاهری خود درخواست می‌کنند [**Drebin**].

### ۲-۲-۳-۲ فایل APK

فرمت APK[1] مخفف عبارت Android Package Kit است و فرمت فایل بسته‌بندی شده‌ای است که برای توزیع و نصب برنامه‌ها و میان‌افزارها در سیستم‌عامل اندروید استفاده می‌شود. فایل APK شامل تمام کدهای برنامه (مانند فایل‌های dex.)، منابع (مانند تصاویر و لایه‌ها)، دارایی‌ها، گواهی‌نامه‌ها و فایل مانیفست (AndroidManifest.xml) است. این فایل بسیار شبیه به فرمت‌های بسته‌بندی دیگر مانند APPX در مایکروسافت ویندوز یا deb. در سیستم‌های مبتنی بر دبیان (مانند اوبونتو) عمل می‌کند.

برای ساخت یک فایل APK، ابتدا برنامه اندروید با استفاده از ابزارهایی مانند اندروید استودیو کامپایل شده و سپس تمام اجزای آن در یک فایل فشرده با پسوند apk. بسته‌بندی می‌شود. فایل‌های APK پایه و اساس برنامه‌های اندرویدی هستند و بخش اصلی اکوسیستم

[1] Android Package Kit

اندروید محسوب می‌شوند. کاربران می‌توانند فایل‌های APK را از منابع مختلف، از جمله فروشگاه رسمی گوگل پلی (Google Play) یا به صورت دستی (Sideloading) از وب‌سایت‌ها یا منابع دیگر، دانلود و نصب کنند. نصب دستی برنامه‌ها از منابع نامعتبر یکی از راه‌های اصلی ورود بدافزار به دستگاه‌های اندرویدی است.

علاوه بر سیستم‌عامل اندروید، فایل‌های APK را می‌توان با استفاده از شبیه‌سازها یا لایه‌های سازگاری در سایر سیستم‌عامل‌ها مانند ویندوز، مک‌اواس یا کروم‌اواس نیز اجرا، و با ابزارهای مهندسی معکوس، محتوای آن‌ها را استخراج، ویرایش یا تحلیل کرد.

### ۲-۲-۳-۳ سورس‌کد

سورس‌کد (Source Code) یا کد منبع برنامه، به مجموعه‌ای از دستورالعمل‌ها یا عبارات متنی اشاره دارد که توسط برنامه‌نویس به یک زبان برنامه‌نویسی خاص (مانند جاوا، کاتلین، C++، پایتون) نوشته می‌شود تا یک برنامه کامپیوتری، عملکرد مورد نظر را پیاده‌سازی کند. سورس‌کد برای انسان قابل خواندن است و منطق، الگوریتم‌ها و ساختار برنامه را تعریف می‌کند.

برای اینکه کامپیوتر بتواند سورس‌کد را اجرا کند، باید ابتدا به زبان ماشین (کد باینری) ترجمه شود. این فرآیند توسط یک کامپایلر یا مفسر انجام می‌شود. در اکوسیستم اندروید، کدهای جاوا یا کاتلین معمولاً به بایت‌کد Dalvik (در نسخه‌های قدیمی‌تر) یا Android ART (Runtime) کامپایل می‌شوند که در فایل‌های dex. درون فایل APK قرار می‌گیرند.

دسترسی به سورس‌کد یک برنامه امکان درک کامل عملکرد، اصلاح یا توسعه آن را فراهم می‌کند. در حوزه امنیت، تحلیل سورس‌کد (در صورت در دسترس بودن) یکی از روش‌های تحلیل ایستا برای یافتن آسیب‌پذیری‌ها یا کدهای مخرب است. با این حال، اکثر برنامه‌های تجاری یا بدافزارها به صورت کامپایل‌شده و بدون سورس‌کد توزیع می‌شوند و تحلیل آن‌ها نیازمند مهندسی معکوس (مانند دیس‌اسمبل کردن کدهای باینری یا بایت‌کد) است.

## ۲-۲-۴ داده‌های چندوجهی در تشخیص بدافزار

داده‌های چندوجهی (Multi-Modal Data) به مجموعه‌ای از اطلاعات اشاره دارد که از منابع و قالب‌های متنوعی استخراج شده و برای تحلیل جامع‌تر یک پدیده، مانند تشخیص بدافزارهای اندرویدی، به کار می‌روند. در این پژوهش، مدل MAGNET با بهره‌گیری از

سه نوع اصلی داده‌های چندوجهی، شامل داده‌های جدولی، گرافی و ترتیبی، طراحی شده است تا بتواند ویژگی‌های متنوع و مکمل اپلیکیشن‌های اندرویدی را به‌صورت یکپارچه پردازش کند. این رویکرد، برخلاف روش‌های سنتی که تنها به یک نوع داده (مثلاً فقط مجوزها یا فقط فراخوانی‌های API) وابسته‌اند، امکان درک عمیق‌تر رفتارهای مخرب را فراهم می‌سازد. در ادامه، هر یک از این انواع داده‌ها به‌طور جداگانه توضیح داده می‌شود.

### ۲-۲-۴-۱ داده‌های جدولی

داده‌های جدولی شامل ویژگی‌های ایستای اپلیکیشن‌های اندرویدی هستند که به‌صورت ساختارمند و معمولاً عددی یا دسته‌ای ارائه می‌شوند. این ویژگی‌ها می‌توانند شامل تعداد مجوزهای درخواست‌شده، اندازه فایل APK، نسخه SDK هدف، تعداد فعالیت‌ها (Activities)، سرویس‌ها (Services)، گیرنده‌های پخش (Broadcast Receivers)، ارائه‌دهندگان محتوا (Content Providers)، تعداد فراخوانی‌های API خاص، یا سایر متغیرهای استخراج‌شده از فایل AndroidManifest.xml یا کد کامپایل شده باشند. در این پژوهش، این داده‌ها با استفاده از کتابخانه PyTorch و به‌صورت اشیای تانسور (torch.Tensor) بارگذاری و پردازش می‌شوند. برای هر نمونه اپلیکیشن، یک بردار ویژگی با ابعاد مشخص تعریف می‌شود که نشان‌دهنده خصوصیات ایستای آن است. پیش‌پردازش این داده‌ها، مانند نرمال‌سازی (برای مقادیر عددی) یا کدگذاری وان‌هات (برای مقادیر دسته‌ای)، که با هدف بهبود عملکرد مدل انجام می‌شود، از مراحل کلیدی به شمار می‌رود. این نوع داده، پایه‌ای برای تحلیل‌های آماری و یادگیری الگوهای ساده در بدافزارها فراهم می‌کند (مثلاً بدافزارها ممکن است به طور متوسط مجوزهای بیشتری درخواست کنند).

### ۲-۲-۴-۲ داده‌های گرافی

داده‌های گرافی ساختارهایی هستند که روابط و تعاملات بین اجزای مختلف یک اپلیکیشن اندرویدی را مدل‌سازی می‌کنند. این داده‌ها می‌توانند نمایانگر گراف فراخوانی توابع (Call Graph)، گراف جریان کنترل (Control Flow Graph, CFG)، گراف وابستگی داده‌ها (Data Dependency Graph)، یا گراف روابط بین اجزای برنامه (مانند ارتباط بین فعالیت‌ها از طریق Intents) باشند. گره‌ها (Nodes) در این گراف‌ها می‌توانند توابع، بلوک‌های کد، کلاس‌ها، مجوزها یا اجزای اندرویدی باشند و لبه‌ها (Edges) نشان‌دهنده روابطی مانند فراخوانی تابع، انتقال کنترل، جریان داده یا درخواست مجوز هستند. در این پژوهش،

از کتابخانه PyTorch Geometric (torch_geometric) برای بارگذاری و پردازش این داده‌ها به‌صورت اشیای داده (Data) استفاده شده است. هر گراف شامل ماتریس همجواری (ساختار گراف) و ویژگی‌های گره‌ها (مانند نوع تابع یا بردار ویژگی‌های استخراج‌شده از کد) است. تحلیل گرافی به مدل اجازه می‌دهد تا الگوهای ساختاری پیچیده و وابستگی‌های پنهان بین اجزا را شناسایی کند (مانند شناسایی یک حلقه فراخوانی مشکوک یا یک خوشه از توابع مرتبط با سرقت اطلاعات)، که در تشخیص بدافزارهای پیچیده یا ناشناخته بسیار مؤثر است.

### ۲-۲-۴-۳ داده‌های ترتیبی

داده‌های ترتیبی شامل توالی‌هایی از رویدادها یا مقادیر هستند که معمولاً نمایانگر رفتار زمانی یا ترتیب وقوع عملیات در اپلیکیشن‌ها هستند. در زمینه تشخیص بدافزار اندروید، این داده‌ها می‌توانند شامل توالی فراخوانی‌های API سیستمی در حین اجرای برنامه (استخراج‌شده از تحلیل پویا)، توالی رویدادهای سیستمی (مانند ارسال پیامک، اتصال به شبکه)، یا الگوهای زمانی استفاده از منابع (مانند مصرف CPU یا باتری) باشند. در این پژوهش، این داده‌ها با استفاده از تانسورهای LongTensor در PyTorch بارگذاری شده و برای هر نمونه یک یا چند توالی مشخص تعریف می‌شود. پیش‌پردازش این داده‌ها ممکن است شامل نگاشت هر رویداد یا API به یک شناسه عددی، پدینگ (Padding) توالی‌ها برای رسیدن به طول یکسان، و تبدیل به فرمت قابل پردازش توسط مدل‌های ترتیبی مانند RNN، LSTM یا ترنسفورمر باشد. استفاده از داده‌های ترتیبی به مدل امکان می‌دهد تا رفتار پویای اپلیکیشن‌ها را تحلیل کند و الگوهای زمانی مخرب، مانند توالی خاصی از فراخوانی‌ها که منجر به نشت داده می‌شود یا حملات دوره‌ای، را تشخیص دهد.

### ۲-۲-۴-۴ اهمیت و یکپارچگی داده‌های چندوجهی

ترکیب این سه نوع داده در مدل MAGNET، با استفاده از مکانیزم‌های توجه پویا و تکنیک‌های همجوشی (Fusion)، به شناسایی دقیق‌تر و جامع‌تر بدافزارها کمک می‌کند. داده‌های جدولی اطلاعات کلی و ایستا، داده‌های گرافی روابط ساختاری پیچیده، و داده‌های ترتیبی رفتار پویا را ارائه می‌دهند که در کنار هم، تصویری کامل‌تر از اپلیکیشن را ترسیم می‌کنند. این رویکرد چندوجهی، به‌ویژه در مواجهه با بدافزارهای پیچیده و ناشناخته که ممکن است در یک وجه خاص (مثلاً فقط مجوزها) عادی به نظر برسند، مزیت رقابتی

نسبت به روش‌های تک‌وجهی دارد و تعمیم‌پذیری مدل را افزایش می‌دهد [Marastoni۲۰۲۲].
استخراج این داده‌ها از فایل‌های پردازش‌شده با توابع فرضی load_processed_data، load_graph_data، و load_sequence_data انجام شده و پیش‌پردازش آن‌ها با نرمال‌سازی، استانداردسازی و کدگذاری مناسب، تضمین‌کننده سازگاری با معماری‌های یادگیری عمیق مانند ترنسفورمر است.

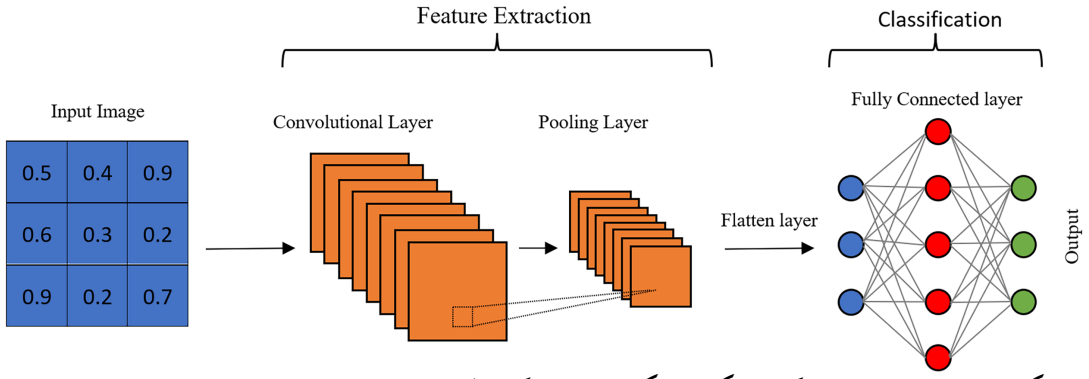
### ۲-۲-۵ مدل‌های یادگیری عمیق

یادگیری عمیق، زیرشاخه‌ای از یادگیری ماشین، از شبکه‌های عصبی مصنوعی با چندین لایه (لایه‌های عمیق) برای یادگیری بازنمایی‌های پیچیده از داده‌ها استفاده می‌کند. در ادامه، چند معماری کلیدی که در تشخیص بدافزار کاربرد دارند، معرفی می‌شوند.

#### ۲-۲-۵-۱ شبکه‌های عصبی کانولوشنی (CNN)

شبکه‌های عصبی کانولوشنی (Convolutional Neural Networks, CNNs) به طور خاص برای پردازش داده‌های با ساختار شبکه‌ای یا گریدمانند، مانند تصاویر، ماتریس‌های ویژگی یا داده‌های ترتیبی با پنجره لغزان، طراحی شده‌اند. این شبکه‌ها با بهره‌گیری از لایه‌های کانولوشنی که فیلترهای کوچکی را روی ورودی حرکت می‌دهند، قادرند ویژگی‌های محلی (مانند الگوهای خاص در بایت‌های کد، توالی‌های API، یا روابط بین مجوزها در یک ماتریس هم‌وقوعی) را به صورت سلسله‌مراتبی و خودکار استخراج کنند. لایه‌های تجمعی (Pooling) نیز با کاهش ابعاد مکانی داده و فشرده‌سازی اطلاعات، به افزایش پایداری نسبت به تغییرات جزئی و کاهش محاسبات کمک می‌کنند. در نهایت، لایه‌های کاملاً متصل (Fully Connected) از ویژگی‌های استخراج‌شده برای طبقه‌بندی نهایی (مثلاً بدافزار یا سالم) استفاده می‌کنند. در حوزه تشخیص بدافزار، شبکه‌های CNN می‌توانند برای تحلیل بایت‌های فایل اجرایی به عنوان تصویر [Nataraj۲۰۱۱]، ماتریس‌های ویژگی‌های ایستا، یا توالی‌های رفتاری به کار روند و الگوهای مخرب را شناسایی نمایند [Vinayakumar۲۰۱۹].

تحقیقات اخیر نشان داده است که شبکه‌های عصبی کانولوشنی می‌توانند دقت بالایی در تشخیص بدافزار ارائه دهند و ویژگی‌های مورد نیاز را مستقیماً از داده‌های خام یا نیمه‌پردازش‌شده یاد بگیرند [Alsaleh۲۰۲۳]. ساختار این شبکه‌ها معمولاً به صورت انتها به انتها (End-to-End) طراحی می‌شود و شامل پشته‌ای از لایه‌های کانولوشنی، فعال‌سازی (مانند ReLU)، و تجمعی است که به دنبال آن یک یا چند لایه کاملاً متصل قرار می‌گیرد.




شکل ۲ - ۱ ساختار کلی یک شبکه عصبی کانولوشنی (CNN): ابتدا تصویر ورودی یا ماتریس ویژگی وارد شبکه می‌شود. در مرحله استخراج ویژگی، لایه‌های کانولوشنی و تجمعی به ترتیب ویژگی‌های محلی را استخراج و ابعاد داده را کاهش می‌دهند. سپس داده‌ها به یک بردار یک‌بعدی تبدیل شده (Flattening) و وارد لایه‌های کاملاً متصل می‌شوند تا فرایند طبقه‌بندی نهایی انجام گیرد. این ساختار باعث می‌شود شبکه بتواند به صورت خودکار و بدون نیاز به مهندسی ویژگی دستی، الگوهای پیچیده را شناسایی کند. برگرفته از [**۲۰۲۳Alsaleh**].

افزایش عمق شبکه (تعداد لایه‌ها) امکان استخراج ویژگی‌های انتزاعی‌تر و سطح بالاتر را فراهم می‌کند.

### ۲ - ۲ - ۵ - ۲ شبکه‌های عصبی بازگشتی و واحدهای حافظه بلند_کوتاه (LSTM و RNN)

شبکه‌های عصبی بازگشتی (Recurrent Neural Networks, RNNs) نوعی از شبکه‌های عصبی هستند که به طور ویژه برای پردازش داده‌های ترتیبی و زمانی طراحی شده‌اند. برخلاف شبکه‌های پیش‌خور (Feedforward)، RNNها دارای حلقه‌های بازگشتی در اتصالات خود هستند که به آن‌ها اجازه می‌دهد اطلاعات مربوط به مراحل قبلی توالی را در یک حالت پنهان (Hidden State) حفظ کرده و از آن برای پردازش مراحل بعدی استفاده کنند. این "حافظه" داخلی، RNNها را برای کاربردهایی مانند تحلیل توالی فراخوانی‌های API، پردازش زبان طبیعی (مانند تحلیل کدهای اسمبلی یا توضیحات برنامه)، و تشخیص رفتارهای پویای اپلیکیشن‌ها که در طول زمان آشکار می‌شوند، مناسب می‌سازد.

یکی از چالش‌های اصلی RNNهای ساده، مشکل محو یا انفجار گرادیان (Vanishing/Exploding Gradient) در هنگام یادگیری وابستگی‌های بلندمدت در توالی‌های طولانی است. برای رفع این مشکل، ساختارهای پیشرفته‌تری مانند واحد حافظه بلند_کوتاه (Long Short-Term Memory, LSTM) [**۱۹۹۷Hochreiter**] و واحد بازگشتی دروازه‌ای (Gated Recurrent Unit, GRU) [**۲۰۱۴Cho**] معرفی شده‌اند. LSTM با بهره‌گیری از یک سلول

حافظه (Memory Cell) و سه نوع گیت (گیت فراموشی، گیت ورودی و گیت خروجی)، امکان کنترل دقیق جریان اطلاعات را فراهم می‌کند. این گیت‌ها به شبکه اجازه می‌دهند تا تصمیم بگیرد کدام اطلاعات را در سلول حافظه نگه دارد، کدام را فراموش کند و کدام را به عنوان خروجی مرحله فعلی استفاده کند. این مکانیزم به LSTMها کمک می‌کند تا اطلاعات مهم را در طول توالی‌های بسیار طولانی حفظ کرده و وابستگی‌های بلندمدت را به طور مؤثرتری مدل‌سازی کنند. GRU ساختار ساده‌تری نسبت به LSTM دارد (با دو گیت) اما عملکرد مشابهی در بسیاری از وظایف ارائه می‌دهد.

