



دانشگاه علم و صنعت ایران  
دانشکده مهندسی کامپیوتر

# تولید خودکار داده آزمون در فازرهای قالب فایل

پایان نامه برای دریافت درجه کارشناسی ارشد در رشته مهندسی کامپیوتر  
گرایش نرم افزار

دانشجو:

مرتضی ذاکری نصرآبادی

استاد راهنما:

دکتر سعید پارسا

شهریور ۹۷

سُبْرَةِ

## تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه

نام دانشکده: دانشکده مهندسی کامپیوتر

نام دانشجو: مرتضی ذاکری نصرآبادی

عنوان پایان‌نامه: تولید خودکار داده آزمون در فازرهای قالب فایل

تاریخ دفاع: شهریور ۹۷

رشته: مهندسی کامپیوتر

گرایش: نرم‌افزار

ردیف	سمت	نام و نامخانوادگی	مرتبه دانشگاهی	دانشگاه / مؤسسه	امضا
۱	استاد راهنمای اول	دکتر سعید پارسا	دانشیار	دانشگاه علم و صنعت ایران	
۲	استاد راهنمای دوم	-	-	-	
۳	استاد مشاور	-	-	-	
۴	استاد مدعو داخلی	دکتر محمد عبدالله ازگمی	دانشیار	دانشگاه علم و صنعت ایران	
۵	استاد مدعو خارجی	دکتر مجتبی وحیدی‌اصل	استادیار	دانشگاه شهید بهشتی	

ب

## تأییدیهی صحت و اصالت نتایج

با اسمه تعالیٰ

اینجانب مرتضی ذاکری نصرآبادی به شماره دانشجویی ۹۵۷۲۳۰۸۸ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی ارشد تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احراق حقوق مکتب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذیصلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: مرتضی ذاکری نصرآبادی

تاریخ و امضا:

پ

## مجوز بهره برداری از پایان نامه

- بهره برداری از این پایان نامه در چهار چوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می شود، بلامانع است:
- بهره برداری از این پایان نامه برای همگان بلامانع است.
  - بهره برداری از این پایان نامه با اخذ مجوز از استاد راهنما، بلامانع است.
  - بهره برداری از این پایان نامه تا تاریخ ..... ممنوع است.

استاد راهنما: دکتر سعید پارسا

تاریخ:

امضا:

## تقدیم

به دوستی که

می‌اندیشد،

می‌جوید،

می‌یابد

و

تغییر می‌دهد (:)

## قدردانی

افرادی در دوره مهیج و کوتاه کارشناسی ارشد و در روند پژوهش و نگارش این پایاننامه به بنده کمک کرده‌اند که برخود لازم می‌دارم از زحمات ایشان سپاس‌گزاری نمایم. در ابتدا از استاد دانشمند و فرهیخته، آقای دکتر سعید پارسا که راهنمایی اینجانب را بر عهده داشتند، صمیمانه قدردانی می‌کنم. بدون وجود حمایت‌ها و راهنمایی‌های ارزنده ایشان انجام این پایاننامه محقق نمی‌گردید. از داوران محترم پایاننامه آقایان دکتر محمد عبداللّهی ازگمی و دکتر مجتبی وحیدی اصل که با مطالعه و نقد سازنده خود، اسباب بهبود علمی و رفع نواقص این کار پژوهشی را فراهم نمودند، کمال تشکر را دارم. از همراهی‌ها و دلسوزی‌های همیشگی دو دوست و هم‌آزمایشگاهی بسیار عزیزم، آقای مهندس محسن امیریان و آقای مهندس سعید امیری، که بیش از ۶ سال با ایشان همکلاسی بودم، بی‌اندازه سپاس‌گزار هستم. از همکلاسی‌های با معرفت و مهربانی در این دوره که همواره حامی و مشوق بنده بوده‌اند، لحظه‌های تکرار نشدنی را کنار ایشان تجربه کرده‌ام و متأسفانه مجال نامبردن از آنها را در این متن کوتاه ندارم، بی‌نهایت ممنون هستم. آشنایی با این عزیزان را دستاورده ارزشمند در زندگی خود می‌دانم و برای شان بهترین‌ها را آرزو ممندم. از دوستان گرامی، آقایان مهندس علی صابری و مهندس علی طاهری در آزمایشگاه رسانه دیجیتال دانشگاه صنعتی شریف، بابت انتقال مفاهیم اولیه یادگیری ژرف و آقای مهندس عرفان شرف‌زاده در آزمایشگاه سیستم‌های توزیع شده دانشکده مهندسی کامپیوتر، بابت پشتیبانی امور مربوط به پردازش ابری و ماشین‌های مجازی، تشکر می‌کنم. از آقای دکتر کارپتی برای وبلاگ و آموزش‌های مفید خود در زمینه یادگیری ژرف سپاس‌گزارم. از دیگر اساتید بزرگوارم در طول دوره کارشناسی ارشد از جمله آقایان دکتر محسن شریفی، دکتر بهروز مینایی و دکتر مهرداد آشتیانی (مدیر مرکز پردازش ابری دانشکده مهندسی کامپیوتر)، ممنون هستم. مهم‌تر از همه در پایان از خانواده عزیزم، به خصوص پدر و مادرم بابت حمایت‌های بی‌دریغ و همیشگی‌شان در همه مراحل زندگیم، مهربانانه سپاس‌گزاری کرده، دست ایشان را می‌بوسم.

مرتضی ذاکری نصرآبادی

شهریور ۹۷

# تولید خودکار داده آزمون در فاز راهی قالب فایل

## چکیده

آزمون فازی (فازینگ) یک فن آزمون پویای نرم افزار است. در این فن با تولید و تزریق مکرر داده‌های آزمون بدشکل به نرم افزار تحت آزمون (SUT)، به دنبال یافتن خطاهای آسیب‌پذیری‌های احتمالی موجود در آن هستیم. برای نیل به این هدف، آزمون فازی نیازمند داده‌های آزمون متنوع است. مشکل اساسی، پیچیده بودن ساختار ورودی برنامه‌هایی است که فایل را به عنوان ورودی می‌پذیرند. بررسی‌ها نشان می‌دهد بسیاری از داده‌های آزمون تولیدی در این موارد، مسیرهای محدود و سطحی را می‌پیمایند؛ زیرا در همان مراحل اولیه به علت بدشکل، بودن توسط تجزیه‌گر SUT رد می‌شوند. استفاده از ساختار گرامری فایل‌ها برای تولید داده‌ها، منجر به افزایش پوشش کد می‌شود؛ اما، استخراج گرامر برای ساختار فایل، اغلب، دستی صورت می‌پذیرد که مستلزم صرف هزینه و زمان زیاد و مستعد خطای فراوان است. در این پایان‌نامه، روشی خودکار برای تولید داده آزمون به صورت ترکیبی ارائه می‌دهیم. در روش خود، از مدل‌های زبانی عصبی (NLMs) که با شبکه‌های عصبی مکرر (RNNs) ساخته می‌شوند، استفاده می‌کنیم. مدل‌های پیشنهادی با فنون یادگیری ژرف، قادر به یادگیری آماری ساختار فایل‌های پیچیده و سپس تولید داده‌های جدید متنی، به صورت مبتنی بر گرامر و داده‌های دودویی به صورت مبتنی بر جابه‌جایی، هستند. فاز (بدشکل‌سازی) داده‌های آزمون، نیز توسط دو الگوریتم فاز جدید، تحت عنوان الگوریتم‌های فاز عصبی، که از این مدل‌ها استفاده می‌کنند، صورت می‌پذیرد. از روش پیشنهادی خود، برای تولید داده و سپس آزمون فازی نرم افزار پیچیده MuPDF که فایل‌های قالب سند حمل‌پذیر (PDF) را به عنوان ورودی می‌پذیرد، استفاده نمودیم. برای آموزش مدل‌های مولد، یک پیکره بزرگ از فایل‌های PDF را گردآوری کردیم. آزمایش‌های ما نشان می‌دهد که داده‌های تولید شده با این روش، منجر به افزایش میزان پوشش کد اجرایی SUT و بهبود بیش از ۷ درصدی آن، در مقایسه با فازرها قابل فایل مشهور، مثل AFL، می‌شود. آزمایش‌ها همچنین، بیان‌گر دقیق‌تر یادگیری NLM‌های ساده‌تر در قیاس با مدل پیچیده‌تر کدگزار—کدگشا و نیز شکست این مدل، در میزان پوشش کد SUT، حین آزمون فازی، هستند.

**واژگان کلیدی:** آزمون فازی، داده آزمون، پوشش کد، یادگیری ژرف، شبکه عصبی مکرر.

# فهرست مطالب

ر

فهرست شکل‌ها

ژ

فهرست جدول‌ها

س

فهرست الگوریتم‌ها

ش

فهرست کوتاه‌نوشت‌ها

۱

فصل ۱ : مقدمه

۱

۱-۱ پیش‌زمینه . . . . .

۲

۲-۱ شرح مسئله . . . . .

۳

۱-۲-۱ شهود اولیه . . . . .

۷

۱-۲-۲-۱ کارهای مرتبط . . . . .

۸

۱-۲-۳-۱ فرضیه‌ها و اهداف . . . . .

۹

۱-۳ روش پیشنهادی و نوآوری‌ها . . . . .

۱۰

۱-۴ اهمیت موضوع . . . . .

۱۲

۱-۵ ساختار پایان‌نامه . . . . .

۱۳

فصل ۲ : مفاهیم اولیه

۱۴

۱-۲ آزمون نرم‌افزار . . . . .

۱۴

۱-۱-۱ معیارهای پوشش . . . . .

۱۷

۱-۲-۱-۲ دیدگاه جعبه . . . . .

## فهرست مطالب

ح

۱۸	۱-۲-۳- اندازهگیری پوشش کد
۱۹	۲-۲ آزمون فازی
۱۹	۲-۲-۱ فازر
۲۱	۲-۲-۲-۲ معماری فازرها
۲۲	۲-۲-۲-۳ آسیب‌پذیری
۲۳	۲-۲-۳-۲ یادگیری ژرف
۲۴	۲-۳-۲ عصب
۲۴	۲-۳-۲ شبکه عصبی روبه‌جلو
۲۵	۲-۳-۳-۳ آموزش شبکه روبه‌جلو
۲۹	۲-۳-۴ شبکه عصبی مکرر
۳۲	۴-۲ مدل زبانی
۳۳	۴-۲-۱ مدل زبانی عصبی
۳۳	۴-۲-۲ ارزیابی مدل زبانی
۳۵	۵-۲ خلاصه

## فصل ۳: کارهای مرتبط

۳۷	AFL ۱-۳ فازر
۳۸	۱-۱-۳ معماری AFL
۳۹	۱-۲-۳ مشکلات AFL
۴۰	۱-۳-۲ AFL افزوده
۴۳	۲-۳ AFL افزوده فازر
۴۵	۱-۲-۳ AFL افزوده مشکلات
۴۵	۳-۳ یادگیری و فاز
۴۶	۳-۳-۱ کدگذار- کدگشا آموزش مدل
۴۷	۳-۳-۲ تولید داده‌های جدید
۴۸	۳-۳-۳ اعمال آزمون فازی
۵۰	۳-۳-۴ مشکلات روش یادگیری و فاز

۵۱	۴-۳ دیگر کارهای مرتبط
۵۲	۵-۳ خلاصه
<b>۵۵</b>	<b>فصل ۴: روش پیشنهادی</b>
۵۵	۴-۱ کلیات
۵۶	۴-۱-۱ تمایز داده‌های متنی و دودویی
۵۸	۴-۱-۲ تمایز داده و فراداده
۵۸	۴-۱-۳ مثال انگیزشی
۵۹	۴-۲ مدل
۶۱	۴-۲-۱ آموزش مدل
۶۶	۴-۲-۲ تولید داده‌های جدید
۶۷	۴-۳ الگوریتم‌های فاز عصبی
۶۸	۴-۳-۱ فاز عصبی داده
۶۹	۴-۳-۲ فاز عصبی فراداده
۷۰	۴-۴ پیاده‌سازی
۷۳	۴-۵ خلاصه
<b>۷۵</b>	<b>فصل ۵: ارزیابی روش پیشنهادی</b>
۷۵	۵-۱ مورد مطالعاتی
۷۶	۵-۲ معیارهای ارزیابی
۷۹	۵-۳ چیدمان آزمایش‌ها
۷۹	۵-۳-۱ مجموعه داده
۸۰	۵-۳-۲ پیش‌پردازش داده‌ها
۸۳	۵-۳-۳ انتخاب فایل‌های میزبان
۸۳	۵-۴-۳ پوشش کد مبنا
۸۵	۵-۴ آزمایش‌ها، یافته‌ها و مقایسه نتایج
۸۶	۵-۴-۱ سرگشتگی، خطای و دقت مدل‌ها

۸۷	۴-۲- پوشش کد مدل‌های مولد . . . . .	۵
۹۱	۴-۳- مقایسه با مدل کدگذار-کدگشا . . . . .	۵
۹۱	۴-۴- مقایسه در دوره‌های مختلف . . . . .	۵
۹۴	۴-۵- آزمون فازی عصبی . . . . .	۵
۹۷	۵- خلاصه . . . . .	۵

۱۰۰	<b>فصل ۶: نتیجه‌گیری و کارهای آتی</b>	
۱۰۰	۱- نتیجه‌گیری . . . . .	۶
۱۰۲	۱-۱- مزایا و معایب یادگیری ژرف . . . . .	۶
۱۰۳	۱-۲- مزایا و معایب آزمون فازی . . . . .	۶
۱۰۴	۲- نوآوری‌ها . . . . .	۶
۱۰۵	۳- ملاحظات اعتبارسنجی . . . . .	۶
۱۰۶	۴- کارهای آتی . . . . .	۶

۱۰۸	<b>مراجع</b>	
-----	--------------	--

۱۱۳	<b>پیوست آ: ساختار فایل PDF</b>	
۱۱۳	آ-۱ ساختار فیزیکی . . . . .	
۱۱۶	آ-۲ ساختار منطقی . . . . .	
۱۱۷	آ-۳ بروزرسانی فایل PDF . . . . .	