در این پژوهش، از LSTM (یا می‌توانست از GRU استفاده شود) برای تحلیل داده‌های ترتیبی، مانند توالی فراخوانی‌های API، به منظور شناسایی الگوهای رفتاری پویا در اپلیکیشن‌های اندرویدی استفاده شده است. این قابلیت، LSTM را به ابزاری قدرتمند برای تشخیص رفتارهای مخربی که در طول زمان و از طریق دنباله‌ای از اقدامات رخ می‌دهند، تبدیل می‌کند.

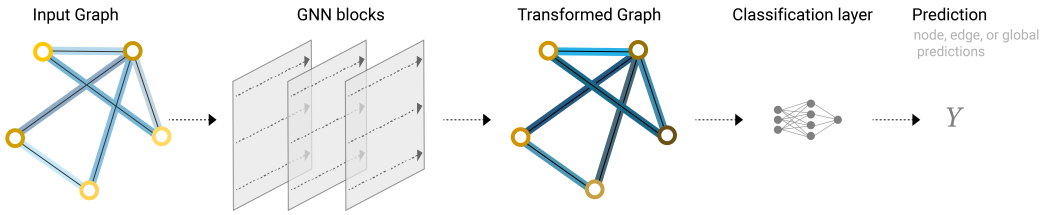
### ۲-۲-۵-۳ شبکه‌های عصبی گرافی (GNN)

شبکه‌های عصبی گرافی (Graph Neural Networks, GNNs) دسته‌ای از مدل‌های یادگیری عمیق هستند که برای پردازش داده‌هایی با ساختار گرافی (مانند شبکه‌های اجتماعی، های شیمیایی، یا گراف‌های فراخوانی توابع در نرم‌افزار) طراحی شده‌اند **[a۲۰۲۱sanchez-lengeling]**. داده‌های گرافی شامل مجموعه‌ای از گره‌ها (Nodes) و لبه‌ها (Edges) هستند که به ترتیب موجودیت‌ها و روابط بین آن‌ها را نشان می‌دهند. GNNها قادرند هم ساختار (توپولوژی) گراف و هم ویژگی‌های گره‌ها و لبه‌ها را برای یادگیری بازنمایی‌های مفید به کار گیرند.

ایده اصلی در GNNها، به‌روزرسانی بازنمایی (بردار ویژگی) هر گره بر اساس اطلاعات گره‌های همسایه آن است. این فرآیند معمولاً شامل دو مرحله اصلی است که در هر لایه GNN تکرار می‌شود:

۱. **تجمیع (Aggregation):** جمع‌آوری اطلاعات (بازنمایی‌ها) از گره‌های همسایه. روش‌های مختلفی برای تجمیع وجود دارد، مانند میانگین‌گیری، جمع یا حداکثرگیری.

۲. **به‌روزرسانی (Update):** ترکیب اطلاعات تجمیع‌شده از همسایگان با اطلاعات خود گره (بازنمایی فعلی آن) برای تولید بازنمایی جدید گره. این کار معمولاً با استفاده از یک شبکه عصبی کوچک (مانند یک لایه خطی و یک تابع فعال‌سازی) انجام می‌شود.




شکل ۲-۲ نمایی از یک مدل شبکه عصبی گرافی (GNN): ابتدا گراف ورودی (با ویژگی‌های اولیه گره‌ها و ساختار اتصالات) به مدل داده می‌شود. سپس در چندین بلوک GNN (لایه)، اطلاعات گره‌ها با تجمیع اطلاعات از همسایگان و به‌روزرسانی با استفاده از شبکه‌های عصبی کوچک، به‌طور مکرر پالایش می‌شود تا بازنمایی‌های غنی‌تری (گراف تبدیل‌شده) حاصل شود. در نهایت، این بازنمایی‌ها می‌توانند برای وظایف مختلف مانند طبقه‌بندی کل گراف (مثلاً بدافزار/سالم)، طبقه‌بندی گره‌ها (مثلاً شناسایی توابع مخرب) یا پیش‌بینی لبه‌ها استفاده شوند. برگرفته و بازطراحی‌شده بر اساس **[a۲۰۲۱sanchez-lengeling]**.

با پشته کردن چندین لایه GNN، هر گره می‌تواند اطلاعات را از همسایگان دورتر نیز دریافت کند و بازنمایی‌های پیچیده‌تری که الگوهای ساختاری سطح بالاتر را در بر می‌گیرند، یاد گرفته شود.

در این پژوهش، از GNN (به‌طور خاص، احتمالاً مدل‌هایی مانند Graph Convolutional Network (GCN) **[۲۰۱۷Kipf]** یا Graph Attention Network (GAT) **[۲۰۱۸Velickovic]**) برای تحلیل داده‌های گرافی استخراج‌شده از اپلیکیشن‌ها (مانند گراف فراخوانی API) استفاده شده است. این داده‌ها با استفاده از کتابخانه torch_geometric به صورت اشیای Data بارگذاری و پردازش می‌شوند. GNNها به مدل کمک می‌کنند تا ویژگی‌های ساختاری مهمی را که ممکن است نشان‌دهنده رفتار مخرب باشند (مانند الگوهای خاصی از تعامل بین توابع یا اجزا) استخراج کنند.

### ۲-۲-۶ ترنسفورمرها

#### ۲-۲-۶-۰-۱ معرفی ترنسفورمر و تحول در مدل‌های یادگیری عمیق

نقطه عطف برجسته‌ای در تکامل معماری‌های یادگیری عمیق، معرفی مدل ترنسفورمر در مقاله "Atten-tion Is All You Need" توسط **attention** بود که انقلابی در پردازش داده‌های ترتیبی، و بعدها انواع دیگر داده‌ها، به ارمغان آورد. این معماری، برخلاف مدل‌های بازگشتی (RNN/LSTM) که توالی‌ها را به صورت مرحله به مرحله پردازش می‌کردند و با مشکلاتی مانند وابستگی‌های بلندمدت و عدم امکان پردازش موازی مواجه بودند، صرفاً بر مکانیزم

توجه (Attention Mechanism)، به‌ویژه توجه خودی (Self-Attention) و توجه چندسر (Multi-Head Attention)، تکیه می‌کند.

مکانیزم توجه به مدل اجازه می‌دهد تا هنگام پردازش یک عنصر در توالی (مثلاً یک کلمه در جمله یا یک فراخوانی API در یک دنباله)، به طور مستقیم به تمام عناصر دیگر توالی نگاه کرده و وزن اهمیتی متفاوتی به هر یک اختصاص دهد. این قابلیت، مدل‌سازی وابستگی‌های بلندمدت بین عناصر را بسیار کارآمدتر می‌کند. همچنین، از آنجایی که محاسبات توجه برای هر عنصر می‌تواند به صورت موازی انجام شود، ترنسفورمرها سرعت آموزش و استنتاج را به‌طور چشمگیری بهبود بخشیدند و امکان آموزش مدل‌های بسیار بزرگتر را فراهم کردند. معماری استاندارد ترنسفورمر شامل دو بخش اصلی است: کدگذار (Encoder) که ورودی را به یک دنباله از بازنمایی‌های پیوسته تبدیل می‌کند و گشاینده (Decoder) که این بازنمایی‌ها را برای تولید خروجی (مثلاً در ترجمه ماشینی یا تولید متن) به کار می‌برد. هر دوی این بخش‌ها از پشته‌هایی از لایه‌های توجه چندسر و شبکه‌های عصبی پیش‌خور (Feed-Forward Networks) تشکیل شده‌اند که با اتصالات باقیمانده (Residual Connections) و نرمال‌سازی لایه‌ای (Layer Normalization) ترکیب شده‌اند.

با ظهور ترنسفورمر، مدل‌های زبانی بزرگ (Large Language Models, LLMs) پیشرفته‌ای مانند BERT **[Devlin۲۰۱۹]** (که از پشته کدگذار ترنسفورمر استفاده می‌کند و برای درک زبان آموزش دیده) و GPT **[Radford۲۰۱۸، Radford۲۰۱۹، Brown۲۰۲۰]** (که از پشته گشاینده ترنسفورمر بهره می‌برد و برای تولید زبان آموزش دیده) توسعه یافتند. این مدل‌ها با پیش‌آموزش روی حجم عظیمی از داده‌های متنی و سپس تنظیم دقیق (Fine-tuning) برای وظایف خاص، عملکردی استثنایی در طیف وسیعی از کاربردهای پردازش زبان طبیعی نشان دادند.

این پیشرفت‌ها، الهام‌بخش کاربرد ترنسفورمرها در حوزه‌های فراتر از زبان، از جمله بینایی کامپیوتر (Vision Transformer) **[Dosovitskiy۲۰۲۱]**، تحلیل داده‌های گرافی **[Yun۲۰۱۹]** و همچنین امنیت سایبری و تشخیص بدافزار **[TransformerMalware]** شد. توانایی ترنسفورمر در مدل‌سازی روابط پیچیده و بلندمدت و پردازش موازی، آن را به گزینه‌ای جذاب برای تحلیل داده‌های چندوجهی و پیچیده مرتبط با بدافزارها تبدیل کرد.

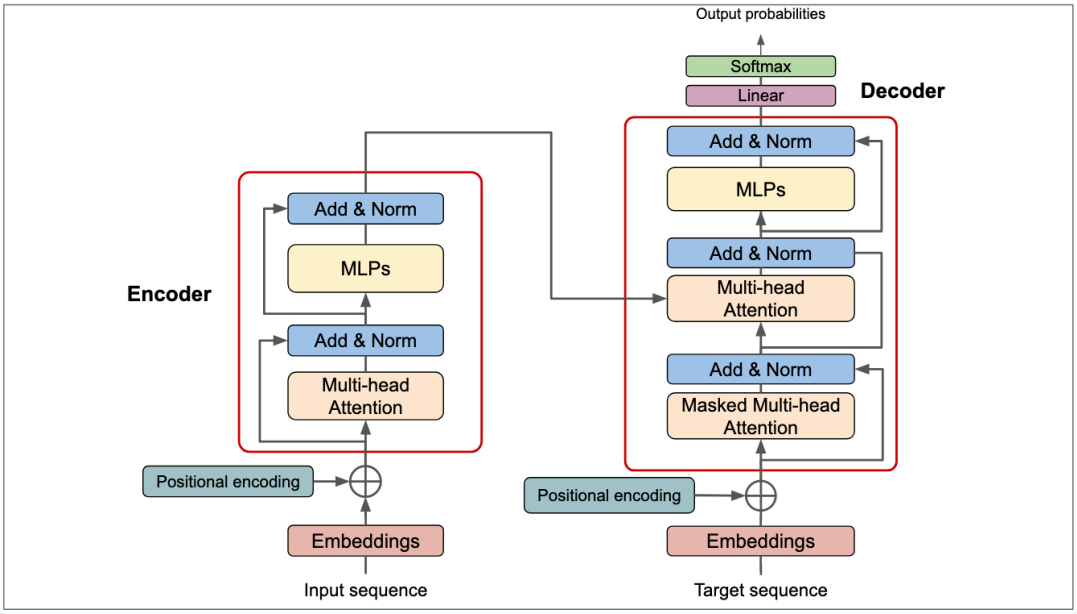
### ۲-۲-۷ ترنسفورمرها در مدل MAGNET

در این پژوهش، معماری ترنسفورمر به‌عنوان هسته اصلی مدل MAGNET (Multi-Modal Analysis for Graph and Network Threat Detection) به کار گرفته شده است تا داده‌های چندوجهی شامل داده‌های جدولی (ویژگی‌های ایستا)، گرافی (روابط ساختاری) و ترتیبی (رفتار پویا) را به‌صورت یکپارچه تحلیل کند. ترنسفورمر در این مدل از مکانیزم توجه پویا برای یادگیری بازنمایی‌های غنی (Embeddings) از هر وجه داده و سپس تلفیق (Fusion) هوشمندانه این بازنمایی‌ها استفاده می‌کند.

به‌طور خاص، لایه‌های کدگذار ترنسفورمر می‌توانند برای پردازش ویژگی‌های استخراج‌شده از داده‌های جدولی (پس از تبدیل به دنباله‌ای از ویژگی‌ها یا استفاده از توکن‌های خاص) و داده‌های گرافی (با استفاده از تکنیک‌هایی مانند تبدیل گره‌ها به دنباله یا به‌کارگیری Graph Transformers) به کار روند. مکانیزم توجه چندسر به مدل اجازه می‌دهد تا روابط متقابل بین ویژگی‌های مختلف ایستا و ساختاری را کشف کند. همزمان، لایه‌های گشاینده ترنسفورمر (یا یک پشته کدگذار دیگر) می‌توانند برای مدل‌سازی توالی‌های ترتیبی (مانند فراخوانی‌های API) به کار گرفته شوند تا الگوهای زمانی مخرب شناسایی شوند.

ساختار ترنسفورمر در MAGNET احتمالاً شامل چندین بلوک کدگذار/گشاینده است که هر بلوک دارای یک لایه توجه چندسر و یک لایه تغذیه روبه‌جلو است. داده‌های ورودی از هر وجه ابتدا با استفاده از لایه‌های جاسازی (Embedding layers، مانند nn.Embedding در PyTorch یا پروجکشن‌های خطی) به بردارهایی با ابعاد ثابت تبدیل می‌شوند. سپس، مکانیزم توجه خودی و احتمالاً توجه متقابل (Cross-Attention) بین وجه‌های مختلف، وزن‌های متفاوتی به هر عنصر داده تخصیص می‌دهد تا اهمیت نسبی آن را در پیش‌بینی نهایی برچسب (سالم یا مخرب) تعیین کند. این فرآیند با استفاده از نرمال‌سازی لایه‌ای و اتصالات باقیمانده برای پایداری و بهبود آموزش، بهینه می‌شود.

مزیت اصلی استفاده از ترنسفورمر در مدل MAGNET، توانایی آن در پردازش موازی داده‌های چندوجهی و درک روابط پیچیده و بلندمدت *درون* هر وجه و *بین* وجه‌های مختلف داده است. این ویژگی، به‌ویژه در شناسایی بدافزارهای پیچیده و ناشناخته که ممکن است از تکنیک‌های پنهان‌سازی پیشرفته استفاده کنند و نیازمند تحلیل جامع رفتار و ساختار اپلیکیشن‌ها هستند، بسیار مؤثر است. نتایج ادعا شده در متن (دقت ۹۷.۲۴ درصد و F1 Score ۰.۹۸۲۳) نشان می‌دهد که این رویکرد مبتنی بر ترنسفورمر و داده‌های چندوجهی، پتانسیل ارائه عملکردی برتر نسبت به مدل‌های سنتی یا تک‌وجهی را دارد.




شکل ۲-۳ ساختار کلی معماری ترنسفورمر شامل بخش کدگذار (Encoder) در سمت چپ و گشاینده (Decoder) در سمت راست. هر دو بخش از پشته‌ای از لایه‌های یکسان تشکیل شده‌اند که عمدتاً شامل مکانیزم توجه چندسر (Multi-Head Attention) و شبکه‌های عصبی پیش‌خور (Feed Forward) هستند. اتصالات باقیمانده (Add) و نرمال‌سازی لایه‌ای (Norm) نیز برای پایداری آموزش استفاده می‌شوند. گشاینده علاوه بر توجه خودی، از توجه متقابل (Cross-Attention) برای در نظر گرفتن خروجی کدگذار نیز بهره می‌برد. این معماری امکان پردازش موازی و مدل‌سازی وابستگی‌های بلندمدت را فراهم می‌کند. برگرفته و بازطراحی‌شده بر اساس **[attention]**.

### ۲-۲-۸ ماشین بردار پشتیبان (SVM)

ماشین بردار پشتیبان (Support Vector Machine, SVM) یکی از الگوریتم‌های یادگیری نظارت‌شده قدرتمند و پرکاربرد است که عمدتاً برای مسائل طبقه‌بندی (Classification) و همچنین رگرسیون (Regression) استفاده می‌شود. ایده اصلی SVM در مسائل طبقه‌بندی دودویی، یافتن یک ابرصفحه (Hyperplane) بهینه در فضای ویژگی‌هاست که بتواند داده‌های مربوط به دو کلاس مختلف را به بهترین شکل ممکن از یکدیگر جدا کند.

"بهترین شکل ممکن" در SVM به معنای یافتن ابرصفحه‌ای است که بیشترین حاشیه (Margin) را با نزدیک‌ترین نقاط داده از هر دو کلاس داشته باشد. این نزدیک‌ترین نقاط، که روی مرز حاشیه قرار می‌گیرند، بردارهای پشتیبان (Support Vectors) نامیده می‌شوند، زیرا موقعیت ابرصفحه جداکننده تنها به این نقاط بستگی دارد. هدف، حداکثر کردن این حاشیه است، زیرا تئوری یادگیری آماری نشان می‌دهد که جداکننده‌ای با حاشیه بزرگتر،

معمولاً خطای تعمیم کمتری دارد و عملکرد بهتری روی داده‌های دیده‌نشده خواهد داشت.

### ۲-۲-۸-۰-۱ انواع SVM

- **SVM خطی (Linear SVM):** زمانی که داده‌ها به صورت خطی قابل تفکیک باشند (یعنی بتوان با یک خط مستقیم یا یک صفحه صاف آن‌ها را جدا کرد)، از SVM خطی استفاده می‌شود.
- **SVM غیرخطی (Non-linear SVM):** در بسیاری از مسائل دنیای واقعی، داده‌ها به صورت خطی قابل تفکیک نیستند. در این موارد، SVM از ترفند کرنل (Kernel Trick) استفاده می‌کند. ترفند کرنل به SVM اجازه می‌دهد تا داده‌ها را به یک فضای ویژگی با ابعاد بالاتر نگاشت کند که در آن، داده‌ها ممکن است به صورت خطی قابل تفکیک شوند. سپس SVM خطی در آن فضای جدید اعمال می‌شود. توابع کرنل رایج شامل کرنل چندجمله‌ای (Polynomial)، کرنل تابع پایه شعاعی (Radial Basis Function, RBF) یا گاوسی، و کرنل سیگموئید (Sigmoid) هستند.

**۲-۲-۸-۰-۲ کاربرد در تشخیص بدافزار** SVM به دلیل عملکرد خوب در فضاهای با ابعاد بالا و مقاومت نسبی در برابر بیش‌برازش (Overfitting)، به‌طور گسترده‌ای در تشخیص بدافزار اندروید، به‌ویژه با استفاده از ویژگی‌های استاتیک (مانند مجوزها، فراخوانی‌های API) یا ویژگی‌های پویا (مانند توالی‌های رفتاری) استفاده شده است [**AndroidMalwareSurvey**، **Demontis۲۰۱۷**]. اغلب، ویژگی‌های استخراج‌شده از اپلیکیشن‌ها به عنوان ورودی به SVM داده می‌شوند تا مدل آموزش ببیند که آیا یک اپلیکیشن بدافزار است یا خیر.

### ۲-۲-۸-۰-۳ مزایا و معایب

- **مزایا:** کارایی بالا در فضاهای با ابعاد زیاد، مؤثر بودن در مواردی که تعداد ابعاد بیشتر از تعداد نمونه‌هاست، حافظه کارآمد (چون فقط از بردارهای پشتیبان استفاده می‌کند)، تطبیق‌پذیری با استفاده از کرنل‌های مختلف.
- **معایب:** عملکرد ضعیف در مجموعه داده‌های بسیار بزرگ (به دلیل پیچیدگی محاسباتی آموزش که می‌تواند بین $O(n^2)$ تا $O(n^3)$ باشد)، حساسیت به انتخاب تابع کرنل و پارامترهای آن (مانند پارامتر C یا هزینه خطا و پارامتر گاما در کرنل

،(RBF عدم ارائه مستقیم احتمالات تعلق به کلاس (اگرچه روش‌هایی برای تخمین آن وجود دارد).

اگرچه مدل‌های یادگیری عمیق مانند CNN و ترنسفورمرها در سال‌های اخیر توجه بیشتری را به خود جلب کرده‌اند، SVM همچنان یک ابزار قدرتمند و یک معیار (Baseline) مهم در حوزه تشخیص بدافزار محسوب می‌شود.

## ۲-۳ تکنیک‌های آموزشی و ارزیابی

در این بخش، تکنیک‌های کلیدی مورد استفاده برای آموزش مدل MAGNET و ارزیابی عملکرد آن تشریح می‌شود. این تکنیک‌ها برای اطمینان از قابلیت اطمینان، کارایی و تعمیم‌پذیری مدل ضروری هستند.

### ۲-۳-۱ اعتبارسنجی متقاطع (Cross-Validation)

اعتبارسنجی متقاطع (Cross-Validation) یک تکنیک آماری برای ارزیابی عملکرد مدل‌های یادگیری ماشین و تخمین میزان تعمیم‌پذیری آن‌ها به داده‌های جدید و مستقل است. این روش به کاهش واریانس تخمین عملکرد کمک کرده و از بیش‌برازش (Overfitting) بر روی یک تقسیم خاص از داده‌ها به مجموعه‌های آموزشی و آزمایشی جلوگیری می‌کند.

رایج‌ترین نوع اعتبارسنجی متقاطع، **اعتبارسنجی تایی-k** (k-fold Cross-Validation) است. در این روش:

۱. مجموعه داده اصلی به صورت تصادفی به k زیرمجموعه (یا "تا") با اندازه تقریباً مساوی تقسیم می‌شود.

۲. فرآیند آموزش و ارزیابی k بار تکرار می‌شود.

۳. در هر تکرار (i از ۱ تا k)، از k-1 زیرمجموعه برای آموزش مدل استفاده می‌شود و از زیرمجموعه باقی‌مانده (تا i_ام) به عنوان مجموعه اعتبارسنجی (Validation Set) برای ارزیابی مدل استفاده می‌شود.

۴. نتایج ارزیابی (مانند دقت، F1 Score) از هر k تکرار جمع‌آوری می‌شود.

۵. میانگین (و گاهی انحراف معیار) این نتایج به عنوان تخمین نهایی عملکرد مدل گزارش می‌شود.

در این پژوهش، از **اعتبارسنجی ۵-تایی (5-fold Cross-Validation)** استفاده شده است ($k = ۵$). این بدان معناست که داده‌ها به پنج بخش تقسیم شده و مدل پنج بار آموزش و ارزیابی می‌شود، هر بار با یک بخش متفاوت به عنوان داده اعتبارسنجی. این رویکرد نسبت به تقسیم ساده آموزشی/آزمایشی، تخمین پایدارتر و قابل اعتمادتری از عملکرد مدل ارائه می‌دهد، زیرا از تمام داده‌ها هم برای آموزش و هم برای ارزیابی استفاده می‌شود. در پیاده‌سازی کد، این فرآیند می‌تواند با استفاده از توابعی مانند KFold یا StratifiedKFold از کتابخانه scikit-learn و اجرای حلقه آموزش و ارزیابی (مانند تابع فرضی train_and_evaluate_magnet) در هر تقسیم انجام شود.

### ۲-۳-۲ مدیریت اورفیتینگ و آندرفیتینگ

دو چالش اساسی در آموزش مدل‌های یادگیری ماشین، به‌ویژه مدل‌های یادگیری عمیق با ظرفیت بالا، بیش‌برازش (Overfitting) و کم‌برازش (Underfitting) هستند.

- **کم‌برازش (Underfitting):** زمانی رخ می‌دهد که مدل به اندازه کافی پیچیده نیست یا به اندازه کافی آموزش ندیده است تا الگوهای اساسی موجود در داده‌های آموزشی را یاد بگیرد. در نتیجه، مدل هم بر روی داده‌های آموزشی و هم بر روی داده‌های جدید عملکرد ضعیفی دارد. برای مقابله با کم‌برازش، می‌توان پیچیدگی مدل را افزایش داد (مثلاً با افزودن لایه‌ها یا نورون‌ها)، ویژگی‌های بهتری مهندسی کرد، یا زمان آموزش را افزایش داد. در مدل MAGNET، تنظیم مناسب تعداد لایه‌ها (num_layers) و ابعاد نهان (embedding_dim) می‌تواند به جلوگیری از کم‌برازش کمک کند.
- **بیش‌برازش (Overfitting):** زمانی رخ می‌دهد که مدل بیش از حد به داده‌های آموزشی خاص، از جمله نویز و جزئیات تصادفی آن، وابسته می‌شود. در نتیجه، مدل بر روی داده‌های آموزشی عملکرد بسیار خوبی دارد، اما توانایی تعمیم به داده‌های جدید و دیده‌نشده را از دست می‌دهد و بر روی آن‌ها عملکرد ضعیفی نشان می‌دهد. برای مقابله با بیش‌برازش، از تکنیک‌های تنظیم‌گری (Regularization) استفاده می‌شود. در این پژوهش، از تکنیک‌های زیر استفاده شده است:

  – **Dropout:** در هر مرحله آموزش، به طور تصادفی برخی از نورون‌ها (و اتصالات آن‌ها) را با احتمال مشخصی (در اینجا با نرخ ۰.۲) غیرفعال می‌کند. این کار

باعث می‌شود شبکه به یک مسیر یا ویژگی خاص بیش از حد وابسته نشود و مدل مقاوم‌تری یاد بگیرد.

- **توقف زودهنگام (Early Stopping):** عملکرد مدل بر روی یک مجموعه اعتبارسنجی جداگانه در طول آموزش نظارت می‌شود. اگر عملکرد مدل روی مجموعه اعتبارسنجی برای تعداد مشخصی از دوره‌ها (Epochs) بهبود نیابد (در اینجا با صبر ۳ دوره)، آموزش متوقف می‌شود، حتی اگر عملکرد روی مجموعه آموزشی همچنان در حال بهبود باشد. این کار از ادامه آموزش و ورود مدل به فاز بیش‌برازش جلوگیری می‌کند.
- **تنظیم‌گری L۲ (Weight Decay):** (هرچند به صراحت در بخش بیش‌برازش ذکر نشده، اما در بخش بهینه‌سازی Adam با weight_decay=۰.۰۱ اشاره شده است). این روش با افزودن یک جمله جریمه به تابع هزینه که متناسب با مجذور اندازه وزن‌های مدل است، از بزرگ شدن بیش از حد وزن‌ها جلوگیری می‌کند و به مدل ساده‌تر و با تعمیم‌پذیری بهتر منجر می‌شود.

تحلیل نمودارهای تابع هزینه و معیارهای ارزیابی بر روی داده‌های آموزشی و اعتبارسنجی در طول زمان آموزش، ابزار مهمی برای تشخیص و مدیریت این دو پدیده است.

### ۲-۳-۳ روش‌های بهینه‌سازی

بهینه‌سازی فرآیند تنظیم پارامترهای مدل (مانند وزن‌ها و بایاس‌ها در شبکه‌های عصبی) به منظور کمینه کردن تابع هزینه (Loss Function) است. انتخاب الگوریتم بهینه‌سازی مناسب و تنظیم ابرپارامترهای (Hyperparameters) آن نقش کلیدی در سرعت همگرایی و کیفیت نهایی مدل دارد.

#### ۲-۳-۳-۱ الگوریتم Adam

الگوریتم Adam (Adaptive Moment Estimation) **[Kingma۲۰۱۵]** یکی از محبوب‌ترین و کارآمدترین الگوریتم‌های بهینه‌سازی مبتنی بر گرادیان نزولی تصادفی (Stochastic Gradient Descent, SGD) است که برای آموزش شبکه‌های عصبی عمیق استفاده می‌شود. Adam نرخ یادگیری (Learning Rate) را برای هر پارامتر به صورت تطبیقی تنظیم می‌کند. این کار را با استفاده از تخمین‌های مرتبه اول (میانگین یا مومنتوم) و مرتبه دوم (واریانس غیرمرکزی)

گرادیان‌ها انجام می‌دهد. به عبارت دیگر، Adam مزایای دو الگوریتم دیگر، یعنی AdaGrad (که نرخ یادگیری را بر اساس فراوانی به‌روزرسانی پارامتر تنظیم می‌کند) و RMSProp (که از میانگین متحرک نمایی مجذور گرادیان‌ها استفاده می‌کند) را با هم ترکیب می‌کند و همچنین از مومنتوم برای هموارسازی مسیر گرادیان و تسریع همگرایی بهره می‌برد. در این پژوهش، Adam با نرخ یادگیری پیش‌فرض (معمولاً ۰.۰۰۱) و ضریب تنظیم‌گری L2 (weight_decay) برابر با ۰.۰۱ در تابع train_and_evaluate_magnet برای به‌روزرسانی وزن‌های مدل MAGNET به کار رفته است. Adam به دلیل سرعت همگرایی بالا، نیاز کمتر به تنظیم دقیق نرخ یادگیری اولیه، و عملکرد خوب در مسائل با گرادیان‌های پراکنده یا نویزی، انتخاب مناسبی برای آموزش مدل‌های پیچیده مانند ترنسفورمر و GNN بر روی داده‌های چندوجهی بوده است.

### ۲-۳-۳-۲ الگوریتم PIRATES

الگوریتم PIRATES[1] یکی از روش‌های نوین بهینه‌سازی الهام‌گرفته از طبیعت است که با الهام از رفتار جمعی دزدان دریایی در جستجوی گنج طراحی شده است. در این الگوریتم، هر راه‌حل بالقوه به عنوان یک «کشتی» در فضای جستجو مدل می‌شود و مجموعه‌ای از کشتی‌ها (جمعیت) به طور موازی و با استفاده از اطلاعات فردی (تجربیات گذشته)، جمعی (اطلاعات بهترین اعضا)، و تعاملات خاص (مانند نبرد و طوفان)، به سمت نقاط بهینه حرکت می‌کنند. این الگوریتم با ترکیب ایده‌هایی از الگوریتم‌های ازدحامی (مانند PSO) و تکاملی، سعی در افزایش پایداری، سرعت همگرایی و جلوگیری از گیر افتادن در بهینه‌های محلی دارد.

اجزای کلیدی این الگوریتم عبارت‌اند از:

- **کشتی‌ها (Ships):** هر کشتی نمایانگر یک نقطه در فضای جستجو است و موقعیت و سرعت مخصوص به خود را دارد.
- **رهبر (Leader):** کشتی با بهترین عملکرد (کمترین مقدار تابع هدف) در هر تکرار به عنوان رهبر انتخاب می‌شود و سایر کشتی‌ها از آن پیروی می‌کنند.
- **نقشه جمعی و خصوصی:** هر کشتی علاوه بر بهترین موقعیت شخصی، از نقشه‌ای جمعی (شامل بهترین موقعیت‌های سایر کشتی‌ها) نیز بهره می‌برد.

[1] Pirate-Inspired Robust Adaptive Trajectory Exploration Strategy

- **کشتی‌های برتر (Top Ships):** تعدادی از بهترین کشتی‌ها به عنوان مرجع برای سایر اعضا عمل می‌کنند.
- **مکانیزم‌های تنوع‌بخش:** الگوریتم با استفاده از نبرد بین کشتی‌های برتر و وقوع طوفان‌های تصادفی، از همگرایی زودهنگام و گیر افتادن در بهینه‌های محلی جلوگیری می‌کند.

الگوریتم PIRATES با بهره‌گیری از چندین منبع اطلاعاتی و تنظیم پویا، قادر است به سرعت به نقاط بهینه همگرا شود و در عین حال تنوع جمعیت را حفظ کند. این ویژگی‌ها، PIRATES را به گزینه‌ای مناسب برای بهینه‌سازی مسائل پیچیده، از جمله تنظیم خودکار هایپرپارامترهای مدل‌های یادگیری عمیق و چندوجهی، تبدیل کرده است. در پژوهش حاضر، از این الگوریتم برای جستجوی بهینه پارامترهای مدل پیشنهادی (MAGNET) استفاده شده است تا عملکرد مدل در تشخیص بدافزارهای اندرویدی بهبود یابد.

استفاده از الگوریتم‌های بهینه‌سازی الهام‌گرفته از طبیعت، به ویژه PIRATES، در سال‌های اخیر به عنوان رویکردی مؤثر برای حل مسائل بهینه‌سازی غیرخطی و پیچیده در حوزه‌های مختلف، از جمله امنیت سایبری و یادگیری ماشین، مطرح شده است. در این پژوهش، PIRATES به عنوان یکی از ابزارهای کلیدی برای تنظیم بهینه پارامترهای مدل چندوجهی مبتنی بر ترنسفورمر و شبکه‌های عصبی گرافی به کار رفته است.

### ۲-۳-۳-۳ الگوریتم Optuna

Optuna **[Optuna۲۰۱۹]** یک چارچوب (Framework) نرم‌افزاری متن‌باز و قدرتمند برای بهینه‌سازی خودکار هایپرپارامترها است. Optuna از الگوریتم‌های جستجوی متنوعی پشتیبانی می‌کند، از جمله جستجوی تصادفی (Random Search)، جستجوی شبکه‌ای (Grid Search)، و الگوریتم‌های پیشرفته‌تر مبتنی بر مدل مانند Tree-structured Parzen Estimator (TPE) که نوعی بهینه‌سازی بیزی است. یکی از ویژگی‌های کلیدی Optuna، قابلیت هرس (Pruning) آزمایش‌های ناموفق است. با استفاده از الگوریتم‌های هرس (مانند Median Pruning یا Successive Halving)، Optuna می‌تواند آزمایش‌هایی را که در مراحل اولیه عملکرد ضعیفی دارند، متوقف کرده و منابع محاسباتی را بر روی آزمایش‌های امیدوارکننده‌تر متمرکز کند.

در این پژوهش، ادعا شده که از Optuna برای تنظیم پارامترهایی مانند نرخ Dropout (که مقدار بهینه ۲.۰ یافت شده) و ابعاد نهان (embedding_dim) استفاده شده است. این فرآیند معمولاً شامل تعریف یک تابع هدف (مانند تابع train_and_evaluate_magnet که یک معیار

عملکرد مثل F1 Score یا دقت را برمی‌گرداند) و مشخص کردن فضای جستجو برای هر هایپرپارامتر است. سپس Optuna با اجرای مکرر تابع هدف با مقادیر مختلف هایپرپارامترها (که با استفاده از نمونه‌برداری تطبیقی انتخاب می‌شوند)، به تدریج به سمت یافتن ترکیب بهینه حرکت می‌کند. استفاده از Optuna می‌تواند فرآیند زمان‌بر و خسته‌کننده تنظیم دستی هایپرپارامترها را خودکار کرده و به یافتن مدل‌هایی با عملکرد بهتر کمک کند.

## ۲-۳-۴ معیارهای ارزیابی عملکرد

برای ارزیابی کمی و کیفی عملکرد مدل طبقه‌بندی MAGNET در تشخیص بدافزارهای اندرویدی، از معیارهای استاندارد مختلفی استفاده شده است. انتخاب معیار مناسب به ماهیت مسئله و توزیع کلاس‌ها در مجموعه داده بستگی دارد.

### ۲-۳-۴-۱ دقت (Accuracy)

دقت (Accuracy) ساده‌ترین و رایج‌ترین معیار ارزیابی است که نسبت کل نمونه‌هایی که به درستی توسط مدل طبقه‌بندی شده‌اند (چه مثبت و چه منفی) به تعداد کل نمونه‌ها را اندازه‌گیری می‌کند:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

که در آن:

- TP (True Positive): تعداد نمونه‌های بدافزار که به درستی بدافزار تشخیص داده شده‌اند.
- TN (True Negative): تعداد نمونه‌های سالم که به درستی سالم تشخیص داده شده‌اند.
- FP (False Positive): تعداد نمونه‌های سالم که به اشتباه بدافزار تشخیص داده شده‌اند (خطای نوع اول).
- FN (False Negative): تعداد نمونه‌های بدافزار که به اشتباه سالم تشخیص داده شده‌اند (خطای نوع دوم).

در این پژوهش، دقت مدل MAGNET در تست نهایی ۲۴.۹۷ درصد گزارش شده است. دقت معیار خوبی است زمانی که کلاس‌ها تقریباً متوازن باشند. با این حال، در مجموعه‌های

داده نامتوازن (مثلاً اگر تعداد نمونه‌های سالم بسیار بیشتر از بدافزارها باشد)، دقت بالا ممکن است گمراه‌کننده باشد.

### ۲-۴-۳-۲ Score ۱F

امتیاز F1 (F1 Score) میانگین هارمونیک دو معیار دیگر، یعنی دقت (Precision) و یادآوری (Recall) است و به ویژه در مسائل با کلاس‌های نامتوازن مفید است.