۱۱۸	<b>پیوست ب: فازر IUST-DeepFuzz</b>	
۱۱۸	ب-۱ پیش‌نیازهای نرم‌افزاری و سخت‌افزاری . . . . .	
۱۱۸	ب-۲ ساختار سطح بالای پروژه . . . . .	
۱۲۰	ب-۳ پیکربندی پروژه . . . . .	
۱۲۱	ب-۴ پیاده‌سازی‌ها . . . . .	
۱۲۱	ب-۴-۱ تعریف مدل‌ها . . . . .	
۱۲۳	ب-۴-۲ آموزش و تولید داده . . . . .	

فهرست مطالب

ذ

ب-۴- پیادهسازی آزمون فازی ..... ۱۲۳

ب-۵ ملاحظات عملی ..... ۱۲۴

واژه‌نامه فارسی به انگلیسی ..... ۱۲۵

واژه‌نامه انگلیسی به فارسی ..... ۱۲۹

# فهرست شکل‌ها

۱۷	۱-۲ دیدگاه جعبه در آزمون نرم افزار
۲۰	۲-۲ روند نمای فرایند آزمون فازی
۲۵	۲-۳ ساختمان داخلی یک عصب
۲۶	۲-۴ گراف محاسباتی شبکه عصبی رو به جلو
۳۰	۲-۵ گراف محاسباتی شبکه عصبی مکرر
۳۱	۲-۶ انواع مختلف شبکه عصبی مکرر بر اساس طول توالی ورودی و خروجی
۳۴	۲-۷ مدل زبانی عصبی مکرر
۳۹	۳-۲ مؤلفه‌های فازر AFL و ارتباط آنها
۳۹	۳-۱ محیط زمان اجرای AFL
۴۶	۳-۳ مدل کدگزار-کدگشا
۴۷	۳-۴ مدل RNN کدگزار-کدگشا برای تولید اشیای داده‌ای PDF
۵۷	۴-۱ روند نمای روشن پیشنهادی در حالت کلی
۶۰	۴-۲ مراحل یادگیری ساختار فایل، تولید و فاز داده آزمون برای فایل PDF
۶۲	۴-۳ گراف محاسباتی مدل‌های عصبی ژرف جدول ۱-۴
۶۵	۴-۴ مثالی از نحوه آموزش مدل‌های چند به یک
۷۴	۴-۵ معما ری فازر قالب فایل پیشنهادی و ارتباط بین مؤلفه‌های مختلف آن
۸۴	۵-۱ پوشش کد برای هریک از میزبان‌ها و پوشش کدهای مبنا
۸۶	۵-۲ نمودار تغییرات خطای مدل‌های ۲ و laf

## فهرست شکل‌ها

ز

۳-۵ نمودار تغییرات پوشش کد مدل‌های مختلف بر حسب تنوع . . . . .	۸۹
۵-۵ نمودار تغییرات پوشش کد بر حسب دوره برای مدل‌های ۲ و laf . . . . .	۹۳
۶-۵ نمونه‌ای از داده‌های آزمون متنوع تولید شده توسط الگوریتم فاز عصبی داده در فرایند آزمون فازی قالب فایل PDF . . . . .	۹۸
آ-۱ یک فایل PDF باز شده در یک ویراستار متنی شامل عبارت Hello World . . . . .	۱۱۴
آ-۲ یک شیء PDF حاوی جریان دودویی . . . . .	۱۱۶
آ-۳ نمایش درختی ساختار منطقی فایل PDF . . . . .	۱۱۶
آ-۴ جدول ارجاع متقابل بروزرسانی شده یک فایل PDF . . . . .	۱۱۷

# فهرست جدول‌ها

۱-۳ مقایسه کارهای مرتبط . . . . .	۵۳
۴-۱ مدل‌های یادگیری ژرف طراحی شده، پارامترها و ابرپارامترهای آنها. . . . .	۶۱
۱-۵ پارامترهای مؤثر در تولید خودکار داده آزمون با روش‌های یادگیری ماشینی و مقادیر قابل آزمایش برای آنها . . . . .	۷۸
۲-۵ مشخصه‌ها و زمان آموزش مدل‌های جدول ۱-۴ در فرایند آموزش. . . . .	۷۹
۳-۵ نحوه و درصدهای تقسیم مجموعه داده به مجموعه‌های آموزش، ارزیابی و آزمون . . . . .	۸۲
۴-۵ سرگشتگی، دقت و خطای مدل‌های مختلف روی مجموعه‌های آموزش و ارزیابی . . . . .	۸۶
۵-۵ مقادیر مجاز و مقادیر داده شده به ثبات و ورودی‌های الگوریتم‌های فازی عصبی داده و فاز عصبی فرا داده در هنگام آزمون فازی قالب فایل PDF . . . . .	۹۵
۶-۵ نتایج پوشش کد حاصل از آزمون فازی نرم‌افزار MuPDF با الگوریتم‌های تولید داده آزمون مختلف . . . . .	۹۵
۷-۵ میزان بهبود پوشش کد ابزار MuPDF توسط الگوریتم‌های روش پیشنهادی در مقایسه با کارهای مرتبط. . . . .	۹۵
ب-۱ بسته‌های نرم‌افزاری مورد نیاز جهت اجرای صحیح برنامه IUST DeepFuzz . . . . .	۱۱۹

# فهرست الگوریتم‌ها

۴۱	Basic-Random Fuzzing	۱_۳
۴۲	AFL Fuzzing	۲_۳
۴۴	Augmented-AFL Fuzzing	۲_۳
۴۹	SampleFuzz	۴_۳
۶۴	ProduceTrainingSamples	۱_۴
۷۱	DataNeuralFuzz	۲_۴
۷۲	MetadataNeuralFuzz	۲_۴

# فهرست کوتاه نوشت‌ها

## A

ASCII .....	American Standard Code for Information Interchange
ASE .....	Automated Software Engineering

## B

BPTT .....	Back Propagation Through Time
------------	-------------------------------

## C

CAN .....	Creative Adversarial Networks
CFG .....	Control Flow Graph
CLI .....	Command Line Interface
CNN .....	Convolutional Neural Network

## D

DAG .....	Directed Acyclic Graph
DFG .....	Data Flow Graph

ص

## G

GAN .....	Generative Adversarial Network
GRU .....	Gated Recurrent Units
GUI .....	Graphical User Interface

## I

ISRT .....	Internet Security Threat Report
------------	---------------------------------

## L

LM .....	Language Model
LSTM .....	Long-Short Term Memory

## M

MOU .....	Multiple Objects Update
-----------	-------------------------

## N

NLM .....	Neural Language Model
NLP .....	Natural Language Processing

## P

PDF .....	Portable Document Format
-----------	--------------------------

## فهرست کوتاه‌نوشت‌ها

ض

### R

RNN ..... Recurrent Neural Network

### S

SDL ..... Security Development Lifecycle

SOU ..... Single Object Update

SUT ..... Software Under Test

# فصل ۱

## مقدمه

«من می‌گوییم، امنیت، بالاترین اولویت ماست؛ زیرا برای همه چیزهای هیجان‌انگیزی که شما قادر به انجام دادن آن با کامپیوترها هستید – سازمان‌دهی زندگی‌تان، در ارتباط ماندن با دیگران، خلاق بودن – اگر ما مسائل امنیتی را حل نکنیم، مردم از همه این‌ها عقب خواهند ماند.»

---

✿ بیل گیتس

### ۱-۱ پیش‌زمینه

در حوزه مهندسی نرم‌افزار خودکار (ASE<sup>۱</sup>)، یکی از زمینه‌های مورد مطالعه و پژوهش، خودکارسازی فرایند آزمون نرم‌افزار، به عنوان یکی از مراحل مهم توسعه و ساخت یک سیستم نرم‌افزاری است. به طور کلی هدف از خودکارسازی، کاهش هزینه و زمان و افزایش دقت در اجرای یک فرایند است. آزمون فازی<sup>۲</sup> یکی از فنون آزمون خودکار نرم‌افزار است. آزمون فازی در یافتن خطای<sup>۳</sup> ها و آسیب‌پذیری<sup>۴</sup> ها در نرم‌افزارهای دنیای واقعی مانند مروگرهای وب، ویرایشگرهای متن، پخش‌کننده‌های چندرسانه‌ای و غیره، بسیار مؤثر واقع شده است [۱، ۲]. در این فن ورودی‌هایی بدشکل<sup>۵</sup> توسط یک برنامه دیگر، یعنی با روش خودکار، تولید شده و

---

<sup>۱</sup>Automated Software Engineering

<sup>۲</sup>Fuzz Testing

<sup>۳</sup>Fault

<sup>۴</sup>Vulnerability

<sup>۵</sup>Malformed

به نرم افزار تحت آزمون (SUT<sup>۱</sup>) تزریق می شود. SUT در عین حال، به امید یافتن خطأ بر اثر پردازش ورودی تزریق شده، پایش<sup>۲</sup> می شود. ورودی تولید شده که به برنامه داده می شود، نقش داده آزمون<sup>۳</sup> را داشته و عامل اصلی نمایان سازی خطأ(های) احتمالی موجود در برنامه با بردن آن به یک حالت خرابی<sup>۴</sup> است. به همین علت مهم ترین مرحله در فرایند آزمون فازی را می توان تولید خودکار داده های آزمون دانست، به نحوی که بیشترین خطأها، ایرادها و آسیب پذیری ها شناسایی گردند.

## ۱-۲ شرح مسئله

راهکارهای مطرح در فن آزمون فازی [۳-۶]، برای شناسایی خطأها و آسیب پذیری ها نیازمند تولید تعداد زیادی داده آزمون هستند. در نرم افزارهایی با ساختار ورودی ساده، تولید داده آزمون نیز ساده است. برای مثال می توان با روش تصادفی این کار را انجام داد. اما در نرم افزارهایی با ساختار ورودی پیچیده، مانند فایل با قالب مشخص تولید داده آزمون متنوع که بتواند مسیرهای اجرایی بیشتری را پوشش دهد، کار آسانی نیست. تعداد و عمق مسیرهای اجرایی در یک برنامه با ساختار ورودی پیچیده به مراتب بیشتر از یک برنامه با ساختار ورودی ساده است. بررسی ها نشان می دهد بسیاری از داده های آزمون تولید شده برای چنین نرم افزارهایی، مسیرهای یکسان و سطحی (کم عمق) را می پیمایند [۷] و در مجموع، آزمون های فازی معمول پوشش کد ضعیفی دارند [۸]. در صد بالایی از داده های آزمون ساخته شده به صورت تصادفی، از لحاظ ساختاری کاملاً نامعتبر هستند و در همان مراحل اولیه بررسی صحت فایل، به وسیله تجزیه گر<sup>۵</sup> ورودی برنامه هدف، رد می شوند [۹، ۷]. در چنین شرایطی، قادر به نفوذ به عمق برنامه، کشف و آزمایش مسیرهای جدید نخواهیم بود. در واقع این نوع ورودی ها به نوعی تکراری و هدر رفته محسوب می گردند.

برای حل مسائل بالا، داده آزمون را با استفاده از قالب یا گرامر ورودی تولید می کنند، روش هایی مثل [۱۰]. قالب یا گرامر اما به صورت دستی و از روی مستندات تهیه می شود که با توجه به پیچیده بودن ساختار آن، عملی زمان بر، پرهزینه و مستعد خطأ است [۱۱]. همچنین مستندات ساختار ورودی همواره در دسترس آزمون گر نیست. با این اوصاف روش مذکور تا به امروز، یکی از مؤثر ترین روش های آزمون و یافتن خطأ در برنامه هایی مانند مروگر های وب بوده، که ساختار ورودی آن فایل هایی با قالب های متنوع و پیچیده هستند [۱۲، ۱۱]. به همین جهت، ارایه روشی برای خودکار سازی تولید داده آزمون بر مبنای قالب ورودی ارزشمند و حائز اهمیت

<sup>1</sup>Software Under Test

<sup>2</sup>Monitor

<sup>3</sup>Test Data

<sup>4</sup>Failure

<sup>5</sup>Parser

است. پیش از ارایه یک روش جدید در ادامه ابتدا مسئله را دقیق‌تر تبیین کرده و راه حل‌های قبلی و نارسانایی‌های هریک از آنها را مطالعه می‌کنیم.