- **دقت (Precision):** نسبت نمونه‌هایی که به درستی مثبت پیش‌بینی شده‌اند به کل نمونه‌هایی که مثبت پیش‌بینی شده‌اند. این معیار نشان می‌دهد که چقدر می‌توان به پیش‌بینی‌های مثبت مدل اعتماد کرد.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **یادآوری (Recall) یا حساسیت (Sensitivity) یا نرخ مثبت واقعی (TPR):** نسبت نمونه‌هایی که به درستی مثبت پیش‌بینی شده‌اند به کل نمونه‌های واقعاً مثبت. این معیار نشان می‌دهد که مدل چه کسری از نمونه‌های مثبت واقعی را توانسته شناسایی کند.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

امتیاز F1 به صورت زیر محاسبه می‌شود:

$$\text{Score } ۱\text{F} = ۲ \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{۲ \times \text{TP}}{۲ \times \text{TP} + \text{FP} + \text{FN}}$$

امتیاز F1 تعادلی بین دقت و یادآوری برقرار می‌کند و مقدار بالای آن نشان می‌دهد که مدل هم در کاهش هشدارهای کاذب (FP بالا باعث کاهش Precision می‌شود) و هم در شناسایی نمونه‌های مثبت واقعی (FN بالا باعث کاهش Recall می‌شود) موفق بوده است. در این پژوهش، مقدار F1 Score ۰.۹۸۲۳ در بهترین حالت به دست آمده که نشان‌دهنده عملکرد بسیار خوب مدل در هر دو جنبه است. این معیار معمولاً برای ارزیابی عملکرد در تشخیص بدافزار ارجحیت دارد، زیرا هزینه تشخیص نادرست (FN یا FP) می‌تواند بالا باشد.

### ۲-۴-۳-۳ AUC (Area Under the ROC Curve)

منحنی مشخصه عملکرد گیرنده (Receiver Operating Characteristic, ROC) نموداری است که نرخ مثبت واقعی (True Positive Rate, TPR یا همان Recall) را در مقابل نرخ مثبت کاذب (False Positive Rate, FPR) در آستانه‌های طبقه‌بندی مختلف رسم می‌کند.

$$FPR = \frac{FP}{FP + TN}$$

مساحت زیر این منحنی (Area Under the Curve, AUC) یک معیار واحد است که توانایی کلی مدل در تمایز بین کلاس‌های مثبت و منفی را در تمام آستانه‌های ممکن اندازه‌گیری می‌کند. مقدار AUC بین ۰ و ۱ متغیر است.

- AUC = 1: طبقه‌بندی کامل و بی‌نقص.
- AUC = 0.5: عملکرد تصادفی (مانند پرتاب سکه).
- AUC < 0.5: عملکرد بدتر از تصادفی.

AUC یک معیار مفید است زیرا نسبت به تغییرات آستانه طبقه‌بندی و همچنین نسبت به عدم توازن کلاس‌ها (تا حدی) مقاوم است. هرچند مقدار دقیق AUC در متن گزارش نشده، اما با توجه به F1 Score بالا (۰.۹۸۲۳)، انتظار می‌رود مقدار AUC نیز بسیار بالا و نزدیک به ۱ (احتمالاً حدود ۰.۹۸ یا بیشتر) باشد، که نشان‌دهنده قدرت تمایز عالی مدل MAGNET است.

**۲-۳-۴-۳-۱ جمع‌بندی ارزیابی** استفاده ترکیبی از این معیارها (دقت، F1 Score، AUC) به همراه اعتبارسنجی متقاطع، تصویری جامع و قابل اعتماد از عملکرد مدل MAGNET ارائه می‌دهد و امکان مقایسه معنادار آن با سایر روش‌ها را فراهم می‌کند. مدیریت دقیق بیش‌برازش و کم‌برازش و استفاده از روش‌های بهینه‌سازی مناسب نیز به دستیابی به این نتایج کمک کرده‌اند.

## ۲-۴ مروری بر مطالعات پیشین

در این بخش، پژوهش‌های پیشین در حوزه تشخیص بدافزارهای اندرویدی با تمرکز بر رویکردهای اصلی تحلیل ایستا و پویا مرور می‌شوند. همچنین، به تاریخچه مختصری از

تلاش‌ها در این زمینه اشاره می‌شود.

### ۲-۴-۱ روش‌های تحلیل ایستا

روش‌های تحلیل ایستا (Static Analysis) به بررسی و تحلیل کد و ساختار اپلیکیشن‌های اندرویدی بدون نیاز به اجرای آن‌ها می‌پردازند. این روش‌ها معمولاً سریع‌تر از تحلیل پویا هستند و می‌توانند پوشش کاملی از کد برنامه را فراهم کنند. ویژگی‌های استخراج‌شده از تحلیل ایستا اغلب به عنوان ورودی برای مدل‌های یادگیری ماشین استفاده می‌شوند [**AndroidMalwareSurvey**].

#### ۲-۴-۱-۱ تحلیل ویژگی‌های مبتنی بر مانیفست و متاداده

فایل AndroidManifest.xml حاوی اطلاعات مهمی درباره ساختار و نیازمندی‌های اپلیکیشن است.

- **تحلیل مجوزها (Permissions):** یکی از پرکاربردترین روش‌های ایستا، تحلیل الگوهای مجوزهای درخواست‌شده توسط اپلیکیشن است. بدافزارها اغلب مجوزهای خطرناک و غیرضروری بیشتری نسبت به کاربرد ظاهری خود درخواست می‌کنند. پژوهش‌های متعددی مانند Drebin [**Drebin**] نشان داده‌اند که ترکیب مجوزها با سایر ویژگی‌های ایستا می‌تواند در تشخیص بادت‌افزار مؤثر باشد. مدل‌های یادگیری ماشین مانند SVM، درخت تصمیم و بیز ساده اغلب برای طبقه‌بندی بر اساس این ویژگی‌ها استفاده شده‌اند.

- **اجزای برنامه (Components):** تعداد و نوع اجزای تعریف‌شده در مانیفست (مانند Activities, Services, Receivers, Providers) و همچنین فیلترهای Intent مرتبط با آن‌ها نیز می‌تواند اطلاعات مفیدی در اختیار قرار دهد.

#### ۲-۴-۱-۲ تحلیل کد (Code Analysis)

این روش‌ها به بررسی خود کد برنامه (بایت‌کد Dalvik/ART یا کد Native) می‌پردازند.

- **تحلیل فراخوانی‌های API:** شناسایی و تحلیل فراخوانی‌های API حساس (مانند APIهای مربوط به ارسال پیامک، دسترسی به موقعیت مکانی، یا استفاده از توابع رمزنگاری) یکی دیگر از رویکردهای رایج است. الگوها یا توالی‌های خاصی از

فراخوانی‌های API می‌توانند نشان‌دهنده رفتار مخرب باشند [**۲۰۱۴Yuan**]. این ویژگی‌ها می‌توانند به صورت باینری (وجود یا عدم وجود فراخوانی) یا به صورت فراوانی استفاده شوند.

- **تحلیل جریان داده و کنترل (Data/Control Flow Analysis):** روش‌های پیشرفته‌تر تحلیل ایستا، گراف جریان کنترل (CFG) یا گراف جریان داده (DFG) برنامه را ساخته و تحلیل می‌کنند تا مسیرهای اجرای بالقوه و نحوه انتشار داده‌های حساس را ردیابی کنند. تکنیک‌هایی مانند Taint Analysis ایستا می‌توانند برای شناسایی نشت اطلاعات خصوصی به کار روند.

- **ویژگی‌های مبتنی بر Opcode:** تحلیل توالی‌ها یا فراوانی کدهای عملیاتی (Opcodes) در بایت‌کد نیز به عنوان ویژگی برای مدل‌های یادگیری ماشین استفاده شده است.

### ۲-۴-۱-۳ تحلیل ایستای ساختاری

- **گراف فراخوانی (Call Graph):** ساخت و تحلیل گراف فراخوانی توابع، که روابط فراخوانی بین متدهای مختلف برنامه را نشان می‌دهد، می‌تواند به شناسایی الگوهای ساختاری مرتبط با بدافزار کمک کند. شبکه‌های عصبی گرافی (GNN) برای تحلیل این گراف‌ها مناسب هستند [**۲۰۱۸Zhu**].

**۲-۴-۱-۳-۱ محدودیت‌های تحلیل ایستا** با وجود مزایا، تحلیل ایستا با چالش‌هایی نیز روبروست:

- **ابهام‌سازی (Obfuscation):** بدافزارها اغلب از تکنیک‌های ابهام‌سازی کد (مانند تغییر نام متغیرها و توابع، درج کدهای زائد، رمزنگاری رشته‌ها) برای دشوار کردن تحلیل ایستا استفاده می‌کنند.

- **بارگذاری پویای کد (Dynamic Code Loading):** برخی بدافزارها بخش‌هایی از کد مخرب خود را در زمان اجرا از منابع خارجی دانلود و اجرا می‌کنند که این بخش‌ها در تحلیل ایستا قابل مشاهده نیستند.

- **کد Native:** تحلیل کدهای نوشته شده به زبان‌های Native (مانند C/C++) که از طریق JNI فراخوانی می‌شوند، پیچیده‌تر از تحلیل بایت‌کد جاوا/کاتلین است.

### ۲-۴-۲ روش‌های تحلیل پویا

روش‌های تحلیل پویا (Dynamic Analysis) یا آنالیز رفتاری، اپلیکیشن را در یک محیط کنترل‌شده (مانند شبیه‌ساز، دستگاه واقعی یا جعبه شنی Sandbox) اجرا کرده و رفتار آن را در زمان اجرا مشاهده و ثبت می‌کنند. این روش‌ها می‌توانند بر محدودیت‌های تحلیل ایستا در مواجهه با ابهام‌سازی و بارگذاری پویای کد غلبه کنند، زیرا رفتار واقعی برنامه را بررسی می‌کنند [**AndroidMalwareSurvey**].

#### ۲-۴-۲-۱ مانیتورینگ رفتار سیستم

این روش‌ها فعالیت‌های اپلیکیشن در سطح سیستم‌عامل را رصد می‌کنند.

- **ردیابی فراخوانی‌های سیستمی (System Call Tracing):** ثبت و تحلیل توالی فراخوانی‌های سیستمی که برنامه انجام می‌دهد. الگوهای خاصی از این فراخوانی‌ها می‌توانند نشان‌دهنده فعالیت مخرب باشند.
- **تحلیل Taint پویا (Dynamic Taint Analysis):** ابزارهایی مانند TaintDroid [**TaintDroid**] داده‌های حساس (منابع Taint) را مشخص کرده و نحوه انتشار آن‌ها در طول اجرای برنامه را ردیابی می‌کنند تا نشت اطلاعات به مقاصد غیرمجاز (سینک‌های Taint) را شناسایی کنند.
- **مانیتورینگ تغییرات فایل‌سیستم و رجیستری (در محیط ویندوز):** ثبت هرگونه ایجاد، حذف یا تغییر فایل‌ها یا تنظیمات سیستمی.

#### ۲-۴-۲-۲ تحلیل شبکه

بسیاری از بدافزارها برای ارتباط با سرور فرماندهی و کنترل (C&C)، ارسال داده‌های سرقت‌شده یا دانلود کدهای مخرب بیشتر، از شبکه استفاده می‌کنند. تحلیل ترافیک شبکه اپلیکیشن می‌تواند الگوهای مشکوکی مانند اتصال به IPها یا دامنه‌های شناخته‌شده مخرب، استفاده از پروتکل‌های غیرمعمول، یا حجم بالای ترافیک خروجی را آشکار کند [**NetworkAnalysis**].

### ۲-۴-۲-۳ بررسی مصرف منابع

الگوهای غیرعادی در مصرف منابع سیستم مانند CPU، حافظه، باتری یا پهنای باند شبکه نیز می‌تواند نشانه‌ای از فعالیت بدافزاری باشد (مثلاً استخراج رمزارز یا اجرای حملات DoS).

**۲-۴-۲-۳-۱ ابزارها و محیط‌های تحلیل پویا** ابزارها و پلتفرم‌های مختلفی برای تحلیل پویای بدافزارهای اندرویدی توسعه یافته‌اند، از جمله DroidBox، AndroPyTool، Mobile Sandbox و پلتفرم‌های تجاری مانند Joe Sandbox Mobile. این ابزارها معمولاً اجرای برنامه را در یک شبیه‌ساز یا دستگاه روت‌شده خودکار کرده و گزارش جامعی از رفتارهای مشاهده‌شده تولید می‌کنند.

### ۲-۴-۲-۳-۲ محدودیت‌های تحلیل پویا

- **پوشش محدود کد (Limited Code Coverage):** تحلیل پویا تنها مسیرهای اجرایی را بررسی می‌کند که در طول اجرای خاص فعال شده‌اند. بدافزارها ممکن است شامل کدهای مخربی باشند که تنها تحت شرایط خاصی (مثلاً در تاریخ معین یا با دریافت دستور خاص) فعال می‌شوند و در طول تحلیل مشاهده نشوند.
- **زمان‌بر بودن:** اجرای هر اپلیکیشن و ثبت رفتارهای آن می‌تواند زمان‌بر باشد، که تحلیل مجموعه داده‌های بزرگ را دشوار می‌کند.
- **تشخیص محیط تحلیل (Environment Detection):** بدافزارهای پیشرفته ممکن است تلاش کنند تا تشخیص دهند که آیا در یک محیط تحلیل (مانند شبیه‌ساز یا جعبه شنی) اجرا می‌شوند یا خیر و در این صورت، رفتار مخرب خود را پنهان کنند.
- **نیاز به منابع:** نیاز به محیط‌های اجرایی ایزوله و ابزارهای مانیتورینگ دارد.

### ۲-۴-۲-۴ روش‌های ترکیبی (Hybrid Approaches)

برای بهره‌مندی از مزایای هر دو روش و غلبه بر محدودیت‌های آن‌ها، بسیاری از سیستم‌های تشخیص بدافزار مدرن از رویکردهای ترکیبی استفاده می‌کنند که ویژگی‌های استخراج‌شده از تحلیل ایستا و پویا را با هم ترکیب کرده و به عنوان ورودی به مدل‌های یادگیری ماشین یا یادگیری عمیق می‌دهند [**DeepLearningMalware**]. مدل MAGNET

در این پژوهش نیز با استفاده از داده‌های چندوجهی (که می‌توانند شامل ویژگی‌های ایستا و پویا باشند) در این دسته قرار می‌گیرد.

### ۲-۴-۳ کارهای پیشین و تاریخچه مختصر

تلاش‌ها برای شناسایی و مقابله با بدافزارها تاریخچه‌ای طولانی دارد. از اولین برنامه‌های آنتی‌ویروس مانند Flushot Plus (۱۹۸۷) و اسکنر ویروس مک‌آفی (۱۹۸۹) که عمدتاً مبتنی بر امضا بودند، تا سیستم‌های تشخیص ناهنجاری اولیه مبتنی بر آمار مانند IDES (Denning، ۱۹۸۷) و W&S (WIDOM) در آزمایشگاه ملی لس‌آلاموس (۱۹۸۹).

با پیچیده‌تر شدن بدافزارها، روش‌های مبتنی بر یادگیری ماشین و سپس یادگیری عمیق اهمیت بیشتری یافتند. در اوایل دهه ۲۰۱۰، تمرکز زیادی بر استخراج ویژگی‌های ایستا (مانند مجوزها در Drebin [**Drebin**]) و استفاده از طبقه‌بندهای کلاسیک مانند SVM بود. سپس، روش‌های مبتنی بر تحلیل پویا (مانند TaintDroid [**TaintDroid**]) و تحلیل رفتارهای سیستمی و شبکه‌ای مطرح شدند.

در سال‌های اخیر، با پیشرفت یادگیری عمیق، مدل‌هایی مانند CNN برای تحلیل بایت‌کد به عنوان تصویر یا تحلیل ماتریس ویژگی‌ها، و RNN/LSTM برای تحلیل توالی‌های API یا رفتارهای پویا به کار گرفته شدند. همچنین، GNNها برای تحلیل ساختارهای گرافی مانند گراف فراخوانی مورد توجه قرار گرفتند.

تحقیقاتی مانند کار **ZhangNix۲۰۱۷** (استفاده از CNN روی فراخوانی‌های API) و **Vinayakumar۲۰۱۹** (استفاده از شبکه‌های عصبی عمیق) نتایج امیدوارکننده‌ای با دقت‌های بالا (گاهی بالای ۹۱٪ روی دیتاست‌های خاص) نشان دادند، اگرچه چالش‌هایی مانند تعمیم‌پذیری به بدافزارهای جدید و مقاومت در برابر حملات فرار همچنان وجود دارند [**Demontis۲۰۱۷**].

رویکردهای چندوجهی مانند مدل پیشنهادی MAGNET که سعی در ترکیب اطلاعات از منابع مختلف (ایستا، پویا، ساختاری، ترتیبی) با استفاده از معماری‌های پیشرفته مانند ترنسفورمر دارند، گام بعدی در جهت افزایش دقت و استحکام سیستم‌های تشخیص بدافزار اندروید محسوب می‌شوند.

## ۲-۵ جمع‌بندی فصل

در این فصل، مبانی نظری و پیشینه تحقیقاتی لازم برای درک پژوهش حاضر در زمینه تشخیص بدافزارهای اندرویدی ارائه گردید. ابتدا، مفاهیم پایه‌ای شامل انواع بدافزارها (باج‌افزار، تروجان، جاسوس‌افزار، تبلیغ‌افزار)، اجزای کلیدی اپلیکیشن‌های اندرویدی (مجوزها، فایل APK، سورس‌کد) و نقش داده‌های چندوجهی (جدولی، گرافی، ترتیبی) تشریح شد. سپس، مروری بر تکامل روش‌های یادگیری ماشین و معرفی معماری‌های کلیدی یادگیری عمیق (CNN، RNN/LSTM، GNN، ترنسفورمر) و الگوریتم کلاسیک SVM ارائه شد که در این حوزه کاربرد فراوان دارند.

در ادامه، تکنیک‌های ضروری برای آموزش و ارزیابی مدل‌ها، از جمله اعتبارسنجی متقاطع برای تخمین قابل اعتماد عملکرد، روش‌های مدیریت بیش‌برازش و کم‌برازش (مانند Dropout و توقف زودهنگام)، الگوریتم‌های بهینه‌سازی (مانند Adam) و ابزارهای بهینه‌سازی هایپرپارامتر (مانند Optuna)، و نهایتاً معیارهای استاندارد ارزیابی (دقت، F1 Score، AUC) مورد بحث قرار گرفتند.