## ۱-۲-۱ شهود اولیه

برای روشن شدن مسئله و شناسایی مشکلات موجود در تولید داده آزمون، مسئله را به زیر مسائل کوچک‌تر شکسته و از زوایایی گوناگون تشریح می‌کنیم. ساختار پیچیده ورودی، ساختار پیچیده کد و تمایز داده و فراداده<sup>۱</sup> سه زیر مسئله‌ای هستند که ما آنها را شناسایی کرده و در این بخش، مطرح می‌کنیم. در ادامه این پایان‌نامه تمرکز خود را بر روی حل این مسائل منعطف خواهیم کرد.

### ساختار پیچیده ورودی

نخستین مورد حائز اهمیت ساختار ورودی برنامه است. در برنامه‌هایی با ورودی خط فرمان (CLI)<sup>2</sup> ساختارها به نسبت ساده هستند. اما برنامه‌هایی با ورودی فایل، ساختار ورودی بسیار پیچیده‌تری دارند. در واقع بسته به کاربرد، آنها یک یا چندین قالب فایل تعریف شده را پشتیبانی می‌کنند. همچنین برای یک قالب فایل شناخته‌شده ممکن است چندین نرم‌افزار وجود داشته باشد. یعنی در حالت کلی یک ارتباط چندبه‌چند بین قالب فایل ورودی و نرم‌افزار وجود دارد. هنگامی که یک نرم‌افزار برای آزمون انتخاب می‌شود هریک از قالب‌های فایلی که پشتیبانی می‌کند بخشی از کد نرم‌افزار را اجرا خواهد کرد.

فایل<sup>3</sup> PDF را می‌توان نمونه‌ای از یک ورودی پیچیده برای نرم‌افزارهای PDF‌خوان، مثل اغلب مرورگرهای وب، تلقی کرد. مجموعه اسناد توصیف کننده مشخصه‌ها<sup>۴</sup> ای کامل قالب PDF بیش از ۱۳۰ صفحه است [۱۱]. جزئیات ساختار این قالب را در پیوست آ<sup>۵</sup> بیان کرده‌ایم. در ساختارهای پیچیده هر بایت و در مواردی هر بیت نقش ویژه‌ای ایفا می‌کند که تولید تصادفی آنها تنوعی در پوشش کد برنامه ایجاد نمی‌کند؛ زیرا اغلب در دام کدهای کنترل استثنای<sup>۶</sup> می‌افتد. بنابراین داشتن یک درک حداقلی از ساختار در هنگام تولید داده جدید بسیار کمک کننده خواهد بود. چگونگی کسب این درک به صورت خودکار مسئله‌ای است که بایستی حل شود.

<sup>1</sup> Metadata

<sup>2</sup> Command Line Interface

<sup>3</sup> Portable Document Format

<sup>4</sup> Specification

<sup>5</sup> Exception Handling

### ساختار پیچیده کد

پیچیده بودن ساختار ورودی، منجر به پیچیده شدن کد اجرایی و در نتیجه ممانعت از پوشش کد بالا در آزمون فازی خواهد شد. برای درک بهتر این مسئله برنامه <sup>۱</sup>-۱ به زبان C را درنظر می‌گیریم. به خاطر سرعت بالای اجرا، بیشتر تجزیه‌گرهای قالب‌های پیچیده به این زبان نوشته می‌شوند. این برنامه یک فایل را از ورودی خوانده و براساس بایت‌های مشخصی در آدرس نسبی آن، مسیرهای معینی را اجرا می‌کند. چندین نکته قابل توجه در قطعه کد مذکور وجود دارد [۷] :

۱. بایت‌های جادویی<sup>۱</sup>: بایت دوم و بایت اول ابتدا برای اعتبارسنجی ورودی با مقادیر ثابتی مقایسه می‌شوند. اگر نتیجه این مقایسه صحیح نباشد؛ ورودی درجا رد می‌شود. در سطر ۱۳ این مثال ابتدا آدرسی نسبی ۱ با مقدار ۰xEF و سپس آدرس نسبی ۰ با مقدار ۰xFD مقایسه می‌گردد. بایت‌های جادویی در قالب‌های فایل بسیاری وجود دارند. از جمله قالب فایل jpeg که در ابزار <sup>۲</sup>djpeg با همین روش، اعتبارسنجی می‌شود.

۲. شرط‌های تودرتو: در اجرای برنامه، هر مسیر اجرایی مهم است. هرچند رسیدن به برخی مسیرها ممکن است دشوارتر باشد یا حتی امکان‌پذیر نباشد [۱۳]. در این مثال برای رسیدن به خط ۱۸ کد بایستی همه شرایط موجود در خط ۱۷ برقرار باشد که بهنوبه خود نیاز هست تا شرط موجود در خط ۱۵ نیز برقرار شد و به همین ترتیب. لذا داده آزمون تولیدی باید تا حد زیادی معتبر باشد تا بتواند به عمق مدنظر دسترسی پیدا کند.

۳. نشانگر<sup>۳</sup>ها: برای رسیدن به کد خط‌دادار در سطر ۱۹ بایستی شرط سطر ۱۸ ارضاء شود. این مقایسه یک توالی از نشانه‌ها انجام می‌شود که آدرس نسبی شروع آن لزوماً ثابت نیست؛ البته در این مثال ثابت نشان داده شده است. در قالب‌های فایل‌هایی مانند png، jpeg و gif این قبیل نشانگرها دیده می‌شود.

۴. آدرس‌های نسبی متغیر: برای رسیدن به مسیر اجرایی سطر ۱۸ یک مقایسه برمبانی آدرس‌های نسبی در سطر ۱۷ انجام می‌شود. آدرس نسبی به کار رفته در این مقایسه‌ها، از ورودی خوانده شده یا داخل برنامه محاسبه شده‌اند و بنابراین ممکن است که در هر بار اجرا متفاوت باشند. این امر برخلاف مورد بایت‌های جادویی است که آدرس نسبی ثابتی دارند.

<sup>1</sup>Magic Bytes

<sup>2</sup><https://linux.die.net/man/1/djpeg>

<sup>3</sup>Markers

```

1 #include <stdio.h>
2 void main(int argc, char *argv[]){
3     unsigned char buffer[1024]; //Fixed size buffer
4     int fd, size, i, j;
5     /* Some initialization here */
6     if((fd = open(argv[1], O_RDONLY)) == -1)
7         exit(0);
8     fstat(fd, &s);
9     size = s.st_size;
10    if(size > 1024)
11        return -1;
12    read(fd, buffer, size);
13    if(buffer[1] == 0xEF && buffer[0] == 0xFD) //Complex
logic expression
14        printf("Magic bytes matched!\n");
15    else
16        EXIT_ERROR("Invalid input file\n");
17    if(buffer[i] == '%' && buffer[j] == '$'){
18        if(strcmp(&buffer[15], "MAZE", 4) == 0) //Nested
condition
19            /* Codes contain bug here */
20        else{
21            /* *** Render file here (lines of code) *** */
22            close(fd);
23            return 0;
24        }
25    }else{
26        EXIT_ERROR("Invalid bytes");
27        close(fd);
28        return 0;
29    }
30    close(fd);
31 }
```

برنامه ۱-۱: یک قطعه کد به عنوان نمونه‌ای از نرم‌افزار تحت آزمون در این پایان‌نامه، با ساختار تودرتو که چالش‌های پیچیدگی برنامه تحت آزمون و پوشش کد در آزمون فازی قالب فایل را نشان می‌دهد [V] (با تغییر).

## تمایز داده و فراداده

برنامه مبتنی بر ورودی فایل، به طور معمول دو گام مجزا را برای پردازش یک فایل طی می‌کند: گام اول تجزیه<sup>۱</sup> فایل و گام دوم پرداخت<sup>۲</sup> آن. در مرحله تجزیه، فایل در حافظه بارگذاری، مقادیر فیلدهای آن خوانده شده و تبدیل به داده‌ساختارهای داخل حافظه اصلی (مثل بافر، ساختمان یا رکورد، آرایه و غیره) می‌شود. در این مرحله چنان‌چه اشکال<sup>۳</sup> نحوی در ساختار فایل باشد (فایل از مشخصه‌های قالب خود پیروی نکند)، باید توسط تجزیه‌گر تشخیص داده شود و گرنه منجر به اشکال فساد حافظه<sup>۴</sup> و خرابی برنامه می‌شود. در مرحله پرداخت، برنامه روی اطلاعات خوانده شده از فایل پردازش لازم را انجام می‌دهد و خروجی تولید می‌کند (مثلاً نمایش یک تصویر روی صفحه نمایش یا اجرای یک ویدئو و غیره) [۱۴]. خطاهای این مرحله معمولاً جدی‌تر بوده و تشخیص آن نیز مشکل‌تر است، زیرا در عمق بیشتری از کد اجرایی رخ می‌دهند. جایی که داده‌های آزمون کمتری به آن دست پیدا می‌کنند.

با توجه به توضیح بالا، می‌توان یک فایل را حاوی دو دسته از مقادیر دانست: اول، مقادیری که مشخص کننده ساختار آن فایل هستند؛ برای مثال نام فیلدها. این مقادیر را فراداده یا دادگان (داده برای داده) می‌نامند. دوم، مقادیری که مشخص کننده اطلاعات هر فیلد هستند یا همان داده‌های فایل. مسئله نهفته در اینجا آن است که رویکرد آشکارسازی خطای برای هر کدام از این قسمت‌ها متفاوت خواهد بود؛ چراکه طبیعت خطاهای هر قسمت با یکدیگر متفاوت بوده و همان‌طور که گفته شد در مراحل مختلفی هم روی می‌دهند. برای آشکار کردن خطاهای تجزیه‌گر، لازم است تا فایل‌هایی تولید کنیم که بخش فراداده آن بدشکل شده‌اند در حالی که برای آشکار کردن خطاهای بخش پرداخت، بایستی فایل‌هایی تولید کنیم که از لحاظ نحوی معتر بوده و بخش داده آن بدشکل شده باشند. شرط لازم هر دو نوع بدشکل‌سازی داشتن سازوکاری برای تشخیص داده و فراداده از یکدیگر، در هنگام تولید داده‌های آزمون است.

همان‌طور که گفتیم، رسیدن به کدهای مرحله پرداخت یک فایل (منظور تزریق داده آزمونی است که منجر به اجرای آن شود) سخت‌تر است. در برنامه ۱-۱، فرض شده است که به عنوان مثال پرداخت فایل در خط ۲۱ انجام می‌شود؛ یعنی، بعد از گذشتن از تمامی شرایط و بررسی‌های انجام شده توسط تجزیه‌گر و انتقال فیلدهای داخلی فایل به حافظه اصلی (فیلدهایی مثل فیلد size در برنامه مذکور). هر داده آزمونی که یکی از شرایط قبل از خط ۲۱ را نداشته باشد، رد شده و آن اجرا از برنامه به اجرای خط ۲۱ منتهی نمی‌گردد. برای آن که درصد خوبی از داده‌های آزمون تولید شده به اجرای خط ۲۱ منجر شوند، بایستی یک فایل تقریباً معتر و

<sup>1</sup>Parse

<sup>2</sup>Render

<sup>3</sup>Error

<sup>4</sup>Memory Corruption

پیروی کننده از قواعد قالب فایل مورد انتظار برنامه ۱-۱ را به عنوان داده آزمون تولید کرد. آنچه از شهود داده شده در این قسمت نتیجه می‌شود آن است که از یک برنامه قابل اطمینان و غیر قابل نفوذ، انتظار می‌رود که تحت هیچ عنوان بر اثر پردازش یک ورودی دچار خطا نشود. تنها زمانی می‌توان این ادعا را داشت که مطمئن شویم برنامه ورودی‌های به اندازه کافی متنوع را پردازش کرده و در هیچکدام از آنها دچار خطا نشده است. ورودی‌ها بایستی قادر به اجرای بخش‌های زیادی از کد برنامه باشند. حالت ایده‌آل اجرای تمام کد یک برنامه پیچیده است.

## ۲-۲-۱ کارهای مرتبط

تعدادی کار در ارتباط با استخراج خودکار گرامر فایل انجام شده‌اند. Bastani و همکاران [۱۵] الگوریتمی برای تولید یک گرامر مستقل از متن روی یک مجموعه از ورودی‌های نمونه داده شده ارایه کرده‌اند، که در نهایت برای تولید داده‌های جدید مورد نیاز آزمون فازی استفاده می‌شود. این الگوریتم یک مجموعه از مراحل تعمیم‌پذیری را با معرفی ساختارهای تکراری و متناوب برای عبارت‌های منظم به‌کار می‌بندد و غیر پایانه‌ها را برای گرامر مستقل از متن در هم ادغام می‌نماید که به‌نوبه خود یک گرامر یکنواخت از زبان ورودی به‌دست می‌دهد؛ اما، این روش برای قالب‌هایی مثل PDF که ساختار مسطح (غیر تو در تو) ولی در عین حال محتوای مختلفی از انواع و جفت‌های کلید-مقدار دارند، مناسب نیست [۱۱].

AUTOGRAM [۱۶] نیز به صورت غیر-احتمالاتی یک گرامر مستقل از متن را یاد می‌گیرد. یک مجموعه ورودی داده شده و به صورت پویا مشخص می‌شود که چگونه ورودی‌ها در برنامه پردازش می‌شوند. در واقع برنامه تحت آزمون با آلدگی پویا مشاهده می‌شود که حافظه را با قطعات ورودی که از آنها می‌آیند، برچسب‌گذاری می‌کند. بخش‌هایی از ورودی‌ها که توسط برنامه پردازش می‌شود، نهادهای نحوی در گرامر می‌شوند.