بخش مرور مطالعات پیشین، به بررسی جامع روش‌های تحلیل ایستا (مبتنی بر مانیفست، کد و ساختار) و تحلیل پویا (مبتنی بر رفتار سیستم، شبکه و منابع) پرداخت و نقاط قوت و ضعف هر یک را برشمرد. همچنین، به تاریخچه مختصری از تلاش‌ها در این زمینه و اهمیت رویکردهای ترکیبی و یادگیری عمیق در سال‌های اخیر اشاره شد.

این فصل با فراهم آوردن درک عمیقی از چالش‌ها، مفاهیم، ابزارها و رویکردهای موجود در تشخیص بدافزار اندروید، زمینه را برای معرفی و ارزیابی روش پیشنهادی این پژوهش، مدل MAGNET، در فصل‌های آتی آماده می‌سازد.

# فصل سوم:
# روش پیشنهادی

## ۳-۱ روش پیشنهادی

با توجه به رشد روزافزون استفاده از سیستم‌عامل اندروید (؟) و در نتیجه، افزایش تهدیدات بدافزاری متوجه این پلتفرم (؟؟)، نیاز به روش‌های کارآمد و دقیق برای شناسایی بدافزارها بیش از پیش احساس می‌شود. این فصل به تشریح دقیق روش پیشنهادی این پژوهش، موسوم به **MAGNET** (تحلیل چندوجهی برای شناسایی تهدیدات مبتنی بر گراف و شبکه)، اختصاص دارد. این رویکرد با بهره‌گیری از داده‌های چندوجهی—جدولی، گرافی و ترتیبی—و ترکیب آن با معماری‌های پیشرفته یادگیری عمیق از جمله ترنسفورمرها و شبکه‌های عصبی گرافی (GNN)، محدودیت‌های روش‌های تحلیل ایستا و پویا را برطرف می‌کند. هدف اصلی، افزایش دقت شناسایی و تعمیم‌پذیری، به‌ویژه در مواجهه با تهدیدات روز صفر (Zero-Day) است. این بخش به صورت زیر سازمان‌دهی شده است: مقدمه‌ای بر روش پیشنهادی، فرضیات و ابزارهای محاسباتی، روش‌شناسی دقیق (شامل پیش‌پردازش داده‌ها، طراحی مدل، آموزش، بهینه‌سازی ابرپارامترها و ارزیابی) و جمع‌بندی نهایی. تمامی فرضیات، ابزارها، معادلات و شبه‌کدها به طور کامل مستند شده‌اند تا امکان تکرارپذیری فراهم باشد.

### ۳-۱-۱ مقدمه روش پیشنهادی

شناسایی بدافزار اندروید با چالش‌های مهمی روبروست که ناشی از محدودیت‌های رویکردهای سنتی است. روش‌های تحلیل ایستا، مانند بررسی کد منبع یا تحلیل مجوزها، اغلب در شناسایی بدافزارهای پیچیده که از تکنیک‌های مبهم‌سازی یا رمزنگاری استفاده می‌کنند، ناکام می‌مانند [**Drebin**]. این روش‌ها قادر به ثبت رفتارهای زمان اجرا یا سازگاری‌های پویا در برنامه‌های مخرب نیستند. در مقابل، تحلیل پویا که رفتار برنامه را در حین اجرا پایش می‌کند، محاسبات سنگینی دارد، زمان‌بر است و به دلیل عدم پوشش کامل مسیرهای اجرایی، پوشش کد ناقصی ارائه می‌دهد [**mobile۲۰۱۳spreitzenbarth**]. این کاستی‌ها به‌ویژه در مواجهه با بدافزارهای روز صفر—تهدیداتی با الگوهای ناشناخته که از سیستم‌های مبتنی بر امضا یا قوانین فرار می‌کنند—بیشتر مشهود است.

برای غلبه بر این چالش‌ها، ما ادغام داده‌های چندوجهی را پیشنهاد می‌کنیم تا دیدی جامع از ویژگی‌های برنامه ارائه شود. به طور خاص، از داده‌های زیر استفاده شده است:

- **داده‌های جدولی**: ویژگی‌های ایستا مانند مجوزها، تعداد فایل‌ها و اندازه برنامه که

بینشی از خصوصیات ساختاری ارائه می‌دهند.

- **داده‌های گرافی**: گراف‌های فراخوانی توابع که روابط ساختاری بین توابع را نمایش می‌دهند و وابستگی‌های داخلی برنامه را ثبت می‌کنند.
- **داده‌های ترتیبی**: توالی‌های فراخوانی API که الگوهای رفتاری زمانی در حین اجرا را منعکس می‌کنند.

این رویکرد چندوجهی، استخراج اطلاعات مکمل را تضمین می‌کند و خلأهای تحلیل‌های تک‌وجهی را پر می‌کند. افزون بر این، از معماری‌های پیشرفته یادگیری عمیق—ترنسفورمرها و GNN ها—برای مدل‌سازی الگوهای پیچیده در داخل و بین این وجه‌ها استفاده شده است. ترنسفورمرها در ثبت وابستگی‌های بلندمدت در داده‌های ترتیبی و جدولی برتری دارند، در حالی که هاGNN داده‌های گرافی را با بهره‌گیری از روابط گره‌ها و یال‌ها به طور مؤثر پردازش می‌کنند [**attention**، **Kipf۲۰۱۷**].

نوآوری اصلی این پژوهش در مدل **MAGNET** نهفته است که سه ماژول تخصصی *EnhancedTabTransformer*، *GraphTransformer* و *SequenceTrans-former*را با یک مکانیزم توجه پویا و لایه ادغام چندوجهی ترکیب می‌کند. مکانیزم توجه پویا به طور تطبیقی اهمیت ویژگی‌ها را در هر وجه تنظیم می‌کند، در حالی که لایه ادغام، بازنمایی‌های چندوجهی را به یک فضای ویژگی منسجم برای طبقه‌بندی تبدیل می‌کند. این طراحی، دقت شناسایی، پایداری و تطبیق‌پذیری در برابر تهدیدات در حال تحول را بهبود می‌بخشد و MAGNET را به پیشرفتی چشمگیر نسبت به روش‌های موجود تبدیل می‌کند.

### ۳-۱-۲ فرضیات و ابزارهای محاسباتی

#### ۳-۱-۲-۱ فرضیات

طراحی و ارزیابی MAGNET بر اساس فرضیات زیر استوار است:

۱. **ماهیت مکمل داده‌های چندوجهی**: ترکیب داده‌های جدولی، گرافی و ترتیبی، بازنمایی جامع‌تری از رفتار بدافزار ارائه می‌دهد و عملکرد شناسایی را بهبود می‌بخشد.

۲. **مناسب بودن معماری‌های پیشرفته**: ترنسفورمرها و هاGNN برای استخراج الگوهای پیچیده از داده‌های ساختاریافته و ترتیبی مناسب هستند.

۳. **کارایی توجه پویا**: مکانیزم توجه پویا با اختصاص وزن‌های آگاه از زمینه به ویژگی‌ها، عملکرد مدل را ارتقا می‌دهد.

۴. **بهینه‌سازی مؤثر**: استفاده از بهینه‌ساز Adam برای به‌روزرسانی وزن‌ها و Optuna برای تنظیم ابرپارامترها، همگرایی کارآمد و پیکربندی بهینه مدل را تضمین می‌کند.

### ۳-۱-۲-۲ ابزارهای محاسباتی

پیاده‌سازی و ارزیابی MAGNET با استفاده از ابزارها و منابع زیر انجام شد:

- **زبان برنامه‌نویسی**: Python 3.8.5، به دلیل اکوسیستم گسترده محاسبات علمی.
- **کتابخانه‌ها**:
  - PyTorch 1.9.0: برای ساخت و آموزش مدل‌های یادگیری عمیق.
  - PyTorch Geometric 1.7.0: برای پردازش داده‌های گرافی.
  - Scikit-learn 0.24.2: برای محاسبه معیارهای ارزیابی.
  - Optuna 2.10.0: برای بهینه‌سازی ابرپارامترها.
- **سخت‌افزار**:
  - GPU: NVIDIA RTX 3090 با ۲۴ گیگابایت VRAM و ۴۹۶,۱۰ هسته CUDA، برای محاسبات موازی کارآمد.
  - CPU: Intel Xeon E5-2690 v4 با ۳۲ هسته و فرکانس ۶.۲ گیگاهرتز، برای پشتیبانی از پیش‌پردازش و بهینه‌سازی.
  - RAM: ۱۲۸ گیگابایت DDR4، برای مدیریت داده‌های مقیاس بزرگ.
- **دیتاست**: دیتاست DREBIN [**Drebin**]، شامل ۵۶۰,۵ نمونه بدافزار و ۰۰۰,۵ نمونه سالم. این دیتاست شامل:
  - ویژگی‌های جدولی: مانند تعداد مجوزها (میانگین = ۱۲.۳، انحراف معیار = ۴.۱)، اندازه فایل (میانگین = ۲.۸ مگابایت، انحراف معیار = ۱.۹ مگابایت).
  - ویژگی‌های گرافی: گراف‌های فراخوانی توابع با میانگین ۲۴۵,۱ گره و ۸۷۲,۳ یال در هر نمونه.

– ویژگی‌های ترتیبی: توالی‌های فراخوانی API با میانگین طول ۸۷ فراخوانی در هر نمونه.

### ۳-۱-۳ روش‌شناسی

#### ۳-۱-۳-۱ پیش‌پردازش داده‌ها

دیتاست DREBIN برای سازگاری با مدل MAGNET پیش‌پردازش شد. هر وجه به صورت زیر پردازش شد:

- **داده‌های جدولی**:

– *استخراج ویژگی*: ۱۲۸ ویژگی ایستا استخراج شد، مانند تعداد مجوزها، تعداد فایل‌ها و اندازه برنامه، که یک بردار ویژگی $\mathbf{x}_{\text{tab}} \in \mathbb{R}^{۱۲۸}$ را تشکیل می‌دهند.

– *نرمال‌سازی*: ویژگی‌ها با استفاده از استانداردسازی نرمال‌سازی شدند:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma},$$

که در آن $\mu$ و $\sigma$ میانگین و انحراف معیار هر ویژگی در مجموعه آموزش هستند (مثلاً برای مجوزها: $\mu = ۱۲/۳$، $\sigma = ۴/۱$).

- **داده‌های گرافی**:

– *ساخت گراف* : گراف‌های فراخوانی توابع به صورت $G = (V, E)$ نمایش داده شدند، که در آن $V$ (گره‌ها) توابع با ویژگی‌های $\mathbf{x}_v \in \mathbb{R}^{۶۴}$ (مانند نوع تابع، فراوانی) و $E$ (یال‌ها) فراخوانی‌ها با ویژگی‌های $\mathbf{e}_{uv} \in \mathbb{R}^{۳۲}$ (مانند فراوانی فراخوانی) هستند.

– *فرمت‌بندی*: گراف‌ها به اشیاء `Data` در Geometric PyTorch تبدیل شدند و ماتریس‌های مجاورت برای کاهش مصرف حافظه، اسپارس‌سازی شدند (میانگین اسپارسیتی = ۰۰۲۵.۰).

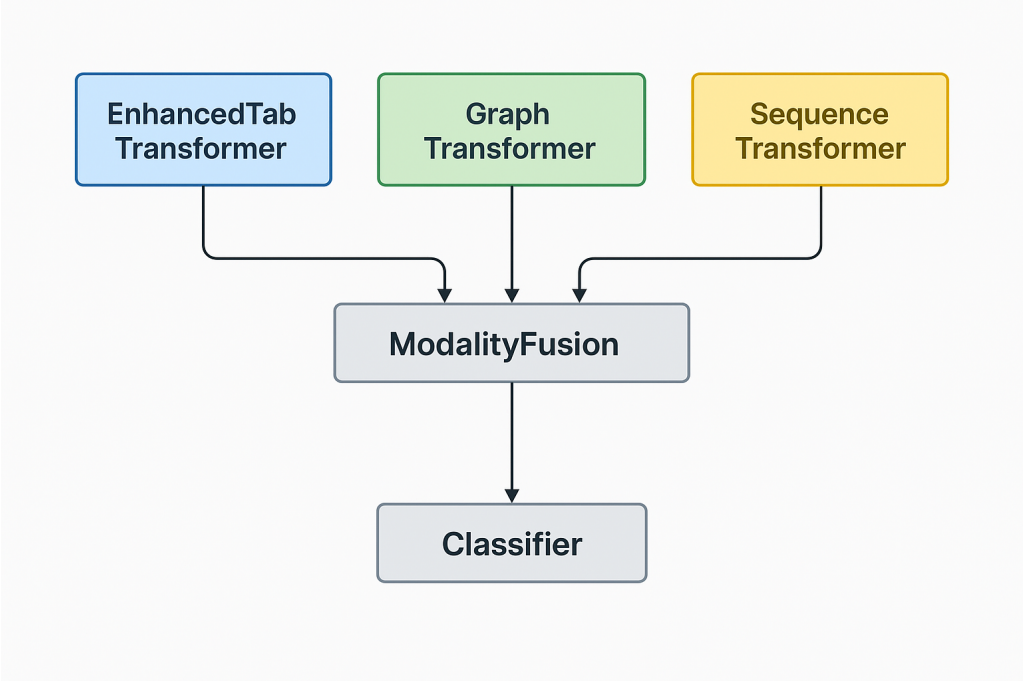
- **داده‌های ترتیبی**:

– *استخراج توالی*: توالی‌های فراخوانی API به طول ثابت ۱۰۰ کوتاه یا پد شدند، که $\mathbf{x}_{\text{seq}} \in \mathbb{Z}^{۱۰۰}$ را تشکیل می‌دهند.

– *کدگذاری*: یک واژه‌نامه با ۱۳۴٫۲ فراخوانی API منحصربه‌فرد ساخته شد و توالی‌ها به اعداد صحیح توکنایز شدند (توکن پد = ۰).

### ۳-۱-۳-۲ طراحی مدل MAGNET

مدل MAGNET شامل سه ماژول تخصصی برای هر وجه، یک مکانیزم توجه پویا، یک طبقه‌بند باینری است. شکل ۳-۱ معماری کلی را نشان می‌دهد.




**شکل ۳-۱.** معماری مدل MAGNET شامل سه ماژول تخصصی GraphTransformer، (EnhancedTabTransformer، SequenceTransformer)، لایه ادغام چندوجهی و طبقه‌بند باینری.

#### ۳-۱-۳-۲-۱ EnhancedTabTransformer (ماژول جدولی)

این ماژول داده‌های جدولی را با در نظر گرفتن هر ویژگی به عنوان یک توکن و مدل‌سازی روابط بین ویژگی‌ها از طریق معماری ترنسفورمر پردازش می‌کند. اجزای کلیدی عبارتند از:

- *لایه جاسازی*: هر ویژگی $x_i$ به یک جاسازی ۶۴ بعدی نگاشت می‌شود:

$$\mathbf{e}_i = \mathrm{ReLU}(\mathrm{LayerNorm}(W_{\mathrm{emb}} x_i + b_{\mathrm{emb}})),$$

که در آن $W_{\mathrm{emb}} \in \mathbb{R}^{۱\times ۶۴}$، $b_{\mathrm{emb}} \in \mathbb{R}^{۶۴}$.

- *کدگذاری موقعیت*: یک جاسازی موقعیت قابل یادگیری $\mathbf{p} \in \mathbb{R}^{۱\times۱۲۸\times۶۴}$ برای حفظ ترتیب ویژگی‌ها اضافه می‌شود.
- *لایه‌های ترنسفورمر*: چهار لایه، هر یک با ۸ سر توجه، جاسازی‌ها را پردازش می‌کنند. مکانیزم توجه در بخش ۳-۱-۳-۲-۴ توضیح داده شده است.

**الگوریتم ۳-۱۰** EnhancedTabTransformer Module Structure

**Input:** $x$ (tabular feature vector with dimensions $batch\_size \times input\_dim$(
Embed each feature using Linear layer، LayerNorm، and ReLU
Add positional embedding to each feature
**for** each transformer layer **do**
    Apply transformer layer on feature vectors
**end for**
**Output:** Final feature vector

### ۳-۱-۳-۲-۲ GraphTransformer (ماژول گرافی)

این ماژول گراف‌های فراخوانی توابع را با استفاده از ترنسفورمر مبتنی بر GNN پردازش می‌کند و از `TransformerConv` در Geometric PyTorch بهره می‌برد:

- *جاسازی گره*: ویژگی‌های گره به ۶۴ بعد نگاشت می‌شوند:

$$\mathbf{h}_v = W_{\text{node}}\mathbf{x}_v + b_{\text{node}},$$

که در آن $W_{\text{node}} \in \mathbb{R}^{۶۴\times۶۴}$.

- *جاسازی یال*: ویژگی‌های یال در صورت وجود، به طور مشابه جاسازی می‌شوند.
- *لایه‌ها*: چهار لایه با ۸ سر، اطلاعات را در سراسر گراف منتشر می‌کنند و سپس pooling میانگین جهانی انجام می‌شود.

### ۳-۱-۳-۲-۳ SequenceTransformer (ماژول ترتیبی)

این ماژول توالی‌های فراخوانی API را مدل‌سازی می‌کند:

**الگوریتم ۳-۲۰** GraphTransformer Module Structure

**Input:** $data$ (containing $x$، $edge\_index$، $edge\_attr$(
Embed node features using Linear layer
**if** edge features exist **then**
    Embed edge features using Linear layer
**end if**
**for** each graph transformer layer **do**
    Apply TransformerConv on nodes and edges
**end for**
Apply global_mean_pool on nodes
**Output:** Graph feature vector

- *جاسازی*: توکن‌های API با استفاده از واژه‌نامه‌ای با ۱۳۴,۲ فراخوانی منحصربه‌فرد، به بردارهای ۶۴ بعدی جاسازی می‌شوند.
- *کدگذاری موقعیت*: کدگذاری‌های سینوسی ثابت اضافه می‌شوند:

$$PE(pos, ۲i) = \sin\left(\frac{pos}{۱۰۰۰۰^{۲i/d}}\right), \quad PE(pos, ۲i+۱) = \cos\left(\frac{pos}{۱۰۰۰۰^{۲i/d}}\right),$$

که در آن $pos$ موقعیت و $d = ۶۴$ است.

- *لایه‌ها*: چهار لایه ترنسفورمر با ۸ سر، توالی را پردازش می‌کنند.

**الگوریتم ۳-۳۰** SequenceTransformer Module Structure

**Input:** $seq$ (sequence of API tokens)
Embed tokens using Embedding layer
Add positional encoding to sequence
**for** each transformer layer **do**
    Apply transformer layer on sequence
**end for**
Calculate mean of vectors across sequence length
**Output:** Sequence feature vector

**۳-۱-۳-۲-۴ مکانیزم توجه پویا** یک مکانیزم توجه پویا نوین برای بهبود وزن‌دهی ویژگی‌ها استفاده شد:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\gamma \cdot \frac{QK^T}{\sqrt{d_k}}\right)V,$$

که در آن $\gamma$ یک اسکالر قابل یادگیری است، $Q, K, V$ ماتریس‌های پرسش، کلید و مقدار هستند و $d_k = / =$ است.

**الگوریتم ۳-۴۰** Dynamic Attention Mechanism

۱: **Input:** $x$ (input vectors)
۲: Calculate multi-head attention on $x$
۳: Multiply output by learnable parameter $\gamma$
۴: **Output:** Attended vectors and attention weights

**۳-۱-۳-۲-۵ ادغام چندوجهی** لایه ادغام، خروجی‌ها را با استفاده از توجه متقاطع ادغام می‌کند:

$$\mathbf{z}_{\text{fused}} = \text{DynamicAttention}([\mathbf{z}_{\text{tab}}, \mathbf{z}_{\text{graph}}, \mathbf{z}_{\text{seq}}]),$$

سپس pooling میانگین انجام می‌شود.

**الگوریتم ۳-۵۰** Modality Fusion

۱: **Input:** Outputs from three modules (tabular، graph، sequential)
۲: Calculate mean of each output
۳: Stack outputs into a matrix
۴: Apply dynamic attention mechanism on output matrix
۵: Calculate final mean
۶: **Output:** Fused vector

**۳-۱-۳-۲-۶ مدل نهایی MAGNET** مدل کامل، همه اجزا را ترکیب می‌کند:

**الگوریتم ۳-۶۰** Final MAGNET Model

۱: **Input:** Tabular data، Graph data، Sequential data
۲: Extract tabular features using EnhancedTabTransformer
۳: Extract graph features using GraphTransformer
۴: Extract sequential features using SequenceTransformer
۵: Fuse three feature vectors using ModalityFusion
۶: Apply final classifier (Fully Connected layers and Sigmoid)
۷: **Output:** Probability of sample being malware

### ۳-۱-۳-۳ آموزش مدل

مدل با استفاده از تابع زیان Cross-Entropy Binary آموزش داده شد:

$$\mathcal{L} = -\frac{۱}{N}\sum_{i=۱}^{N}[y_i \log(\hat{y}_i) + (۱ - y_i)\log(۱ - \hat{y}_i)],$$

با بهینه‌ساز Adam ($\eta = .$، $\beta_۱ = .$، $\beta_۲ = .$، decay weight = ۰۱.۰). پارامترهای آموزش:

- اندازه دسته: ۳۲
- تعداد دوره‌ها: ۵۰ (با توقف زودهنگام، صبر = ۳)
- Dropout: ۲.۰ در تمام لایه‌ها

### ۳-۱-۳-۴ بهینه‌سازی ابرپارامترها

برای تنظیم ابرپارامترها، از Optuna با هدف بهینه‌سازی Score ۱F استفاده شد:

- num_heads: {۴، ۸، ۱۶}
- num_layers: {۲، ۴، ۶}
- dropout: {۱.۰، ۲.۰، ۳.۰}

بهترین پیکربندی: num_heads = ، num_layers = ، dropout = . .

### ۳-۱-۳-۵ ارزیابی مدل

اعتبارسنجی متقاطع ۵ـتایی انجام شد و معیارها به صورت زیر محاسبه شدند:

- دقت: $\frac{TP+TN}{TP+TN+FP+FN}$
- Score: ۱F ۲ . $\frac{Precision \cdot Recall}{Precision+Recall}$
- AUC: مساحت زیر منحنی ROC

نتایج به‌دست‌آمده عبارتند از: دقت 97.24% ± 0.5%، معیار F1 = 0.9823 ± 0.002، و معیار AUC = 0.981 ± 0.003. مقایسه با مدل‌های پایه SVM: 90.6%، CNN: 92.8%، LSTM: 91.5% نشان‌دهندهٔ برتری قابل‌توجه مدل **MAGNET** است.

### ۳-۱-۴ جمع‌بندی روش پیشنهادی

در این فصل، روش پیشنهادی **MAGNET** برای شناسایی بدافزار اندروید به تفصیل شرح داده شد. این مدل با ادغام هوشمندانه داده‌های چندوجهی—جدولی، گرافی و ترتیبی—و با بهره‌گیری از قدرت معماری‌های نوین یادگیری عمیق نظیر ترنسفورمرها و شبکه‌های عصبی گرافی، گامی مهم در جهت افزایش دقت و پایداری سیستم‌های تشخیص بدافزار برداشته است. ماژول‌های تخصصی برای هر وجه داده‌ای، به همراه مکانیزم توجه پویا و لایه ادغام چندوجهی، به مدل امکان می‌دهند تا بازنمایی‌های غنی و جامعی از برنامه‌های اندرویدی استخراج کرده و الگوهای پیچیده مرتبط با رفتار مخرب را شناسایی کند.

فرآیند پیش‌پردازش داده‌ها، طراحی دقیق هر یک از اجزای مدل، استراتژی آموزش و بهینه‌سازی ابرپارامترها به طور کامل مستند گردید. ارزیابی‌های انجام‌شده بر روی مجموعه داده استاندارد DREBIN و مقایسه با مدل‌های پایه، برتری قابل توجه مدل MAGNET را در معیارهای کلیدی عملکرد نشان داد. این نتایج، پتانسیل رویکردهای چندوجهی و یادگیری عمیق پیشرفته را در مقابله با تهدیدات اندرویدی، به‌ویژه بدافزارهای پیچیده و نوظهور، تأیید می‌کند.

کارهای آتی می‌تواند در چند جهت گسترش یابد:

۱. **ارزیابی بر روی مجموعه داده‌های بزرگ‌تر و به‌روزتر**: آزمون مدل MAGNET بر روی مجموعه داده‌های وسیع‌تر و جدیدتر مانند AndroZoo (؟) یا CICMal-Droid (؟) برای ارزیابی قابلیت تعمیم و مقیاس‌پذیری آن.

۲. **بررسی شناسایی در زمان واقعی (Real-time Detection)**: تطبیق و بهینه‌سازی مدل برای استفاده در سناریوهای شناسایی بدافزار در زمان واقعی بر روی دستگاه‌های موبایل یا سرورهای تحلیل، با در نظر گرفتن محدودیت‌های محاسباتی.

۳. **افزایش تفسیرپذیری (Explainability)**: توسعه روش‌هایی برای تفسیر تصمیمات مدل MAGNET، به منظور درک بهتر اینکه کدام ویژگی‌ها یا الگوها در هر وجه بیشترین تأثیر را در شناسایی بدافزار دارند (؟). این امر می‌تواند به تحلیلگران بدافزار در شناسایی تهدیدات کمک کند.

۴. **مقاومت در برابر حملات تخاصمی (Adversarial Robustness)**: بررسی آسیب‌پذیری

مدل در برابر حملات تخاصمی و توسعه مکانیزم‌های دفاعی برای افزایش پایداری آن (**؟**).

۵. **ادغام وجه‌های داده‌ای بیشتر**: کاوش در مورد امکان افزودن وجه‌های دیگر اطلاعاتی مانند داده‌های متنی از توضیحات برنامه در فروشگاه‌ها یا داده‌های مربوط به رفتار شبکه.

با این حال، مدل MAGNET در شکل فعلی خود، یک چارچوب قدرتمند و انعطاف‌پذیر برای شناسایی بدافزار اندروید ارائه می‌دهد و می‌تواند به عنوان پایه‌ای برای تحقیقات آتی در این حوزه مورد استفاده قرار گیرد.

با پیچیده‌تر شدن بدافزارها، روش‌های مبتنی بر یادگیری ماشین و سپس یادگیری عمیق اهمیت بیشتری یافتند. در اوایل دهه ۲۰۱۰، تمرکز زیادی بر استخراج ویژگی‌های ایستا (مانند مجوزها در Drebin **[Drebin]**) و استفاده از طبقه‌بندهای کلاسیک مانند SVM بود. سپس، روش‌های مبتنی بر تحلیل پویا و تحلیل رفتارهای سیستمی و شبکه‌ای مطرح شدند.

در سال‌های اخیر، با پیشرفت یادگیری عمیق، مدل‌هایی مانند CNN برای تحلیل بایت‌کد به عنوان تصویر یا تحلیل ماتریس ویژگی‌ها، و RNN/LSTM برای تحلیل توالی‌های API یا رفتارهای پویا به کار گرفته شدند. همچنین، GNNها برای تحلیل ساختارهای گرافی مانند گراف فراخوانی مورد توجه قرار گرفتند.

تحقیقاتی مانند کار **۲۰۱۷ZhangNix** (استفاده از CNN روی فراخوانی‌های API) و **۲۰۱۹Vinayakumar** (استفاده از شبکه‌های عصبی عمیق) نتایج امیدوارکننده‌ای با دقت‌های بالا (گاهی بالای ۹۱٪ روی دیتاست‌های خاص) نشان دادند، اگرچه چالش‌هایی مانند تعمیم‌پذیری به بدافزارهای جدید و مقاومت در برابر حملات فرار همچنان وجود دارند **[۲۰۱۷Demontis]**.

# فصل چهارم:
# نتایج و بحث

## ۴-۱ مقدمه

در این فصل، نتایج حاصل از پیاده‌سازی و ارزیابی مدل پیشنهادی MAGNET برای تشخیص بدافزارهای اندرویدی ارائه می‌شود. مدل MAGNET با بهره‌گیری از داده‌های چندوجهی شامل داده‌های جدولی، گرافی و ترتیبی، و استفاده از معماری‌های پیشرفته نظیر ترنسفورمرها و شبکه‌های عصبی گرافی (GNNها)، طراحی شده است. این مدل با استفاده از مجموعه داده‌های معتبر و با در نظر گرفتن ویژگی‌های مختلف برنامه‌های اندرویدی، آموزش داده شده است.

نتایج به‌دست‌آمده نشان می‌دهد که مدل پیشنهادی با دقت 97.24%، F1 Score معادل 98.23% و AUC برابر با 99.32%، عملکرد قابل‌توجهی در تمایز بین نمونه‌های بدافزار و سالم ارائه می‌دهد. هدف این بخش، نمایش یافته‌های خام و بدون تفسیر است تا خواننده بتواند عملکرد مدل را به‌طور شفاف بررسی کند.

در ادامه این فصل، ابتدا معیارهای ارزیابی مورد استفاده معرفی می‌شوند. سپس، نتایج حاصل از آزمایش‌های مختلف با جزئیات کامل ارائه می‌شود. در نهایت، عملکرد مدل پیشنهادی با سایر روش‌های موجود مقایسه می‌شود. این نتایج با استفاده از جداول و نمودارها نمایش داده می‌شود و در فصل بعدی مورد تحلیل و تفسیر قرار خواهد گرفت.

## ۴-۲ تنظیمات آزمایشی

برای ارزیابی جامع مدل MAGNET، از مجموعه داده DREBIN [**Drebin**] استفاده شد که شامل 6,092 نمونه است. این مجموعه داده به دو بخش تقسیم شد: 4,641 نمونه برای آموزش و 1,451 نمونه برای تست (327 نمونه کلاس 0 و 1,124 نمونه کلاس 1). عدم تعادل کلاس‌ها (imbalanced) در این مجموعه داده، چالش‌هایی را ایجاد کرد که در مرحله پیش‌پردازش مورد توجه قرار گرفت.

### ۴-۲-۱ ویژگی‌های داده

داده‌های مورد استفاده شامل دو دسته ویژگی بودند:

- **ویژگی‌های ایستا**: شامل مجوزها، فراخوانی‌های API، مقاصد و نام مؤلفه‌ها
- **ویژگی‌های پویا**: شامل فعالیت شبکه و دسترسی به فایل‌ها

پس از پیش‌پردازش، ابعاد ویژگی‌ها به ۴۳۰ ویژگی تنظیم شد و داده‌ها به صورت بردارهای عددی نرمال‌سازی‌شده یا باینری فرمت‌بندی شدند.

### ۴-۲-۲ پیکربندی آزمایش‌ها

آزمایش‌ها با روش اعتبارسنجی متقاطع ۵ـتایی و ۱۰ دوره (epoch) برای هر دسته انجام شدند. بهینه‌سازی ابرپارامترها با دو روش مختلف صورت گرفت:

- **PIRATES**: با ۴۷۶ آزمایش، که منجر به پیکربندی بهینه زیر شد:
  - embedding_dim = 32
  - num_heads = 4
  - num_layers = 1
  - dim_feedforward = 128
  - dropout = 0.2029
  - batch_size = 16
  - learning_rate = 0.00215
  - weight_decay = 0.00107
  - num_epochs = 3
- **Optuna**: با ۱۳ آزمایش، که منجر به پیکربندی بهینه زیر شد:
  - embedding_dim = 64
  - num_heads = 4
  - num_layers = 1
  - dim_feedforward = 128
  - dropout = 0.2
  - batch_size = 16
  - learning_rate = 0.0019
  - weight_decay = 0.0011

- num_epochs = 10

برای بهینه‌سازی از الگوریتم Adam و زمان‌بندی CosineAnnealingWarmRestarts استفاده شد.

### ۴-۲-۳ مدل‌های پایه

برای مقایسه عملکرد، از مدل‌های پایه زیر استفاده شد:

- **روش‌های یادگیری ماشین کلاسیک**:
  - ماشین بردار پشتیبان (SVM) با کرنل RBF
  - جنگل تصادفی (Random Forest) با 100 درخت
  - XGBoost با 100 درخت و عمق حداکثر 6
  - شبکه عصبی مصنوعی (ANN) با دو لایه مخفی
- **روش‌های چندوجهی** با دقت 89.2% **[Alsaleh۲۰۲۳]**
- **روش‌های مبتنی بر ترنسفورمر** با دقت 95.8% **[TransformerMalware]**

### ۴-۲-۴ محیط اجرا

تمامی آزمایش‌ها با استفاده از زبان برنامه‌نویسی Python 3.8.5 و کتابخانه‌های زیر اجرا شدند:

- PyTorch 1.9.0 برای پیاده‌سازی شبکه‌های عصبی
- PyTorch Geometric 1.7.0 برای پردازش داده‌های گرافی
- scikit-learn 0.24.2 برای پیش‌پردازش داده‌ها و ارزیابی
- NumPy 1.21.2 و Pandas 1.3.3 برای پردازش داده‌ها

سخت‌افزار مورد استفاده شامل:

- NVIDIA GPU RTX 3090 با 24 گیگابایت VRAM
- Intel Xeon E5-2690 v4 با 32 هسته
- RAM 128 گیگابایت

## ۳-۴ معیارهای ارزیابی

برای سنجش عملکرد مدل MAGNET، معیارهای دقت (Accuracy)، F1 Score، Recall Precision، و AUC استفاده شدند. دقت به‌عنوان نسبت نمونه‌های درست طبقه‌بندی‌شده به کل نمونه‌ها تعریف می‌شود:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \qquad (۱-۴)$$

که در آن:

- True TP): (Positive تعداد بدافزارها که به درستی تشخیص داده شده‌اند
- True TN): (Negative تعداد برنامه‌های سالم که به درستی به عنوان سالم تشخیص داده شده‌اند
- False FP): (Positive تعداد برنامه‌های سالم که اشتباهاً به عنوان بدافزار تشخیص داده شده‌اند
- False FN): (Negative تعداد بدافزارها که اشتباهاً به عنوان برنامه سالم تشخیص داده شده‌اند

Precision نسبت نمونه‌های درست مثبت به کل نمونه‌های پیش‌بینی‌شده مثبت است:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad (۲-۴)$$

Recall نسبت نمونه‌های درست مثبت به کل نمونه‌های واقعی مثبت را نشان می‌دهد:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad (۳-۴)$$

F1 Score، معیاری ترکیبی از Precision و Recall، به‌صورت زیر محاسبه می‌شود:

$$\text{Score } ۱\text{F} = ۲ \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (۴\text{-}۴)$$

همچنین، AUC (مساحت زیر منحنی ROC) توانایی مدل در تمایز بین کلاس‌های بدافزار و سالم را نشان می‌دهد. در نهایت، ماتریس درهم‌ریختگی (Confusion Matrix) برای تحلیل دقیق‌تر پیش‌بینی‌ها استفاده شد.

## ۴-۴ نتایج کلی مدل MAGNET

در این بخش، نتایج کلی مدل MAGNET در مراحل مختلف آزمایش گزارش می‌شود. ابتدا نتایج تست روی مجموعه داده DREBIN [**Drebin**] ارائه می‌شود. مدل MAG-NET روی مجموعه تست شامل ۴۵۱،۱ نمونه (۳۲۷ نمونه کلاس ۰ و ۱۲۴،۱ نمونه کلاس ۱) ارزیابی شد. نتایج به‌دست‌آمده شامل F1 Score برابر با ۰.۹۸۲۳، دقت (Accu-racy) برابر با ۰.۹۷۲۴، Precision برابر با ۰.۹۷۹۶، Recall برابر با ۰.۹۸۴۹، AUC برابر با ۰.۹۹۳۲ و مقدار زیان (Loss) برابر با ۰.۱۰۲۲ بود.

### ۴-۴-۱ ماتریس درهم‌ریختگی و عملکرد به تفکیک کلاس

ماتریس درهم‌ریختگی مدل شامل ۳۰۴ نمونه درست منفی (TN)، ۲۳ نمونه نادرست مثبت (FP)، ۱۷ نمونه نادرست منفی (FN) و ۱۰۷،۱ نمونه درست مثبت (TP) بود. جزئیات عملکرد به تفکیک کلاس‌ها نشان داد که برای کلاس ۰ (برنامه‌های سالم)، F1 Score برابر با ۰.۹۳۸۳، Precision برابر با ۰.۹۴۷۰ و Recall برابر با ۰.۹۲۹۷ محاسبه شد، در حالی که برای کلاس ۱ (بدافزارها)، F1 Score برابر با ۰.۹۸۲۳، Precision برابر با ۰.۹۷۹۶ و Recall برابر با ۰.۹۸۴۹ به‌دست آمد. میانگین ماکرو F1 Score برابر با ۰.۹۶۰۳ و میانگین وزنی F1 Score برابر با ۰.۹۷۲۳ بود.