در پژوهش‌های اخیر تمایل زیادی به استفاده از شبکه‌های عصبی برای تحلیل و تولید برنامه‌ها به وجود آمده است. در سال ۲۰۱۷، Godefroid [۱۱] و همکاران روش جدیدی را برای تولید داده آزمون جهت استفاده در آزمون فازی بر مبنای مدل کدگذار-کدگشا<sup>۱</sup> [۱۸، ۱۷] ارایه کردند. در مقاله آنها، ساختار فایل PDF برای آزمون انتخاب شده است. ایده اصلی یادگیری یک مدل مولد روی مجموعه‌ای از ویژگی‌های اشیای داده‌ای PDF با داشتن مجموعه‌ای از نمونه‌های اولیه است. مدل کدگذار-کدگشا اجازه یادگیری متن با طول دلخواه را برای پیش‌بینی توالی بعدی کاراکترها، می‌دهد.

مدل استفاده شده توسط Godefroid و همکاران، مدل مبنایی و ظایفی مانند ترجمه ماشینی یا تبدیل گفتار

<sup>۱</sup>Encoder-Decoder Model

به نوشتار است که یادگیری ساختار فایل را نمی‌توان در این وظایف گنجاند؛ زیرا، این مدل برای نگاشت دو توالی با دامنه‌های مختلف به کار گرفته می‌شود و این در حالی است که یادگیری ساختار فایل چنین وظیفه‌ای نیست. یعنی می‌توان از مدل‌های ساده‌تری مانند مدل زبانی نیز برای یادگیری ساختار فایل استفاده کرد. روش پیشنهادی آنها، تنها ساختارهای متنی فایل را مورد یادگیری قرار می‌دهد. این در حالی است که ساختار فایل‌های پیچیده هم متنی و هم دودویی هستند. افزون بر این، الگوریتم پیشنهادی آنها برای تولید داده آزمون نیز مشکلاتی دارد. از جمله اینکه ممکن است هیچ‌گاه پایان نیابد. در فصل <sup>۳</sup>، ضمن تشریح کامل این روش، مشکلات آن را نیز به صورت کامل تری بیان می‌کنیم. همچنین در فصل <sup>۳</sup>، دو فازر قالب فایل دیگر تحت عنوان AFL <sup>[۱۹]</sup> و AFLافزووده <sup>[۲۰]</sup> که با روش‌هایی غیر از یادگیری گرامر سعی در بهبود پوشش کد SUT در فرایند آزمون فازی را دارند، نیز بررسی می‌کنیم و مشکلات آنها را بیان خواهیم کرد.

در هیچ‌کدام از کارهای قبلی، مسئله مطرح شده در ارتباط با تمایز میان داده و فراداده در هنگام آزمون فازی دیده نشده است. به عبارت دیگر، این دیدگاه به آزمون فازی قالب فایل، دیدگاهی نو است و ارزش آزمایش شدن دارد. امکان استفاده از الگوریتم ارایه شده در روش <sup>[۱۱]</sup>، برای تمایز میان داده و فراداده وجود دارد اما برای حل مابقی مشکلات، یک روش جدید را در فصل <sup>۴</sup>، پیشنهاد خواهیم داد.

## ۱-۲-۳ فرضیه‌ها و اهداف

هدف اصلی در پایان‌نامه پیش‌رو، ارائه روشی کارا جهت یافتن خطاهای آسیب‌پذیری‌ها در نرم‌افزارهایی مثل PDF‌خوان‌ها بوده که ورودی آنها فایل با ساختار مشخص و عموماً پیچیده است. در این راستا تولید خودکار فایل‌های ورودی با هدف افزایش پوشش کد<sup>۱</sup> SUT از اهمیت ویژه‌ای برخوردار است. برای نیل بدین اهداف از فنون یادگیری ژرف<sup>۲</sup> در یادگیری و درک خودکار ساختار فایل و سپس تولید فایل‌های جدید، استفاده خواهیم کرد.

چون ایده استفاده از یادگیری ماشینی در آزمون فازی جدید است، این حوزه هنوز برای پژوهشگران ناشناخته بوده و بنابراین یکی دیگر از اهداف این پایان‌نامه شناسایی، تعریف و تفکیک پارامترهای حاکم در حوزه مذکور است. به نظر می‌رسد که فنون یادگیری ماشینی راهگشای حل مسائل شرح داده شده در بخش ۱-۲ باشد. به همین جهت فراهم آوردن چارچوبی استاندارد برای شکل‌دهی به کارهای آتی، مفید و مثمر ثمر خواهد بود. به طور خلاصه ما چندین فرضیه در این پایان‌نامه درنظر داریم، که تدوین سازوکارهایی برای رد یا تأیید صحت آنها، اهدافِ ما خواهند بود:

<sup>1</sup>Code Coverage

<sup>2</sup>Deep Learning

### ۱-۳ . روش پیشنهادی و نوآوری‌ها

- استفاده از فنون یادگیری ژرف بالاخص شبکه‌های عصبی مکرر ژرف، در یادگیری خودکار ساختار فایل، امکان‌پذیر و نتیجه‌بخش است.
- خودکارسازی کامل فرایند آزمون فازی مبتنی بر گرامر با ترکیب مدل یادگیری (مدل زبانی عصبی) و روش‌های فاز (بد-شکل‌سازی) ورودی، به خوبی میسر می‌شود.
- روش‌های ترکیبی تولید داده آزمون، یعنی روش تولید مبتنی بر گرامر به همراه روش تولید مبتنی بر جابه‌جایی، منجر به افزایش پوشش کد SUT می‌گردد.
- امکان کشف خطاهای و آسیب‌پذیری‌های احتمالی موجود در SUT از طریق آزمون فازی با داده‌های آزمون تولید شده از طریق مدل‌های یادگیری ژرف، وجود دارد.

## ۱-۳ روش پیشنهادی و نوآوری‌ها

در این پایان‌نامه یک روش برای یادگیری خودکار ساختار فایل و سپس تولید داده‌های آزمون بر اساس آن ارایه می‌شود. برای یادگیری از مدل زبانی (LM<sup>1</sup>) که یک مفهوم ابتدایی در پردازش زبان طبیعی (NLP<sup>2</sup>)، است استفاده می‌کنیم. مدل زبانی را با استفاده از کلاس خاصی از شبکه عصبی ژرف<sup>3</sup> موسوم به شبکه عصبی مکرر (RNN<sup>4</sup>)، ایجاد می‌کنیم که در نتیجه به آن مدل زبانی عصبی (NLM<sup>5</sup>) هم گفته می‌شود. روش ارایه شده در اینجا، همچنین، بخش‌های غیرمنتی را نیز در آزمون فازی لحاظ می‌کند، به هیچ قالب فایل خاصی وابستگی نداشته و به سبب پیاده‌سازی با زبان پایتون قابلیت اجرا بر روی هر ماشینی را دارد.

روش پیشنهادی در فصل ۴، در سه بخش کلی ارائه شده است. بخش اول به یادگیری ساختار فایل می‌پردازد (بخش ۲-۴)، بخش دوم روشی برای تولید و بدشکل‌سازی همزمان داده‌های آزمون ارایه می‌دهد (بخش ۴-۳) و در نهایت بخش سوم یک فازر کاملاً پیمانه‌ای را معرفی می‌کند که از آن برای آزمون فازی قالب فایل استفاده خواهد شد (بخش ۴-۴). در حالی که تمرکز اصلی بر روی نحوه تولید داده‌های آزمون است، اما برای انجام آزمون فازی به ابزارهای دیگری مانند تزریق کننده داده آزمون و نیز پایش SUT جهت ثبت خطاهای رخ داده شده نیاز است. نوآوری‌های اصلی روش پیشنهادی به طور خلاصه عبارتند از:

۱. یادگیری گرامر یا ساختار یک قالب فایل با استفاده از مدل‌های زبانی عصبی،

<sup>1</sup>Language Model

<sup>2</sup>Natural Language Processing

<sup>3</sup>Deep Neural Network

<sup>4</sup>Recurrent Neural Network

<sup>5</sup>Neural Language Model

۲. تولید داده‌های آزمون متنی و دودویی همگام با بدشکل‌سازی آنها با استفاده از یک روش ترکیبی،  
 ۳. ایجاد یک فاژر قالب فایل و یک مجموعه داده آزمون برای آزمون فازی نرم‌افزارهای PDF‌خوان،  
 ۴. و بررسی و شناسایی پارامترهای مؤثر در یادگیری ساختار فایل با استفاده از فنون یادگیری ژرف.

توضیح مبسوط‌تری از نوآوری‌های و دستاوردهای این پایان‌نامه در فصل ۶، ارایه شده است. در آن فصل همچنین مزایا و معایب فنون یادگیری ژرف در یادگیری ساختار فایل و نیز مزایا و معایب روش پیشنهادی بررسی و بیان شده‌اند.

## ۱-۴ اهمیت موضوع

مانند هر محصول دیگری، نرم‌افزار نیازمند آزمون<sup>۱</sup> و راستی آزمایی است. ماهیت غیرقابل لمس و پیچیدگی ذاتی نرم‌افزار سبب می‌شود تا فرایند آزمون آن نیز متفاوت، پیچیده و پرهزینه باشد. اما این دشواری‌ها از اهمیت موضوع آزمون نمی‌کاهد. خطاهای نرم‌افزاری در مواردی سبب خسارت‌های مالی و جانی جبران ناپذیری شده‌اند. راکت آریان<sup>۲</sup> در سال ۱۹۹۶، تنها ۳۷ ثانیه پس از پرتاب منفجر شد. علت آن وقوع خطا در تبدیل نوع یک عدد ممیز شناور به عدد صحیح بود [۱۳]. وجود خطا در ماشین پرتو درمانی Therac-25<sup>۳</sup>، سبب کشته شدن دست‌کم سه انسان بر اثر تشیع بیش از حد پرتو، در سال‌های ۱۹۸۵ تا ۱۹۸۷ شد. مثال‌های دیگری از این قبیل در [۲۱، ۱۳]<sup>۴</sup> آمده است.

در مواردی وجود خطا منجر به آسیب‌پذیری می‌شود که امکان سوء استفاده و دسترسی‌های غیرمجاز را به مهاجمان<sup>۴</sup> می‌دهد. باج‌افزار WannaCrypt<sup>۵</sup><sup>۶</sup> که در نیمه اول سال ۲۰۱۷، بیش از ۲۳۰ هزار رایانه را در ۱۵۰ کشور جهان آلوده ساخت، از یک آسیب‌پذیری در هسته نسخه‌های قدیمی، سیستم عامل ویندوز شرکت مایکروسافت بهره‌برداری کرده بود. این باج‌افزار اطلاعات کاربر را رمزنگاری و برای رمزگشایی آن

<sup>1</sup>Test

<sup>2</sup>[https://en.wikipedia.org/wiki/Ariane\\_5](https://en.wikipedia.org/wiki/Ariane_5)

<sup>3</sup><https://en.wikipedia.org/wiki/Therac-25>

<sup>4</sup>Attacker

<sup>5</sup>Ransomware

<sup>6</sup><https://docs.microsoft.com/en-us/windows/security/threat-protection/wannacrypt-ransomware-worm-targets-out-of-date-systems-wdsi>

درخواست پرداخت هزینه می‌کرد. شرکت سیمنتک<sup>۱</sup> در گزارش ISRT<sup>۲</sup> خود در سال ۲۰۱۸<sup>۳</sup> [۲۲]، افزایش ۶۰۰ درصدی حملات در اینترنت چیزها<sup>۴</sup> (اینترنت اشیاء) و افزایش تهدیدات در دیگر حوزه‌های سایبری از جمله تلفن همراه، را اعلام کرده است. در هر حال، کشف خطأ و آسیب‌پذیری احتمالی ناشی از آن، در نرم‌افزارهایی که به طور گسترده توسط همگان مورد استفاده قرار می‌گیرند، مثل سیستم‌عامل‌ها، مرورگرهای وب، PDF‌خوان‌ها و غیره، بسیار حائز اهمیت است؛ زیرا، در صورت برطرف نشدن آن خطر وقوع حملات مشابه حملات بالا دور از انتظار نخواهد بود.

هنگامی که نرم‌افزارها بزرگ می‌شوند، آزمون دستی پاسخ‌گو نیست و خودکارسازی آزمون اهمیت می‌یابد. آزمون فازی همان‌طور که در ابتدای فصل بیان شد، به عنوان یک فن مؤثر آزمون نرم‌افزار در شناسایی خطاهای حافظه و آسیب‌پذیری‌ها شناخته شده است. برای مثال چرخه حیات امن نرم‌افزار (SDL)<sup>۵</sup> شرکت مايكروسافت<sup>۶</sup>، در مرحله درستی‌یابی<sup>۷</sup>، استفاده از آزمون فازی را به عنوان یک روش استاندارد، اجباری می‌کند [۲۳]. آزمون فازی جعبه سفید<sup>۸</sup> (رجوع کنید به بخش ۲-۱-۲)، حدود یک سوم کل آسیب‌پذیری‌های شناخته شده در سیستم عامل ویندوز ۷ این شرکت را کشف کرده است [۱۰]. شرکت گوگل در سال ۲۰۱۲، اطلاعاتی راجع به ابزار ClusterFuzz خود منتشر کرد که از آن برای آزمون فازی پروژه‌های Chromium<sup>۹</sup> (شامل OS و مرورگر وب Chromium) استفاده می‌کند [۱۲]. این شرکت همچنین به افرادی که موفق به کشف آسیب‌پذیری در پروژه‌های نامبرده شوند، جوازی اهدا می‌کند.

تولید داده آزمون را بایستی مهم‌ترین مرحله در آزمون فازی دانست؛ چراکه داده‌هایی که نرم‌افزار با آنها آزمون می‌شود عامل اصلی اجرا شدن کدهای قسمت‌های مختلف SUT است و در صورتی که خطای در آنها وجود داشته باشد، تنها از این طریق است که خود را نشان می‌دهد. البته باید بدین مسئله توجه کرد که اجرای کد خطادار شرط لازم برای آشکارسازی خطأ است ولی کافی نیست و روش تولید داده‌های آزمون، می‌بایست شرایط خاص بدلشکل بودن را نیز محیا کند. تولید داده مبتنی بر گرامر، مؤثرترین روش آزمون برنامه‌هایی با ساختار ورودی پیچیده است [۲۴]. موارد بیان شده در این بخش، به خوبی اهمیت موضوع تولید خودکار داده آزمون در آزمون فازی و لزوم ارایه روش‌های جدید را توجیه کرده و انگیزه کافی را برای پژوهش در این

<sup>1</sup><https://www.symantec.com/>

<sup>2</sup>Internet Security Threat Report

<sup>3</sup><https://www.symantec.com/security-center/threat-report>

<sup>4</sup>Internet of Things

<sup>5</sup>Security Development Lifecycle

<sup>6</sup><https://www.microsoft.com/en-us/sdl>

<sup>7</sup>Verification

<sup>8</sup>White Box

<sup>9</sup><https://www.chromium.org/>

زمینه ایجاد می کنند.

## ۱-۵ ساختار پایان نامه

این پایان نامه در شش فصل و دو پیوست تنظیم شده است و ساختار ادامه آن به قرار زیر است. در فصل ۲ ادبیات موضوع شامل آزمون نرم افزار، آزمون فازی و یادگیری ژرف را مطرح می کنیم. در این فصل ابتدا معیارهای سنجش کیفیت آزمون و چگونگی محاسبه آنها را توضیح داده، سپس به معرفی آزمون فازی، فرایند کلی و روش های تولید داده آزمون در آن می پردازیم. در بخش پایانی مباحث یادگیری ژرف را با تمرکز بر مفاهیم مرتبط با یادگیری ساختار فایل، عنوان خواهیم کرد.

در فصل ۳ به پیشنهاد پژوهش و بیان کارهای مرتبط در تولید خودکار داده آزمون و نقد و بررسی آنها می پردازیم. به طور خلاصه برخی راه حل های دیگران برای مسائل مطرح شده در بخش ۱-۲ را معرفی و سپس مشکلات آنها را بیان می کنیم. روش پیشنهادی در راستای حل این مسائل و ارزیابی ما در مقایسه با نتایج ارائه شده قبلی در این فصل خواهد بود.

در فصل ۴ روش پیشنهادی خود را برای تولید داده آزمون مطرح می کنیم. روش پیشنهادی در این فصل، همان طور که بدان اشاره شد، یک روش تولید مبتنی بر مدل های زبانی است که ما جایه جایی های تصادفی را نیز به آن اضافه کرده و روشی ترکیبی خلق نموده ایم. در همین فصل، ما دو الگوریتم جدید را برای فاز داده های آزمون معرفی می کنیم.

در فصل ۵ معیارهای ارزیابی روش پیشنهادی، چیدمان آزمایش ها و نتایج حاصل از اجرای آنها را ذکر خواهیم کرد. مورد مطالعاتی ما در آزمایش های این فصل نرم افزار MuPDF<sup>۱</sup> [۲۵] و قالب فایل PDF است که در ابتدای فصل آنها را مختصر معرفی خواهیم کرد.

در نهایت فصل ۶ را به بیان نتیجه گیری، یافته ها و نوآوری های پایان نامه، محدودیت های روش پیشنهادی و کارهای قابل انجام در آینده اختصاص داده ایم. همچنین در پیوست آ ساختار فایل PDF و در پیوست ب جزئیات پیاده سازی محصلو نهایی پایان نامه را درج کرده ایم.

<sup>1</sup><https://mupdf.com/>

## فصل ۲

### مفاهیم اولیه

«آزمون نرم‌افزار می‌تواند وجود خطاهای را نشان دهد، اما هرگز نمی‌تواند  
نبود آنها را تضمین کند.»

#### ✳ ادسخر دایکسترا

در بخش اول این فصل تعاریف پایه‌ای آزمون نرم‌افزار و معیارهای سنجش آزمون، روش اندازه‌گیری و گزارش آنها را بیان می‌کنیم. در بخش دوم، آزمون فازی، فنون مختلف به کار گرفته شده در این آزمون، آزمون فازی قالب فایل، روش‌های تولید خودکار داده آزمون در فازر<sup>۱</sup>‌ها و معماری فازرها را توضیح می‌دهیم. بخش سوم و چهارم را به معرفی فنون یادگیری ژرف و استفاده از آنها در وظایف یادگیری مبتنی بر توالی<sup>۲</sup> تخصیص داده‌ایم. در پایان نیز خلاصه‌ای از مطالب مطرح شده در فصل را می‌آوریم. مفاهیم اولیه بسیار گستردگر از آنچه در این فصل مطرح گردیده هستند و به همین دلیل در بخش پایانی خواننده را به منابع مناسبی ارجاع داده‌ایم. همچنین مطالعه کامل‌تری در باب مطالب این فصل در گزارش سمینار کارشناسی ارشد اینجانب [۲۶] انجام شده است.

---

<sup>1</sup>Fuzzer

<sup>2</sup>Sequence

## ۱-۲ آزمون نرم‌افزار

مجموعه فنون کشف و آشکارسازی خرابی‌های نرم‌افزار در مراحل مختلف توسعه آن را آزمون نرم‌افزار گویند. منظور از خرابی بروز رفتار(های) ناخواسته و خلاف مشخصه‌ها در یک نرم‌افزار یا قسمتی از آن است. خرابی خود حاصل یک خطأ (نقص) ایستا در نرم‌افزار است. حالت داخلی نادرست برنامه را که ناشی از یک خطأ است، اشکال می‌گویند [۱۳]. واژه‌های خطأ، اشکال و خرابی از حوزه اتکاپذیری<sup>۱</sup> وارد آزمون نرم‌افزار شده‌اند [۲۱] و تعاریف یکسانی برای آنها وجود ندارد [۲۷، ۱۳].

آزمون نرم‌افزار منحصر به کد اجرایی برنامه نیست و شامل آزمون نیازمندی<sup>۲</sup> ها، آزمون مشخصه‌ها و طراحی<sup>۳</sup> نیز می‌گردد. در آزمون فازی اما مُنحصاراً به کد اجرایی پرداخته می‌شود. لذا در ادامه این پایان‌نامه، منظور از آزمون، فقط آزمون مربوط به کد اجرایی برنامه خواهد بود. واژگان حاکم بر حوزه آزمون نرم‌افزار بسیار گسترده می‌باشد. در اینجا روی برخی از مفاهیم پایه‌ای و مرتبط با آزمون فازی و تولید داده آزمون تمرکز می‌کنیم.

### ۱-۱-۲ معیارهای پوشش

متأسفانه آزمون تنها قادر است وجود خرابی را نشان دهد و نبود آن را تضمین نمی‌کند. بهبیان صوری، مسئله یافتن تمامی خرابی‌ها در یک برنامه تصمیم‌ناپذیر<sup>۴</sup> است [۱۳]. این موضوع سبب می‌شود به دنبال راهکارهایی برای اندازه‌گیری آزمون و تعیین میزان خوب بودن آن باشیم. در دهه‌های اخیر تمرکز بر روی معیارهای پوشش<sup>۵</sup>، شناسایی، تفکیک و سنجش آنها بوده است. معیارهای پوشش بیشتر با هدف کمی‌سازی و اندازه‌گیری مقدار آزمون انجام شده مطرح شده‌اند؛ هرچند متقابلاً در تولید داده آزمون هم کاربرد دارند. چون با مسئله‌ای تصمیم‌ناپذیر موافق هستیم، بایستی معیاری وجود داشته باشد که مشخص کند چه زمانی می‌توانیم آزمون را خاتمه دهیم و در این زمان آزمون تا چه حد خوب انجام شده است. آمان<sup>۶</sup> و آفوت<sup>۷</sup> [۱۳] معیارهای پوشش

<sup>1</sup>Dependability

<sup>2</sup>Requirement

<sup>3</sup>Design

<sup>4</sup>Undecidable

<sup>5</sup>Coverage Criteria

<sup>6</sup>P.Ammann (<https://cs.gmu.edu/~pammann>)

<sup>7</sup>J. Offutt (<https://cs.gmu.edu/~offutt/>)

را به صورت چهار معیار افزای فضای ورودی<sup>۱</sup>، پوشش گراف<sup>۲</sup>، پوشش منطق<sup>۳</sup> و پوشش ساختار نحوی<sup>۴</sup> مطرح کردند.

پوشش گراف در سطح کد اجرایی که به آن پوشش کد هم گفته می‌شود، شامل پوشش گراف جریان کنترل (CFG<sup>۵</sup>) و گراف جریان داده (DFG<sup>۶</sup>) برنامه می‌شود. پوشش منطق، مقداردهی و تعیین ارزش عبارات منطقی ظاهر شده در متن برنامه است. افزای فضای ورودی یعنی انتخاب از بین حالت‌های مختلف ترکیب ورودی‌ها و در نهایت پوشش ساختار نحوی، استفاده از قوانین گرامر برای اعتبارسنجی<sup>۷</sup> داده‌های ورودی یا تولید داده‌های جدید آزمون را شامل می‌شود.

معیارهای پوشش به دو روش مختلف قابل استفاده هستند. یک روش به این صورت است که مقادیر داده‌های آزمون منحصراً برای برآوردن یک معیار داده شده، تولید گرددند. این روش در برخی موارد بسیار دشوار است. بهویژه اگر ابزاری برای تولید خودکار داده آزمون نداشته باشیم یا ساختار ورودی پیچیده باشد. روش دوم تولید داده‌های آزمون به طور کاملاً مجرزا و سپس اندازه‌گیری میزان پوشش معیار مربوطه است. برای روش اول یک برنامه شناسنده<sup>۸</sup> برآورده شدن معیار و برای روش دوم یک برنامه مولد<sup>۹</sup> داده آزمون نیاز است. در صنعت استفاده از روش دوم مرسوم‌تر است [۱۳].

در این پایان‌نامه، معیار دوم از معیارهای پوشش (پوشش کد) با روش دوم را مبنا قرار می‌دهیم. به این صورت که یک روش تولید خودکار داده آزمون و اندازه‌گیری میزان پوشش کد ارایه خواهیم داد. این فن آزمون، به‌خوبی عملی بوده و آزمون فازی معمول نیز بر اساس این روش است. پوشش کد خود در سطوح مختلفی مطرح است؛ پوشش دستور<sup>۱۰</sup>، پوشش شاخه<sup>۱۱</sup> و پوشش مسیر<sup>۱۲</sup> سه سطح شناخته شده هستند. در پوشش دستور اجرای حداقل یک مرتبه هر دستور برنامه به عنوان نیازمندی تعریف و سپس اندازه‌گیری می‌شود. در پوشش شاخه اجرای حداقل یک مرتبه هر انشعاب برنامه به عنوان نیازمندی مطرح است و بالآخره در پوشش مسیر اجرای حداقل یک مرتبه هر مسیر اجرایی مدققت است. بدیهی است که پوشش مسیر، پوشش شاخه و

<sup>1</sup>Input Space Partitioning

<sup>2</sup>Graph Coverage

<sup>3</sup>Logic Coverage

<sup>4</sup>Syntax-Based Coverage

<sup>5</sup>Control Flow Graph

<sup>6</sup>Data Flow Graph

<sup>7</sup>Validation

<sup>8</sup>Recognizer Program

<sup>9</sup>Generator Program

<sup>10</sup>Statement Coverage

<sup>11</sup>Branch Coverage

<sup>12</sup>Path Coverage

پوشش شاخه، پوشش دستور را شامل می‌شود. تعاریف و فهرست کاملی از سطوح پوشش کد و رابطه بین آنها در [۱۲] ذکر شده است.

شرکت مایکروسافت در محیط توسعه مجتمع ویژوال استادیو<sup>۱</sup>، معیارهای پوشش خط<sup>۲</sup>، پوشش بلوک پایه<sup>۳</sup> و پوشش خط جزئی<sup>۴</sup> را ارایه کرده است. پوشش خط همان پوشش دستور در کد سطح بالا (بهجای کد اسمبلی/ماشین) است. پوشش بلوک پایه تعیینی از پوشش دستور است که در آن یک توالی از دستورهای برنامه که شامل هیچ دستور پرشی در بین خود نیستند، یک دستور واحد در نظر گرفته می‌شوند. مزیت پوشش کد در سطح دستور و بلوک پایه<sup>۵</sup> سادگی اندازه‌گیری آن و عدم نیاز به وجود کد منبع<sup>۶</sup> برنامه است [۲۱]. پوشش خط جزئی منظور اجرا شدن برخی از قسمت‌های یک خط کد سطح بالا است. برای مثال در خطوطی با عبارات شرطی طولانی بر اثر ارزیابی مدار کوتاه<sup>۷</sup> ممکن است کلیه دستورات آن خط کد اجرا نگردد. در نتیجه پوشش خط جزئی معیاری متفاوت از پوشش خط است.