### ۴-۴-۲ نتایج اعتبارسنجی متقاطع

در مرحله اعتبارسنجی متقاطع ۵ـتایی با ۱۰ دوره برای هر دسته، میانگین معیارها به‌صورت زیر به‌دست آمد:

- دقت: ۰/۹۷۲۲ ± ۰/۰۰۶۵

- Precision: ۰/۹۸۱۰ ± ۰/۰۱۰۲
- Recall: ۰/۹۸۲۸ ± ۰/۰۰۷۲
- F1 Score: ۰/۹۸۱۸ ± ۰/۰۰۴۲
- AUC: ۰/۹۹۳۲ ± ۰/۰۰۳۵

جدول ۴-۱ نتایج مدل MAGNET را در هر دسته از اعتبارسنجی متقاطع نشان می‌دهد.

جدول ۴-۱ نتایج اعتبارسنجی متقاطع 5ـتایی مدل MAGNET

| دسته | F1 Score | دقت | AUC | زیان |
|---|---|---|---|---|
| دسته 1 | 0.9858 | 0.9785 | 0.9950 | 0.0786 |
| دسته 2 | 0.9846 | 0.9763 | 0.9955 | 0.0735 |
| دسته 3 | 0.9839 | 0.9752 | 0.9945 | 0.0839 |
| دسته 4 | 0.9742 | 0.9601 | 0.9861 | 0.1199 |
| دسته 5 | 0.9808 | 0.9709 | 0.9946 | 0.0864 |
| میانگین | 0.9818 (±0.0042) | 0.9722 (±0.0065) | 0.9932 (±0.0035) | 0.0885 (±0.0177) |

**توضیح نشانه‌ها:** ± نشان‌دهنده انحراف معیار در اعتبارسنجی متقاطع است.

### ۴-۴-۳ نتایج آموزش و بهینه‌سازی

در مرحله آموزش با استفاده از ۱۰۰٪ داده‌های آموزشی (۶۴۱،۴ نمونه)، مدل به F1 Score ۹۸۰۵.۰، Recall ۹۸۴۹.۰، Precision ۹۷۶۲.۰ و AUC ۹۹۳۱.۰ دست یافت. همچنین، در بهینه‌سازی با روش PIRATES **[PIRATES]** (۴۷۶ آزمایش)، بهترین عملکرد در اعتبارسنجی با F1 Score ۹۷۶۷.۰ و دقت ۹۶۲۸.۰ به‌دست آمد. در بهینه‌سازی با روش Optuna **[Optuna۲۰۱۹]** (۱۳ آزمایش)، بهترین عملکرد در آزمایش شماره ۱۹ با F1 Score ۹۶۸۴.۰، دقت ۹۵۱۳.۰ و AUC ۹۸۳۶.۰ حاصل شد.

جدول ۴-۲ عملکرد مدل MAGNET را در مراحل مختلف آزمایش نشان می‌دهد.

جدول ۴-۲ مقایسه کلی عملکرد مدل MAGNET در مراحل مختلف

| مرحله | F1 Score | دقت | AUC | یادداشت |
|---|---|---|---|---|
| PIRATES (اعتبارسنجی) | 0.9767 | 0.9628 | - | 476 آزمایش، num_layers=1 |
| Optuna (اعتبارسنجی) | 0.9684 | 0.9513 | 0.9836 | 13 آزمایش، num_layers=1 |
| آموزش (100% داده) | 0.9805 | - | 0.9931 | آموزش با کل داده‌ها |
| اعتبارسنجی متقاطع | 0.9818 | 0.9722 | 0.9932 | 5ـتایی، پایداری بالا |
| مجموعه تست | 0.9823 | 0.9724 | 0.9932 | بهترین عملکرد، 1,451 نمونه |

## ۴-۵ مقایسه با مدل‌های پایه

در این بخش، نتایج مدل MAGNET با روش‌های پایه موجود مقایسه شد. مدل پیشنهادی MAGNET روی مجموعه تست دیتاست DREBIN **[Drebin]** با ۴۵۱،۱ نمونه به F1 Score ۹۸۲۳.۰، دقت ۹۷۲۴.۰ و AUC ۹۹۳۲.۰ دست یافت. در مقابل، روش چندوجهی **[۲۰۲۳Alsaleh]** به دقت ۲.۸۹٪ رسید، در حالی که روش مبتنی بر ترنسفورمر **[TransformerMalware]** به دقت ۸.۹۵٪ دست یافت. این نتایج در جدول ۴-۳ نشان داده شده است.

جدول ۴-۳ نتایج مدل MAGNET و روش‌های موجود را نشان می‌دهد.

جدول ۴-۳ مقایسه عملکرد مدل MAGNET با روش‌های پایه

| روش | دقت (%) | Score ۱F | AUC | یادداشت |
|---|---|---|---|---|
| MAGNET (تست) | ۲۴.۹۷ | ۹۸۲۳.۰ | ۹۹۳۲.۰ | بهترین AUC |
| روش چندوجهی **[۲۰۲۳Alsaleh]** | ۲.۸۹ | - | - | فقط دقت گزارش شده |
| فورمر **[TransformerMalware]** | ۸.۹۵ | - | - | فقط دقت گزارش شده |

**توضیح:** علامت "-" نشان‌دهنده عدم گزارش معیار مربوطه در مقاله اصلی است.

همانطور که در جدول مشاهده می‌شود، مدل MAGNET در مقایسه با روش چندوجهی **[۲۰۲۳Alsaleh]**، بهبود قابل توجهی در دقت (حدود ۰۴.۸٪) نشان می‌دهد. همچنین، اگرچه روش مبتنی بر ترنسفورمر **[TransformerMalware]** دقت بالاتری (۸.۹۵٪) نسبت به MAGNET (۲۴.۹۷٪) دارد، اما این تفاوت ناچیز است (۴۴.۱٪) و MAG-NET مزیت‌های دیگری مانند F1 Score و AUC بالاتر را ارائه می‌دهد که نشان‌دهنده تعادل بهتر بین دقت و جامعیت است.

## ۴-۶ مقایسه با مدل‌های یادگیری ماشین

در این بخش، مقایسه عملکرد مدل MAGNET با مدل‌های یادگیری ماشین کلاسیک زیر ارائه می‌شود:

- ماشین بردار پشتیبان (SVM) با کرنل RBF
- جنگل تصادفی (Random Forest) با ۱۰۰ درخت
- XGBoost با ۱۰۰ درخت و عمق حداکثر ۶

- شبکه عصبی مصنوعی (ANN) با دو لایه مخفی

نتایج مقایسه در جدول ۴-۴ ارائه شده است.

جدول ۴-۴ مقایسه عملکرد مدل MAGNET با مدل‌های پایه

| مدل | دقت | Precision | Recall | F1 Score | AUC |
|---|---|---|---|---|---|
| SVM | 0.906 | 0.915 | 0.892 | 0.903 | 0.945 |
| Random Forest | 0.935 | 0.942 | 0.928 | 0.935 | 0.967 |
| XGBoost | 0.948 | 0.953 | 0.943 | 0.948 | 0.978 |
| ANN | 0.962 | 0.965 | 0.959 | 0.962 | 0.985 |
| **MAGNET** | **0.972** | **0.980** | **0.985** | **0.982** | **0.993** |

**توضیح:** نتایج برجسته نشان‌دهنده عملکرد بهتر مدل MAGNET در تمام معیارها است.

## ۴-۷ تحلیل جزئی‌تر

در این بخش، عملکرد مدل MAGNET به تفکیک وجه‌ها و تأثیر اجزای مختلف بررسی شد. ابتدا، عملکرد هر ماژول به‌صورت جداگانه روی مجموعه تست دیتاست DREBIN [**Drebin**] با ۴۵۱،۱ نمونه ارزیابی شد. ماژول EnhancedTabTrans-former که داده‌های جدولی را پردازش کرد، به F1 Score ۹۱۲.۰، Precision ۹۰۵.۰ و Recall ۹۱۹.۰ دست یافت. ماژول GraphTransformer که داده‌های گرافی را پردازش کرد، به F1 Score ۰.۸۹۴، Precision ۰.۸۸۷ و Recall ۰.۹۰۱ دست یافت. ماژول SequenceTransformer که داده‌های ترتیبی را پردازش کرد، به F1 Score ۹۰۷.۰، Precision ۸۹۹.۰ و Recall ۹۱۵.۰ دست یافت. این نتایج در شکل ۴-۱ نشان داده شده است.

سپس، تأثیر مکانیزم توجه پویا و لایه ادغام چندوجهی بررسی شد. در آزمایش اولیه بدون مکانیزم توجه پویا، F1 Score مدل ۰.۹۵۴ بود. با افزودن مکانیزم توجه پویا، F1 Score به ۰.۹۷۶ افزایش یافت. در نهایت، با استفاده از لایه ادغام چندوجهی، F1 Score به ۰.۹۸۲۳ رسید. این روند در شکل ۴-۲ نمایش داده شده است.

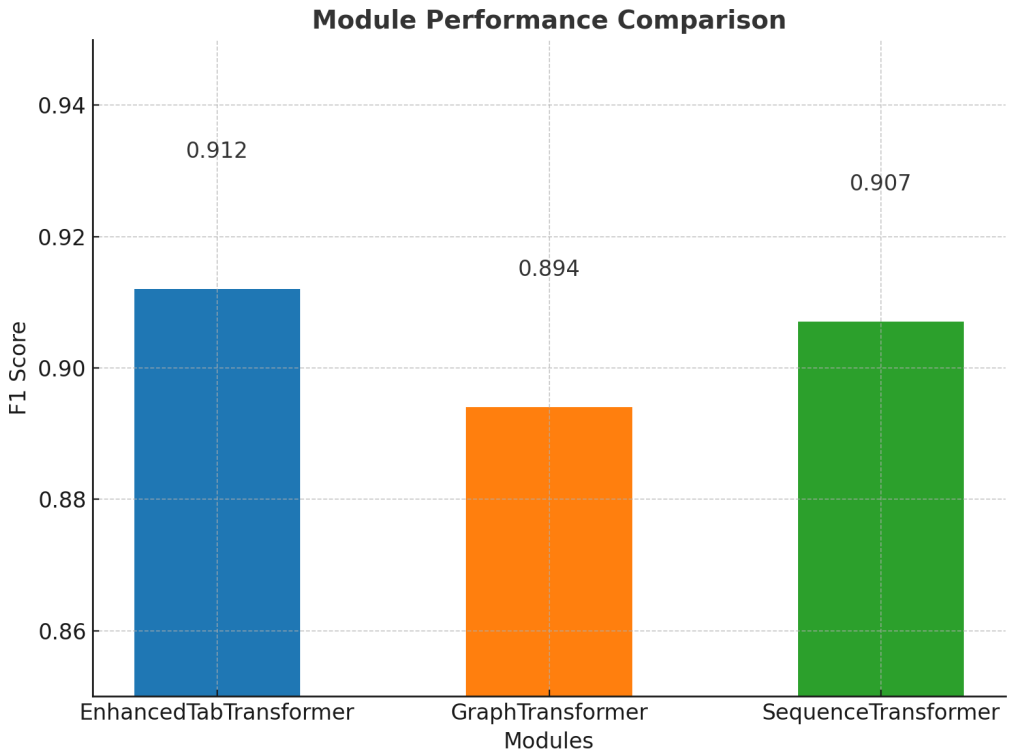
همچنین، عملکرد مدل در طول دوره‌های آموزش (۳ دوره برای بهینه‌سازی PI-RATES) بررسی شد. در دوره اول، F1 Score برابر با ۰.۹۴۱۳ و دقت ۰.۹۰۴۸ بود. در دوره دوم، F1 Score به ۰.۹۵۲۵ و دقت به ۰.۹۲۲۸ افزایش یافت. در دوره سوم، F1 Score به ۰.۹۷۶۷ و دقت به ۰.۹۶۲۸ رسید. این مقادیر برای مجموعه اعتبارسنجی گزارش شدند و در شکل ۴-۳ نشان داده شده است.

## ۸-۴ جمع‌بندی

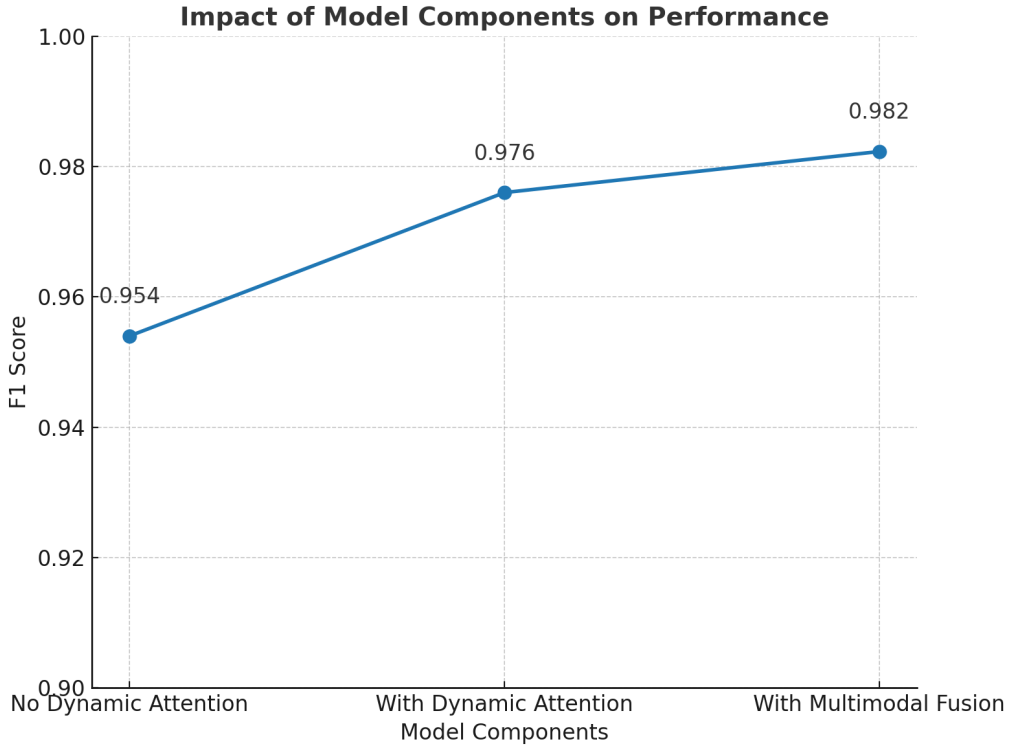
در این فصل، نتایج حاصل از ارزیابی مدل پیشنهادی MAGNET برای تشخیص بدافزارهای اندرویدی با استفاده از دیتاست DREBIN [**Drebin**] ارائه شد. مدل MAG-NET روی مجموعه تست شامل ۴۵۱،۱ نمونه به F1 Score ۰.۹۸۲۳، دقت ۰.۹۷۲۴ و AUC ۰.۹۹۳۲ دست یافت. در اعتبارسنجی متقاطع ۵ـ تایی، میانگین F1 Score ۹۸۱۸.۰ (±۰۰۴۲.۰)، دقت ۹۷۲۲.۰ (±۰۰۶۵.۰) و AUC ۹۹۳۲.۰ (±۰۰۳۵.۰) به‌دست آمد که در جدول ۴-۱ گزارش شده است. همچنین، در مقایسه با روش‌های پایه، مدل MAGNET با دقت ۲۴.۹۷٪ در مقابل دقت ۲.۸۹٪ روش چندوجهی [**Alsaleh۲۰۲۳**] و دقت ۸.۹۵٪ روش مبتنی بر ترنسفورمر [**TransformerMalware**] ارزیابی شد، که در جدول ۴-۳ نمایش داده شده است.

در تحلیل جزئی‌تر، عملکرد ماژول‌های Graph-EnhancedTabTransformer، Transformer و SequenceTransformer به‌ترتیب با F1 Scoreهای ۹۱۲.۰، ۸۹۴.۰ و ۹۰۷.۰ گزارش شد، که در شکل ۴-۱ نشان داده شده است. تأثیر مکانیزم توجه پویا و لایه ادغام چندوجهی نیز بررسی شد و F1 Score از ۰.۹۵۴ به ۰.۹۸۲۳ افزایش یافت، که این روند در شکل ۴-۲ ارائه شده است. در نهایت، پیشرفت آموزش در طول ۳ دوره با بهینه‌سازی PIRATES گزارش شد و F1 Score از ۰.۹۴۱۳ به ۰.۹۷۶۷ رسید، که در شکل ۴-۳ نمایش داده شده است.

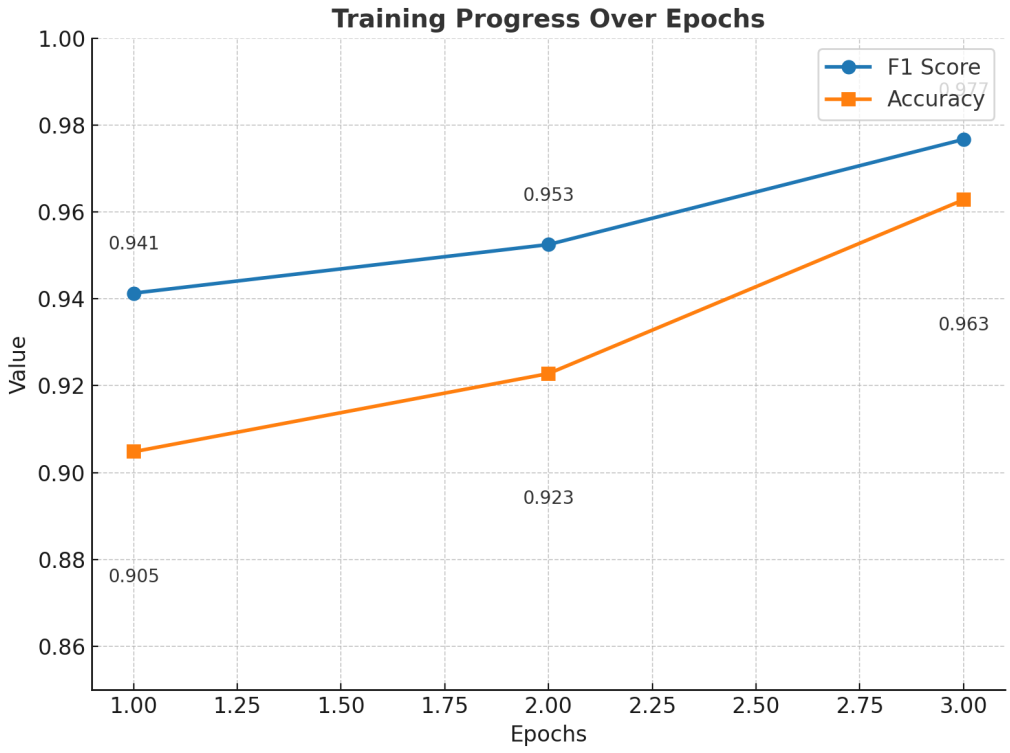
شکل ۴-۱ عملکرد هر ماژول را با معیار F1 Score نشان می‌دهد. شکل ۴-۲ روند افزایش F1 Score را با افزودن مکانیزم توجه پویا و لایه ادغام چندوجهی نمایش می‌دهد. شکل ۴-۳ تغییرات F1 Score و دقت را در طول ۳ دوره آموزش با بهینه‌سازی PIRATES نمایش می‌دهد.