علاوه بر موارد ذکر شده معیارهای دیگری هم برای پوشش کد ارایه گردیده، از جمله پوشش دامنه<sup>۸</sup> که در آزمایشگاه تحقیقاتی مهندسی معکوس<sup>۹</sup> دانشگاه علم و صنعت ایران<sup>۱۰</sup> توسعه داده شده است [۲۸]. پوشش دامنه، نوعی از معیار افزار فضای حالت [۱۳] است که در آن با استناد به تئوری مجموعه‌ها، دامنه‌هایی برای هریک از ورودی‌های برنامه تعیین می‌شود. انتخاب ورودی از این دامنه‌ها در زمان آزمون، منجر به پوشش یک مسیر مشخص شده می‌گردد. از آنجایی که تنها یک بار اجرای یک مسیر، لزوماً خطای موجود در آن را آشکار نمی‌کند؛ در این روش می‌توان مسیر داده شده را به هر تعداد و هر بار با داده‌های آزمون متفاوت آزمون کرد. از این دیدگاه، پوشش دامنه یک روش تولید داده آزمون را ارایه می‌دهد.

در این پایان‌نامه، پوشش بلوک پایه و پوشش خط را برای گزارش میزان پوشش کد به کار می‌بریم. در بین این دو معیار، پوشش بلوک پایه در مقایسه با پوشش خط برنامه معیار مناسب‌تری است؛ زیرا، تعداد خطوط برنامه را می‌توان تغییر داد. برای مثال چندین دستور را در یک خط قرار داد یا اینکه یک دستور را در چند خط نوشت. تعداد بلوک پایه اما مستقل از نحوه آرایش و قرارگیری خطوط برنامه است و این معیار با تغییراتی مانند آنچه گفته شد، تغییر نمی‌کند؛ زیرا، در هر حالت تعداد بلوک‌های پایه یک برنامه و گراف جریان کنترل ثابت

<sup>1</sup>Visual Studio ([visualstudio.microsoft.com](http://visualstudio.microsoft.com))

<sup>2</sup>Line Coverage

<sup>3</sup>Basic Block Coverage

<sup>4</sup>Partial Line Coverage

<sup>5</sup>Basic Block

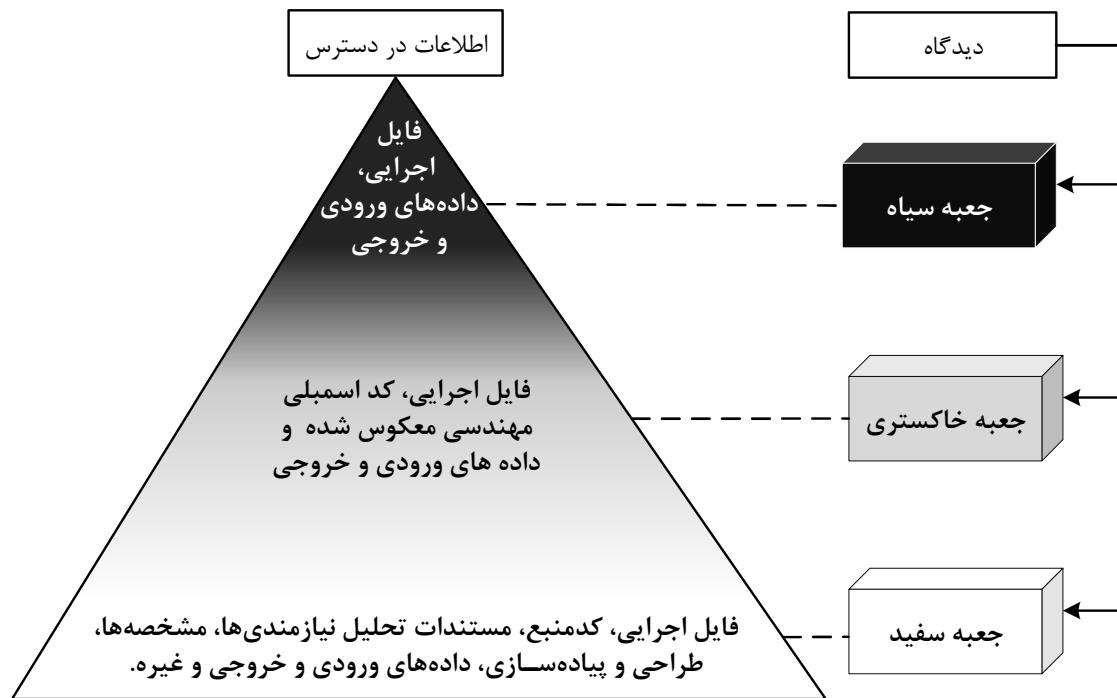
<sup>6</sup>Source Code

<sup>7</sup>Short-Circuit Evaluation

<sup>8</sup>Domain Coverage

<sup>9</sup>Reverse Engineering

<sup>10</sup>IUST Reverse Engineering Research Laboratory (<http://parsa.iust.ac.ir/reverse-engineering-lab/>)



**شکل ۲-۱:** طرح‌واره‌ای از دیدگاه جعبه بر اساس اطلاعات در دسترس به صورت یک مثلث آزمون [۲۶].

است. البته این تأکید صرفاً برای مواردی است که ممکن است آرایش خطوط برنامه در آزمون‌های مختلف تغییر کند، بدون اینکه تغییری در خود کد ایجاد شده باشد.

## ۲-۱-۲ دیدگاه جعبه

نحوه اندازه‌گیری معیارهای بحث شده، تنظیم و سنجش آزمون، منوط به اطلاعات در دسترس از SUT است. آزمون نرم‌افزار را براساس اطلاعات در دسترس از SUT در زمان آزمون، به سه دسته/<sup>۱</sup> حالت جعبه سیاه<sup>۲</sup>، جعبه سفید و جعبه خاکستری<sup>۲</sup> تقسیم‌بندی می‌کنند [۲۹، ۱۳]، که آن را دیدگاه جعبه به آزمون نرم‌افزار گوییم. شکل ۲-۱ این تقسیم‌بندی را به صورت یک طرح‌واره مثالی نشان می‌دهد.

همان‌طور که در این طرح‌واره پیداست، در رأس مثلث، هیچ اطلاعی از ساختار داخلی SUT در اختیار آزمون‌گر نیست. در واقع با برنامه به مثابه یک جعبه سیاه که ورودی را دریافت و خروجی تولید می‌کند، رفتار می‌شود. این نوع آزمون معمولاً روی برنامه‌هایی که منتشر شده‌اند و توسط افرادی خارج از تیم توسعه و پشتیبانی محصول انجام می‌شود. هدف آن هم بیشتر کشف آسیب‌پذیری‌ها و ضعف‌های امنیتی است. در پایین

<sup>۱</sup>Black Box

<sup>۲</sup>Gray Box

## ۱-۲. آزمون نرم‌افزار

مثلث، دیدگاه جعبه سفید وجود دارد. در اینجا تقریباً تمامی اطلاعات SUT در دسترس است. دیدگاه جعبه خاکستری مابین دو دیدگاه قبلی قرار دارد. در این دیدگاه از فنون مهندسی معکوس برای استخراج اطلاعات از SUT استفاده می‌شود. هریک از این دیدگاه‌ها مزایا و معایب خاص خود را دارند [۳۰]. آزمون فازی هم که در ادامه مطرح می‌شود، قابل تقسیم به این سه دسته است. آزمون فازی جعبه خاکستری، برای مثال، بسیار ثمربخش بوده است [۱۹]. روش پیشنهادی در این پایان‌نامه برای تولید داده آزمون، قابل استفاده در هر سه دسته آزمون فازی است. برای ارزیابی روش پیشنهادی در این پایان‌نامه، در [فصل ۵](#)، اما از آزمون جعبه سفید، استفاده کردایم تا بتوانیم معیارهای پوشش کد و بخش‌های اجرا شده را به دقت اندازه‌گیری نماییم. از طرفی SUT انتخابی ما متن باز و رایگان بوده و در این حالت کد منبع آن در دسترس است.

## ۳-۱-۲ اندازه‌گیری پوشش کد

هدف از مطرح کردن دیدگاه جعبه در بخش ۱-۲، اشاره به امکانات لازم برای اندازه‌گیری معیارهای پوشش از جمله پوشش کد بود. در آزمون جعبه سفید که کد منبع برنامه وجود دارد، مشاهده پوشش کد برنامه با اضافه کردن دستوراتی به متن کد به آسانی امکان پذیر است. این عمل را ابزارگذاری<sup>۱</sup> کد یا تجهیز برنامه به کد رهگیری می‌نامند [۱۴]. در محیط GNU/GCC (زبان‌های C و C++)، برای ابزارگذاری کافی است کد منبع برنامه را همراه با پرچم‌های `-fprofile-arcs` و `-ftest-coverage` کامپایل کنیم:

---

```

1  ~$ gcc -g -fprofile-arcs -ftest-coverage -o test test.c
2  ~$ ./test f1 f2
3  ~$ gcov test.c
4  ~$ more test.c.gcov

```

---

برنامه ۲-۱: ابزارگذاری کد در محیط GNU/GCC

سطر اول برنامه ۲-۱ برنامه `test` را با پرچم‌های ذکر شده کامپایل می‌کند. سطر دوم برنامه را با دو آرگومان خط فرمان `f1` و `f2` اجرا می‌کند. سطر سوم پوشش کد اجرای اخیر برنامه را محاسبه و فایلی تحت عنوان `test.c.gcov` تولید می‌کند. در سطر چهارم این فایل برای مشاهده باز می‌شود. افزون بر این، پرچم‌هایی برای اخذ انواع سطوح پوشش کدی که بحث شد، از جمله پوشش شاخه، وجود دارد [۱۴].

محیط ویژوال استادیو ابزار `vsinstr` را برای ابزارگذاری کد برنامه و `VSPerfMon` را برای اندازه‌گیری پوشش کد زبان‌های پشتیبانی شده توسط این محیط، ارائه می‌دهد که البته در حضور کد منبع برنامه قابل استفاده هستند. هنگامی که کد منبع در اختیار نباشد کار اندکی دشوار می‌شود. در این موارد، بایستی از ابزارهای

<sup>1</sup>Instrumenting

مهندسی معکوس و ابزارگذاری کد بازیزی استفاده کرد. <sup>۱</sup>DynamoRIO [۱۴] <sup>۲</sup>Pin, [۳۲] <sup>۳</sup>PaiMei, [۳۱] <sup>۴</sup>QEMU [۳۳] ابزارهایی برای این منظور فراهم کرده‌اند.

## ۲-۲ آزمون فازی

آزمون فازی [۳-۶] همان‌طور که قبلاً هم گفتیم، فرایند ساده تولید و سپس تزریق یک ورودی ناخواسته (بدشکل شده یا نامتعارف) به SUT است. چنان‌چه برنامه بر اثر پردازش این ورودی ناخواسته دچار خرابی شود، حافظه برنامه مورد تحلیل قرار گرفته و خطای احتمالی موجود در کد آشکار می‌گردد. به دلیل اینکه SUT هنگام آزمون اجرا می‌شود، آزمون فازی، نوعی آزمون پویا است. چون SUT با تعداد ورودی‌های بسیار زیادی مورد آزمون قرار می‌گیرد، آزمون فازی را می‌توان نوعی آزمون فشار<sup>۵</sup> هم به‌شمار آورد. فرایند معمول آزمون فازی در شکل ۲-۲ نشان داده شده است.

### ۲-۲-۱ فازر

فازر ابزاری است که فرایند آزمون فازی را خودکار می‌کند. پیاده‌سازی هریک از پیمانه<sup>۶</sup>‌های شکل ۲-۲ و تجمعی آنها در کنار یکدیگر یک فازر را ایجاد می‌کند. مرکز اصلی در یک فازر، نحوه تولید ورودی جدید به عنوان داده آزمون است به‌گونه‌ای که می‌توان آن را وجه تمایز اصلی فازرهای مختلف دانست. روش‌های تولید داده در فازرهای قابل تفکیک به دو دسته کلی روش‌های مبتنی بر جابه‌جایی<sup>۷</sup> یا جهش و روش‌های مبتنی بر تولید<sup>۸</sup> هستند [۲۴].