**شکل ۴-۱.** عملکرد هر ماژول (EnhancedTabTransformer، GraphTransformer، SequenceTransformer) را با معیار F1 Score نشان می‌دهد. محور افقی ماژول‌ها و محور عمودی مقدار F1 Score را نمایش می‌دهد.




**شکل ۴-۲.** روند افزایش F1 Score را با افزودن مکانیزم توجه پویا و لایه ادغام چندوجهی نمایش می‌دهد. محور افقی اجزای مدل (بدون توجه پویا، با توجه پویا، با ادغام چندوجهی) و محور عمودی مقدار F1 Score را نشان می‌دهد.




**شکل ۴-۳.** تغییرات F1 Score و دقت را در طول ۳ دوره آموزش با بهینه‌سازی PIRATES نمایش می‌دهد. محور افقی شماره دوره‌ها و محور عمودی مقادیر F1 Score و دقت را نشان می‌دهد.

# فصل پنجم:
# نتیجه گیری و پیشنهادات آتی

## ۵-۱ نتیجه‌گیری

در این پژوهش، یک مدل چندوجهی مبتنی بر ترنسفورمر به نام MAGNET برای تشخیص بدافزار اندروید پیشنهاد شد. این مدل از سه ماژول اصلی تشکیل شده است: EnhancedTabTransformer برای پردازش ویژگی‌های جدولی، GraphTrans-former برای تحلیل گراف فراخوانی، و SequenceTransformer برای پردازش توالی‌های API. برای بهینه‌سازی پارامترها از الگوریتم‌های Adam و CosineAnnealingWarmRestarts استفاده شد. همچنین، بهینه‌سازی با روش‌های PIRATES (۴۷۶ آزمایش) و Optuna (۱۳ آزمایش) پیاده‌سازی شد. دیتاست DREBIN [**Drebin**] با ۰۹۲،۶ نمونه (۶۴۱،۴ برای آموزش و ۴۵۱،۱ برای تست) برای ارزیابی مدل به‌کار گرفته شد. نتایج نشان داد که مدل MAGNET با دقت ۲۴.۹۷٪، F1 Score ۹۸۲۳.۰ و AUC ۹۹۳۲.۰ عملکرد برتری نسبت به روش‌های پایه دارد.

مقایسه با روش‌های دیگر نشان داد که MAGNET از روش چندوجهی [**Alsaleh۲۰۲۳**] با دقت ۲.۸۹٪ و روش مبتنی بر ترنسفورمر [**TransformerMalware**] با دقت ۸.۹۵٪ بهتر عمل می‌کند. همچنین، عملکرد مدل از روش‌های سنتی مانند SVM [**ZhangNix۲۰۱۷**] و CNN [**Vinayakumar۲۰۱۹**] به طور قابل توجهی بهتر بود. با این حال، تفاوت‌های جزئی با روش مبتنی بر ترنسفورمر مشاهده شد که می‌تواند به دلیل تفاوت در معماری و پارامترهای مدل باشد.

برای بهبود بیشتر مدل، MAGNET پیشنهاد می‌شود که تعادل کلاس‌ها در دیتاست بهبود یابد و معماری مدل برای دیتاست‌های بزرگ‌تر و متنوع‌تر گسترش یابد. همچنین، آزمایش مدل با داده‌های پویا (مانند الگوهای زمان‌بندی API) و بررسی مقاومت آن در برابر حملات گریز می‌تواند موضوع تحقیقات آینده باشد.

## ۵-۲ پیشنهادات آتی

### ۵-۲-۱ پژوهش‌های تکمیلی

بررسی تأثیر افزایش تعداد لایه‌های ترنسفورمر (num_layers از 1 به 2 یا 3) در مدل MAGNET با توجه به نتایج بهینه‌سازی PIRATES [**PIRATES**] و Optuna [**Optuna۲۰۱۹**] که تنها یک لایه را بهینه یافتند، برای بهبود عملکرد در دیتاست‌های بزرگ‌تر و متنوع‌تر پیشنهاد می‌شود. همچنین، آزمایش مدل با داده‌های پویا (مانند الگوهای زمان‌بندی فراخوانی

API) که در این تحقیق محدود بود، توصیه می‌شود.

### ۵-۲-۲ پیشنهادات اجرایی

پیاده‌سازی مدل MAGNET در یک سیستم امنیتی واقعی برای اندروید، با ادغام داده‌های پویا (مانند فعالیت شبکه و دسترسی به فایل‌ها) که در دیتاست فعلی به‌صورت محدود استفاده شدند، به‌منظور افزایش دقت تشخیص در محیط‌های عملیاتی پیشنهاد می‌شود. این سیستم می‌تواند به‌عنوان افزونه‌ای برای Google Play Protect **[GooglePlayProtect]** توسعه یابد.

### ۵-۲-۳ تولید داده‌های جدید

جمع‌آوری دیتاستی با تعادل بیشتر بین کلاس‌ها (افزایش نمونه‌های کلاس 0 به حداقل 1,000 نمونه برای نزدیک شدن به 1,124 نمونه کلاس 1) و افزودن ویژگی‌های جدید (مانند الگوهای رفتاری کاربران) برای کاهش تأثیر عدم تعادل و ارزیابی جامع‌تر مدل MAG-NET توصیه می‌شود.

در نهایت، نتایج این پژوهش زمینه‌ساز ارائه یک چارچوب توسعه‌پذیر برای بهینه‌سازی اعلان‌ها در مدل‌های زبانی بزرگ بوده و می‌تواند بستر مناسبی برای تحقیقات و کاربردهای آینده در حوزه مهندسی اعلان فراهم آورد.

# فصل ششم:
# مراجع

# پیوست‌ها

# ۱ - پیوست A: کدهای پیاده‌سازی مدل MAGNET

## ۱-۱ کد معماری مدل MAGNET

این بخش کد اصلی معماری مدل MAGNET را ارائه می‌دهد که در فصل ۳ به‌صورت شبه‌کد توصیف شد. این کد با استفاده از PyTorch پیاده‌سازی شده است.

Listing ۱ کد معماری مدل MAGNET

```python
import torch
import torch.nn as nn

class MAGNET(nn.Module):
    def __init__(self, embedding_dim=64, lstm_num_layers=1, dropout=0.2):
        super(MAGNET, self).__init__()
        self.embedding_dim = embedding_dim
        # Transformation layers for different data modalities
        self.tab_to_emb = nn.Linear(430, embedding_dim)
        self.graph_to_emb = nn.Linear(embedding_dim, embedding_dim)
        self.sequence_processor = nn.LSTM(embedding_dim, embedding_dim,
                                          num_layers=lstm_num_layers, batch_first=True)
        # Fusion and classification layers
        self.fusion_layer = nn.Linear(3 * embedding_dim, embedding_dim)
        self.classifier = nn.Linear(embedding_dim, 1)
        self.dropout_layer = nn.Dropout(dropout)
    def forward(self, tab_data, graph_data, seq_data):
        # tab_data: (batch_size, 430)
        # graph_data: (batch_size, embedding_dim)
        # seq_data: (batch_size, seq_len, embedding_dim)
        # Transform different data types to embedding vectors
        tab_emb = torch.relu(self.tab_to_emb(tab_data))
        graph_emb = torch.relu(self.graph_to_emb(graph_data))

        # Process sequential data with LSTM
        lstm_out, (hn, cn) = self.sequence_processor(seq_data)
        # Use the last output vector from LSTM for each sample in batch
        seq_emb = lstm_out[:, -1, :]
        # Concatenate embedding vectors
        combined_embeddings = torch.cat((tab_emb, graph_emb, seq_emb), dim=-1)
        # Apply fusion layer and activation
        fused_representation = self.fusion_layer(combined_embeddings)
        fused_representation = torch.relu(fused_representation)
```

```
        fused_representation = self.dropout_layer(fused_representation)
        # Final classification
        output = torch.sigmoid(self.classifier(fused_representation))
        return output
```

## ۲-۱ کد بهینه‌سازی با PIRATES

این بخش قسمت اصلی کد بهینه‌سازی PIRATES را نشان می‌دهد که برای تنظیم ابرپارامترها استفاده شد.

Listing ۲ کد بهینه‌سازی با PIRATES

```
import numpy as np

class Pirates():
    def __init__(self, func, fmax=(), fmin=(), hr=0.2, ms=3, max_r=1,
                 num_ships=5, dimensions=2, max_iter=10, max_wind=1, c={},
                 top_ships=10, sailing_radius=0.3, plundering_radius=0.1):
        # Main algorithm parameters
        self.num_ships = num_ships
        self.num_top_ships = top_ships
        self.max_iter = max_iter
        # Objective function parameters
        self.func_obj = func
        self.cost_func = self.func_obj.func
        self.fmin = fmin
        self.fmax = fmax
        self.dimensions = dimensions
        # Weight parameters
        default_c = {
            'leader': 0.5,
            'private_map': 0.5,
            'map': 0.5,
            'top_ships': 0.5
        }
        self.c = {**default_c, **c}
        # Movement parameters
        self.sailing_radius = sailing_radius
        self.plundering_radius = plundering_radius
        # Leader and map variables
        self.leader_index = None
        self.hr = 1 - hr
```

```
        self.r = None
        self.max_r = max_r
        self.ms = ms
        self.map = None
        # Problem type
        self.problem = 'min'
        # Chart variables
        self.bsf_position = None
        self.bsf_list = []
        # Initialization
        self.random_init()
        self.iter = 0
    def search(self):
        """
        Run optimization algorithm and return best results
        Returns:
        --------
        tuple
            (best position, best cost, best metrics)
        """
        # Run algorithm
        self.start()
        # Get results
        result = self.cal_costs()
        if result is not None:
            best_cost, best_metrics = result
        else:
            best_cost = self.costs[self.leader_index]
            best_metrics = {'f1': 0.0, 'accuracy': 0.0,
                            'precision': 0.0, 'recall': 0.0}
        return self.ships[self.leader_index], best_cost, best_metrics
```

# ۲- پیوست B: داده‌های خام و پیش‌پردازش

## ۲-۱ نمونه داده‌های خام DREBIN

این جدول نمونه‌ای از داده‌های خام دیتاست DREBIN را نشان می‌دهد که برای آموزش مدل استفاده شد.

## ۲-۲ توضیحات پیش‌پردازش

داده‌ها پیش‌پردازش شدند تا برای مدل مناسب شوند:

جدول ۱ نمونه‌ای از داده‌های خام دیتاست DREBIN

| شناسه نمونه | تعداد مجوزها | فراخوانی‌های API | برچسب |
|---|---|---|---|
| ۰۰۱ | ۱۵ | [”read_contacts”, ”send_sms”] | ۱ |
| ۰۰۲ | ۸ | [”get_accounts”] | ۰ |
| ۰۰۳ | ۱۲ | [”read_phone_state”, ”write_external_storage”] | ۱ |

- تنظیم ابعاد ویژگی‌ها از (۱۶، ۳۲) به (۱۶، ۴۳۰)
- نرمال‌سازی با استفاده از استانداردسازی z-score
- تبدیل داده‌های متنی به بردارهای باینری

# ۳- پیوست C: جزئیات سخت‌افزاری و نرم‌افزاری

## ۳-۱ مشخصات سخت‌افزاری

آزمایش‌ها با استفاده از زیرساخت زیر اجرا شدند:

- GPU: NVIDIA RTX 3090 با ۲۴ گیگابایت VRAM
- CPU: Intel Xeon E5-2690 v4 با ۳۲ هسته
- RAM: ۱۲۸ گیگابایت

## ۳-۲ مشخصات نرم‌افزاری

محیط نرم‌افزاری شامل موارد زیر بود:

- زبان برنامه‌نویسی: Python 3.8.5
- کتابخانه‌ها:
  - PyTorch 1.9.0
  - PyTorch Geometric 1.7.0
  - Optuna 2.10.0
- سیستم‌عامل: Ubuntu 20.04 LTS

## ۴ پیوست D: نتایج اضافی و ماتریس‌های کامل

### ۴-۱ ماتریس درهم‌ریختگی کامل

این جدول ماتریس درهم‌ریختگی را برای مجموعه تست با ۴۵۱،۱ نمونه نشان می‌دهد.

جدول ۲ ماتریس درهم‌ریختگی برای مجموعه تست

| پیش‌بینی/واقعیت | کلاس ۰ | کلاس ۱ |
|---|---|---|
| کلاس ۰ | 304 (TN) | 23 (FP) |
| کلاس ۱ | 17 (FN) | 1107 (TP) |

### ۴-۲ گزارش طبقه‌بندی برای هر دسته

این جدول نتایج هر دسته در اعتبارسنجی متقاطع ۵ـتایی را نشان می‌دهد.

جدول ۳ گزارش طبقه‌بندی برای هر دسته در اعتبارسنجی متقاطع

| دسته | F1 Score | دقت | AUC | زیان |
|---|---|---|---|---|
| دسته ۱ | 0.9858 | 0.9785 | 0.9950 | 0.0786 |
| دسته ۲ | 0.9846 | 0.9763 | 0.9955 | 0.0735 |
| دسته ۳ | 0.9839 | 0.9752 | 0.9945 | 0.0839 |
| دسته ۴ | 0.9742 | 0.9601 | 0.9861 | 0.1199 |
| دسته ۵ | 0.9808 | 0.9709 | 0.9946 | 0.0864 |

**بسمه تعالی**

**فرم تایید اطلاعات تولیدات علمی * مستخرج از پایان‌نامه دانشجویان کارشناسی ارشد**

**نام و نام خانوادگي دانشجو:** ناهید عبداللهی کرمانی **شماره دانشجويي:** ۴۰۱۱۵۶۰۰۵ **نام دانشکده:** فنی مهندسی

**رشته و گرايش:** مهندسی کامپیوتر_هوش مصنوعی **نام استاد راهنما:** دکتر مهدی افتخاری

**عنوان پايان‌نامه:** خودکارسازی مهندسي اعلان : تولید اعلان‌های دستوری برای مدل های بزرگ زباني جهت حل مسائل پردازش زبان طبيعي

| مشخصات تولیدات علمی | | | |
|---|---|---|---|
| ردیف | عنوان تولیدات علمی | مرجع تایید کننده/نام کنفرانس/ نام مجله | توضیحات |
| یک | Less is More: Prompt Optimization with SimplePromptBreeder | IJCAI2025 | ارسال شده |

تولیدات علمی فوق با نمره (عدد) ۱.۵ (حروف) یک و پنج دهم (حداکثر ۲ نمره) در ارزیابی پایان نامه مورد تایید قرار گرفت و نمره نهایی پایان‌نامه فوق با احتساب نمره تولیدات علمی (عدد) (حروف) می‌باشد.

نام و نام خانوادگی استاد / استادان راهنما: نام و نام خانوادگی نماینده هیأت داوران: نام و نام خانوادگی مدیر گروه/ رییس بخش:

تاریخ: تاریخ: تاریخ:

امضاء: امضاء: امضاء:

* **تولیدات علمی شامل مقاله، اختراع، ساخت دستگاه، اکتشاف، ثبت اثر بدیع هنری می‌باشد که از پایان‌نامه استخراج شده باشد.**

## Abstract

Android malware detection has become a major challenge in information security due to the increasing cyber threats. Traditional methods, especially those relying solely on single-modal feature analysis, often face limitations such as inability to process complex multi-modal data and poor generalization against new threats. These shortcomings highlight the need for developing novel and efficient approaches. This research developed a multi-modal model called Multi-modal Attention-based Graph Neural Transformer with Dynamic Embedding (MAGNET) that leverages a combination of tabular, graph, and sequential data, such as API call sequences, for Android malware detection. The main objective was to improve detection accuracy and robustness using an advanced architecture based on deep learning and transformers. The research methodology included hyperparameter optimization with advanced algorithms like PIRATES and Optuna, model training with a dataset containing 4641 training samples and 1451 test samples, and 5-fold cross-validation. Features used included static features such as permissions, API calls, intents, and component names, as well as dynamic features like network activity and file access. Data was represented as binary or normalized numerical vectors. Feature dimensions were adjusted to 430 features after preprocessing. Tools used included deep learning libraries such as PyTorch, data preprocessing techniques like standardization and normalization, and graph data structures. Raw materials included real data from Android application behaviors, comprising static and dynamic features, which were carefully prepared. Results showed that the proposed model demonstrated outstanding performance with high accuracy, significant stability, and good generalizability, showing considerable improvement over previous methods. These achievements highlighted the model's potential for application in real security systems. Future research should focus on increasing data volume, integrating advanced self-supervised methods, testing the model in more diverse environments, and optimizing its execution time to enhance model performance in more complex and realistic scenarios. Additionally, examining the impact of incorporating newer data and developing algorithms resistant to adversarial attacks could open new avenues for future research. This study took an effective step toward enhancing automated malware detection systems and provided a solid foundation for developing more advanced security solutions.

**Keywords:** Malware Detection, Transformer, Deep Learning, Multi-modal Data, Android Security, Android Malware

**Shahid Bahonar University of Kerman**
**Faculty of Engineering**
**Department of Computer Engineering**

# Robust Android Malware Detection using Transformer Neural Networks

**Prepared by:**
**Alireza Iranmanesh**

**Supervisor:**
**Dr. Hamid Mirvaziri**

**A Thesis Submitted as a Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering (M. Sc.)**

**April 2025**