در روش مبتنی بر جابه‌جایی، تعداد یک یا بیشتر داده ورودی معتبر<sup>۹</sup> به عنوان دانه اولیه<sup>۱۰</sup> برای تولید داده‌های آزمون بیشتر به‌کار می‌رود. دانه اولیه جهش (تغییر ناگهانی) می‌یابد تا داده آزمون دیگری تولید شود. جهش می‌تواند ساده باشد؛ شبیه معکوس کردن یک بیت، جایگزین کردن یک کاراکتر، یا پیچیده‌تر باشد؛

<sup>1</sup><http://www.dynamorio.org/>

<sup>2</sup><https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>

<sup>3</sup><https://github.com/OpenRCE/paimei>

<sup>4</sup><https://www.qemu.org/>

<sup>5</sup>Stress Testing

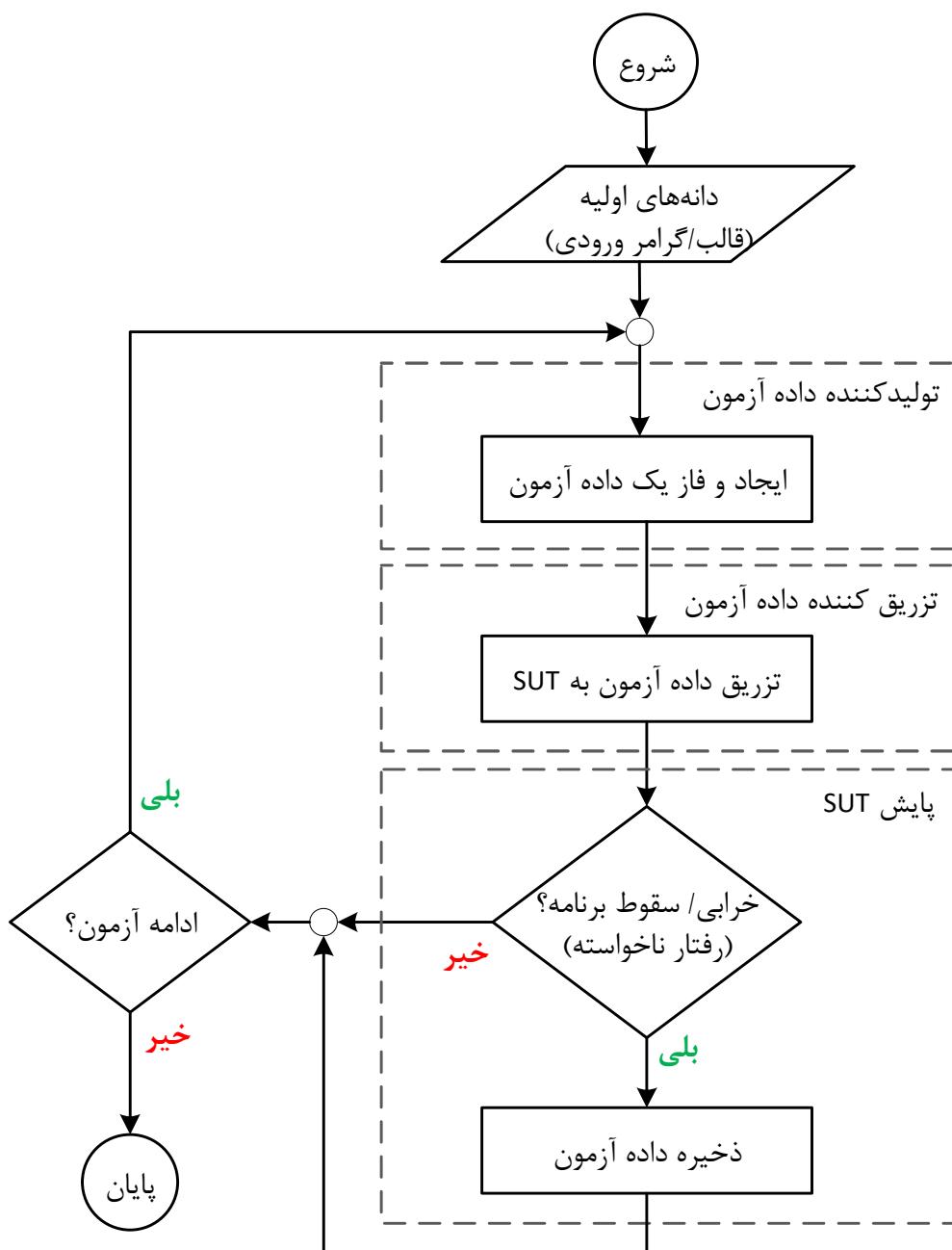
<sup>6</sup>Module

<sup>7</sup>Mutation Based

<sup>8</sup>Generation Based

<sup>9</sup>Valid

<sup>10</sup>Initial Seed



**شکل ۲-۲:** روند نمای فرایند آزمون فازی در حالت ساده که با اقتباس از مراجع مختلف ترسیم شده است. پیمانه‌های مورد نیاز برای خودکارسازی فرایند با مستطیل خطچین مشخص شده‌اند. شرایط ادامه آزمون می‌تواند بر عهده فرد آزمون‌گر قرار داده شود یا توسط خود فازر تعیین گردد.

شبیه شناسایی و تکرار ساختارهای مشخص در داده، جایگزینی اعداد صحیح و حقیقی با مقادیر مرزی (بسیار بزرگ یا کوچک) وغیره. ساختن یک فازر مبتنی بر جابه‌جایی و تولید ورودی بد شکل و مخرب با آن، آسان است و نیاز به شناخت قبلی ساختار داده ورودی مورد جهش ندارد. عیب روش مبتنی بر جابه‌جایی اینجاست که این روش وابسته به تنوع ورودی‌های نمونه است. بدون وجود ورودی‌های مختلف با پیچیدگی کافی، این نوع فازر به پوشش کد بالایی دست نمی‌یابد [۱۲]. به عبارت دیگر دانه اولیه در فازر مبتنی بر جابه‌جایی بسیار حائز اهمیت است. AFL<sup>۱</sup> [۱۹]، FileFuzz<sup>۲</sup> [۲۰] و Radamsa<sup>۳</sup> [۱۲] نمونه‌هایی از فازرهای مبتنی بر جابه‌جایی هستند.

روش مبتنی بر تولید، داده‌های آزمون را به صورت کاملاً تصادفی یا از روی یک توصیف صوری مانند گرامر، قالب، یا مدل تولید می‌کند. در حالت دوم، از مشخصه‌های داده ورودی، برای ساخت یک مدل مولد<sup>۴</sup> استفاده می‌شود. این روش بیشتر روی قالب‌های داده‌ای که مستنداتی از مشخصه‌های آنها در دسترس است، به کار می‌رود که در مقایسه با فازرهای مبتنی بر جابه‌جایی، معمولاً به پوشش کد بالاتری دست می‌یابد. این حال، همان‌طور که در فصل ۱ هم اشاره شد، زمان و هزینه زیادی باید صرف شود تا مشخصه‌های یک قالب داده، کامل فهمیده و مدل خوبی از آن تهیه شود؛ زیرا، این کار تمام خودکار نیست [۳۴] و از طرفی، مستندات ساختار ورودی همواره در دسترس آزمون‌گر، نیستند. SPIKEfile<sup>۵</sup> [۲] و Peach<sup>۶</sup> [۵] نمونه‌ای از فازرهای مبتنی بر تولید هستند. روش‌های ترکیبی نیز وجود دارد که از ویژگی‌های هر دو روش بیان شده در تولید مورد آزمون کمک می‌گیرد. یک مثال از فازرهای ترکیبی LangFuzz<sup>۷</sup> [۳۵] است.

## ۲-۲-۲ معما ری فازرها

یک معما ری مرجع برای فازرها وجود ندارد؛ ولی می‌توان از روی فرایند ترسیم شده برای آزمون فازی (شکل ۲-۲)، پیمانه‌های اصلی لازم برای یک فازر را پیشنهاد کرد. پیمانه اول تولید کننده داده آزمون است که داده‌های ورودی لازم برای آزمون را تولید می‌کند. دیدیم که روش‌های مختلفی برای تولید داده آزمون وجود دارند. پیمانه دوم تزییق کننده داده آزمون است که وظیفه آن تحویل داده‌های تولید شده توسط تولیدکننده به است. هر برنامه واسطه مختص به خود را دارد. واسطه می‌تواند CLI، گرافیکی (GUI)<sup>۷</sup>، پروتکل شبکه یا SUT

<sup>۱</sup>American Fuzzy Lop (<http://lcamtuf.coredump.cx/afl/>)

<sup>۲</sup><http://www.fuzzing.org/>

<sup>۳</sup><https://gitlab.com/akihe/radamsa>

<sup>۴</sup>Generative Model

<sup>۵</sup><http://fuzzing.org/>

<sup>۶</sup><https://www.peach.tech/>

<sup>۷</sup>Graphical User Interface

فایل باشد. هدف این پایان نامه برنامه هایی هستند که ورودی آنها فایل است. معمولاً داده های ورودی به شکل فایل (مثل PDF) ساختار پیچیده تری نسبت به داده های CLI و پروتکل های شبکه دارند و تولید آنها سخت تر است. فازی که برای آزمون این برنامه ها توسعه داده می شود، فازر قالب فایل<sup>۱</sup> هم نامیده می شود [۲]. آخرین پیمانه مورد نیاز ابزار یا محیط پایش است که SUT را پایش می کند تا در صورت بروز خرابی ضمن ذخیره حالت برنامه، محل وقوع خرابی را بتواند با تحلیل حافظه برنامه مشخص کند. البته ابزارهای مستقل زیادی برای این منظور وجود دارند که می توان از آنها بهره گرفت. Application Verifier<sup>۲</sup> [۳۶] یک ابزار پایش قابل استفاده در سیستم عامل ویندوز است. ممکن است فازر، یک بازخورد از ابزار پایش برای تولید داده آزمون بعدی دریافت کند. این بازخورد عموماً حاوی اطلاعات مربوط به پوشش کد اجرای برنامه است. بر این اساس فازرها به دو دسته دارای حلقه بازخورد و بدون حلقه بازخورد تقسیم بندی می شوند [۲۴]. AFL<sup>۳</sup> [۱۹] یک فازر دارای حلقه بازخورد است. در این پایان نامه یک فازر ساده قالب فایل بدون حلقه بازخورد ارائه می دهیم. تمرکز بروی نحوه تولید داده آزمون به روش مبتنی بر تولید / گرامر است.

## ۳-۲ آسیب پذیری

آزمون فازی می تواند برای اهداف مختلفی استفاده شود، اما مهمترین هدف آن تحلیل نرم افزار به منظور کشف آسیب پذیری ها است. یعنی می توان آن را نوعی آزمون نفوذ<sup>۴</sup>، آزمون قدرتمندی<sup>۵</sup> و آزمون امنیت<sup>۶</sup> هم تعریف کرد. تعاریف گوناگونی برای اصطلاح آسیب پذیری نرم افزار وجود دارد. آسیب پذیری را می توان اجتماع سه عنصر دانست: وجود یک خطا در نرم افزار، دسترسی مهاجم به خطا و قابلیت مهاجم برای بهره برداری<sup>۷</sup> از خطا [۳۷، ۲۴]. چنانچه پیش از این اشاره شد، خطا، نرم افزار را در یک حالت اشکال قرار می دهد که منجر به خرابی می شود. بعضی خطاهای ممکن است تنها منجر به خرابی نرم افزار و بروز رفتار ناخواسته (مغایر با مشخصه ها) شوند. بنابراین صرف وجود یک خطا در نرم افزار آسیب پذیری محسوب نمی گردد [۱۴، ۲]. انواع مختلفی از خطاهای وجود دارند که می توانند توسط فازرها کشف شوند؛ به ویژه آنهایی که از جنس نقض دسترسی به حافظه هستند. خطاهای فساد حافظه بدون شک رایج ترین و مؤثر ترین روش بهره برداری خرابکارانه از یک سیستم کامپیوترا م محلی یا راه دور هستند. اگر حافظه بتواند به روشنی خراب شود (یک آدرس بازگشت، یک اشاره گر پشته، یک اشاره گر تابع و غیره) اغلب اجرا می تواند به کد تهیه شده توسط مهاجم

<sup>1</sup> File Format Fuzzer

<sup>2</sup> <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/application-verifier>

<sup>3</sup> Penetration Testing

<sup>4</sup> Robustness Testing

<sup>5</sup> Security Testing

<sup>6</sup> Exploit

هدایت شود. بیشترین آسیب‌پذیری‌هایی که معمولاً توسط محققین حوزه امنیت کشف می‌شوند، مرتبط با میانگیر<sup>۱</sup> (یک ناحیه ثابت از حافظه اصلی در اختیار برنامه) هستند؛ از جمله سرریز میانگیر، سرریز پشته، سرریز عدد صحیح و غیره [۱]. در آزمون فازی به دنبال یافتن چنین خطاهایی از طریق تزریق ورودی بدشکل و خرابی حافظه SUT هستیم. مطالعه و طبقه‌بندی کاملی از انواع آسیب‌پذیری‌ها در [۲۶] انجام شده است.

## ۳-۲ یادگیری ژرف

یادگیری ژرف مجموعه‌ای از فنون یادگیری ماشینی است که در آن بردار ویژگی‌ها به صورت خودکار از داده‌های خام استخراج می‌گردد. ابزار اصلی و غالب در این فنون از یادگیری، شبکه‌های عصبی<sup>۲</sup> هستند. در حالت ساده شبکه‌های عصبی به مسئله یک تابع را نسبت داده و آن را مدل می‌کند. پارامترهای این تابع سپس در فرایند آموزش مدل، تخمین زده می‌شوند. در یادگیری ژرف تعداد این پارامترها صدها، هزارها و گاهی میلیون‌ها برابر افزایش می‌یابد که در نتیجه قدرت بازنمایی<sup>۳</sup> و حل مسئله مدل، افزایش خواهد یافت. شبکه عصبی در این حالت یک گراف با ژرفای بیش از یک یال، خواهد بود که به آن شبکه عصبی ژرف هم می‌گویند. شبکه عصبی ژرف بدین ترتیب قادر به استخراج و تمییز جزئی‌ترین ویژگی‌های مسئله و در نتیجه بهبود دقت حاصله، خواهد بود [۲۸].

وجود تعداد بسیاری پارامتر در مدل‌های یادگیری ژرف، سبب می‌شود که آموزش این مدل‌ها از لحاظ محاسباتی سنگین و زمان‌بر باشد. بسیاری از موانع تئوری و عملی آموزش این مدل‌ها به تازگی رفع شده‌اند. برای مسائل مختلف، مدل‌های متفاوتی با شبکه‌های عصبی ژرف طراحی و ساخته می‌شوند. شبکه عصبی روبه‌جلو<sup>۴</sup> ساده‌ترین نوع است. شبکه عصبی پیچشی (CNN<sup>۵</sup>) مناسب وظایف بینایی ماشین و شبکه عصبی مکرر (RNN) مناسب وظایف پردازش زبان طبیعی (NLP) (وظایف مبتنی بر توالی) ایجاد شده‌اند. وجه مشترک همه شبکه‌های نامبرده، بلوک سازنده آن یعنی عصب<sup>۶</sup> است [۳۹].

<sup>1</sup>Buffer

<sup>2</sup>Neural Network

<sup>3</sup>Representation

<sup>4</sup>Feed-forward Neural Network

<sup>5</sup>Convolutional Neural Network

<sup>6</sup>Neuron

### ۱-۳-۲ عصب

بلوک محاسباتی پایه در بستر شبکه‌های عصبی، عصب یا واحد<sup>۱</sup> است. یک عصب تعدادی عدد حقیقی را به عنوان ورودی گرفته، محاسباتی روی آنها انجام داده و یک خروجی تولید می‌کند. محاسبه بدین صورت است که عصب جمع وزن دار ورودی‌هایی که به آن وارد می‌شوند را به همراه یک مقدار اضافی به عنوان بایاس<sup>۲</sup> حساب می‌کند. سپس برای جلوگیری از محدود شدن فضای توابع قابل تخمین به فضایی خطی، یک تابع غیرخطی، موسوم به تابع انگیزش<sup>۳</sup> (تابع فعالیت)، روی حاصل جمع اعمال می‌شود. تابع انگیزش متداول عبارتند از sigmoid و tanh. شکل ۲-۲ ساختمان داخلی یک عصب تنها با ورودی‌های  $x_1$ ،  $x_2$  و  $x_3$ ، متقابلاً وزن‌های  $w_1$ ،  $w_2$  و  $w_3$  و مقدار بایاس  $b$  را نشان می‌دهد. محاسبات این عصب مطابق رابطه‌های ۱-۲ و ۲-۲ است.

$$z = b + \sum_i w_i x_i \quad (1-2)$$

$$y = \sigma(z) \quad (2-2)$$

رابطه ۱-۲ را می‌توان به صورت ضرب داخلی بردارهای  $W$  و  $x$  هم نشان داد که نمایش مرسوم‌تری است:[۴]

$$z = b + W.x \quad (3-2)$$

### ۲-۳-۲ شبکه عصبی روبه‌جلو

شبکه روبه‌جلو را می‌توان با یک گراف جهت‌دار بدون دور (DAG<sup>۴</sup>) نشان داد. هر گره یک عصب را نشان می‌دهد. گره‌هایی که مسیری به یکدیگر ندارند، یک لایه را تشکیل می‌دهند. ورودی هر لایه، خروجی عصب‌های لایه قبلی است. هر لایه بازنمایی یک مفهوم<sup>۵</sup> در قالب یک تابع است و مسئله ترکیبی از این مفاهیم

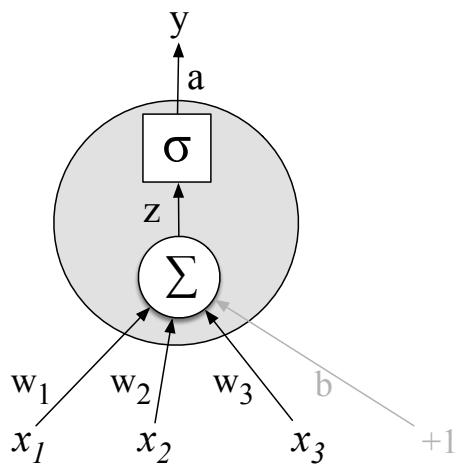
<sup>1</sup>Unit

<sup>2</sup>Bias

<sup>3</sup>Activation Function

<sup>4</sup>Directed Acyclic Graph

<sup>5</sup>Concept



**شکل ۲-۳:** ساختمان داخلی یک عصب با ورودی‌های  $x_1$ ،  $x_2$  و  $x_3$ ، به ترتیب با وزن‌های  $w_1$ ،  $w_2$  و  $w_3$ ،  
قدار بایاس  $b$  و خروجی  $y$  [۳۹].

است. یک شبکه با دو لایه تابع  $f(x) = W_2\sigma(W_1x)$  را پیاده می‌کند که  $W_1$  و  $W_2$  ماتریس پارامترهای تابع و  $\sigma$  تابع انگیزش است.  $f(x) = W_3\sigma(W_2\sigma(W_1x))$  معرف یک شبکه سه‌لایه خواهد بود. شکل ۴-۲ دو گراف محاسباتی با زرفاي ۲ و ۳ لایه (به ترتیب از چپ به راست) را نشان می‌دهد. لایه‌های میانی را لایه‌های پنهان<sup>۱</sup> هم می‌گویند. توجه شود که ورودی اول شبکه یک لایه محاسباتی نیست و بنابراین در شمارش لایه‌ها، شمرده نمی‌شود.

### آموزش شبکه روبه‌جلو

هدف از آموزش شبکه، تعیین مقادیر مناسب برای ضرایب و بایاس‌ها و سپس استفاده از شبکه برای انجام وظیفه<sup>۲</sup> مورد نظر است. داده‌هایی که در فرایند آموزش شبکه استفاده می‌گردد مجموعه آموزش<sup>۳</sup> نامیده می‌شوند. داده‌هایی که شبکه با آن مورد سنجش قرار می‌گیرد، مجموعه آزمون<sup>۴</sup> نامیده می‌شوند و مجموعه سوم از داده‌ها که در طول فرایند آموزش شبکه برای انتخاب بهترین راهکارهای یادگیری به کار گرفته می‌شوند، به نام مجموعه ارزیابی<sup>۵</sup> شناخته می‌شوند [۳۸].

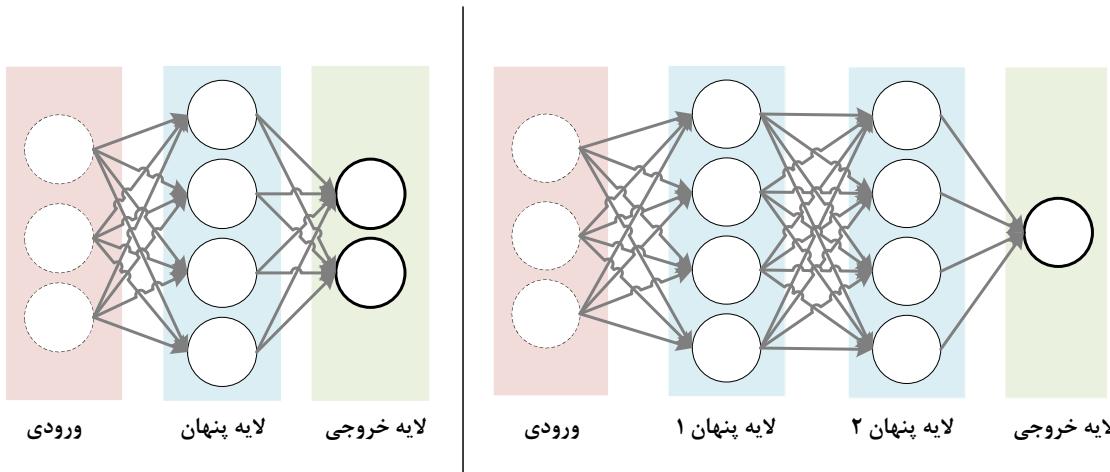
<sup>1</sup>Hidden Layer

<sup>2</sup>Task

<sup>3</sup>Training Set

<sup>4</sup>Test Set

<sup>5</sup>Validation Set



**شکل ۲-۴:** گراف محاسباتی برای دو شبکه عصبی روبه‌جلوی فرضی. چپ: یک شبکه عصبی دو لایه (یک لایه پنهان متشکل از چهار عصب و یک لایه خروجی متشکل از دو عصب) به همراه سه ورودی. راست: یک شبکه عصبی سه‌لایه (دو لایه پنهان هر کدام متشکل از چهار عصب و یک لایه خروجی متشکل از یک عصب) و به همراه سه ورودی.

شبکه عصبی در یک راهبرد بانظارت<sup>۱</sup> آموزش می‌بیند. در یادگیری بانظارت<sup>۲</sup> هر نمونه در مجموعه داده، علاوه بر بردار ویژگی‌ها با یک برچسب<sup>۳</sup> همراه است که نقش ناظر را ایفا می‌کند. طبقه‌بندی<sup>۴</sup> نمونه‌ای از وظایف یادگیری بانظارت است. در این وظیفه برای هر ورودی، کلاس خروجی آن در زمان آموزش مشخص است.

در فرایند آموزش ابتدا پارامترهای شبکه مقداردهی اولیه می‌شوند (معمولًاً به صورت تصادفی). سپس به ازای هر ورودی از مجموعه آموزش، عصب‌های هر لایه خروجی خود را تولید می‌کنند تا زمانی که خروجی نهایی شبکه تولید شود. این مرحله را گذر جلو<sup>۵</sup> می‌گویند. از آنجایی که خروجی درست برای ورودی مربوطه در زمان آموزش درسترس است (یادگیری بانظارت)، اختلاف مقدار محاسبه شده توسط شبکه ( $\hat{y}$ ) و مقدار واقعی ( $y$ )، توسط تابع موسوم به تابع خطای<sup>۶</sup> یا تابع هزینه<sup>۷</sup> یا تابع هدف محاسبه می‌شود [۴۰]. توابع مختلفی برای این منظور می‌توان تعریف کرد. تابع خطای مطلق میانگین<sup>۸</sup> (رابطه ۲-۴)، تابع خطای میانگین

<sup>1</sup>Supervised

<sup>2</sup>Supervised Learning

<sup>3</sup>Label

<sup>4</sup>Classification

<sup>5</sup>Forward Pass

<sup>6</sup>Error Function

<sup>7</sup>Cost Function

<sup>8</sup>Mean Absolute Error

مربعات<sup>۱</sup> (رابطه ۲-۵) و تابع خطای آنتروپی متقاطع<sup>۲</sup> (رابطه ۲-۶) توابعی هستند که بسته به وظیفه مورد نظر استفاده می‌شوند [۴۱].

$$L_{MAE}(\hat{y}, y; W, b) = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i| \quad (4-2)$$

$$L_{MSE}(\hat{y}, y; W, b) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (5-2)$$

$$L_{CE}(\hat{y}, y; W, b) = - \sum_{i=1}^n y_i \log \hat{y}_i \quad (6-2)$$

در روابط ۴-۲ تا ۶-۲،  $n$  تعداد نمونه‌ها یا تعداد مشاهده‌ها است. در این پایان‌نامه از تابع خطای آنتروپی متقاطع استفاده می‌کنیم که برای وظیفه‌های طبقه‌بندی با بیش از دو کلاس مناسب است [۴۰]. وقتی میزان خطا در گذر جلو مشخص شد، هدف تغییر میزان پارامترهای شبکه به نحوی است که خطا کمینه شود (رابطه ۷-۲):

$$f^* = \arg \min_{W, b} L(\hat{y}, y; W, b) \quad (7-2)$$

که  $f^*$  تابع یادگیری شده و مقادیر  $W$  و  $b$  پارامترهای شبکه هستند. سایر مقادیر را که در طول این فرایند یادگیری نمی‌شوند، ابر پارامتر<sup>۳</sup> می‌نامند. از جمله ابر پارامترها می‌توان به تعداد لایه‌ها و تعداد عصب‌ها در هر لایه اشاره کرد، که از آنها برای تعیین اندازه شبکه استفاده می‌گردد.

برای کمینه‌سازی خطا، مشتقات جزئی تابع خطای نسبت به هریک از پارامترها در لایه خروجی محاسبه می‌شود (گرادیان) و سپس این روند از طریق روال پس‌انتشار<sup>۴</sup> تا ابتدای شبکه پیش می‌رود و مقدار هر پارامتر درسی کاهش گرادیان با ضریبی موسوم به نرخ یادگیری<sup>۵</sup> بروزرسانی می‌شود. نرخ یادگیری عدد کوچکی (در مقیاس صدم و هزارم) در نظر گرفته می‌شود که برای همگرایی آهسته و پیوسته شبکه و جلوگیری از

<sup>1</sup>Mean Squared Error

<sup>2</sup>Cross Entropy Error

<sup>3</sup>Hyper-parameter

<sup>4</sup>Backpropagation

<sup>5</sup>Learning Rate

و اگرا شدن آن لازم است. نرخ یادگیری یکی دیگر از ابر پارامترهای مورد نیاز آموزش شبکه‌های ژرف است. الگوریتم پسانشان در واقع یک کاربرد بازگشتی از قاعده مشتق زنجیری در حساب دیفرانسیل است [۴۰]. در اینجا به بیان جزئیات الگوریتم‌های بهینه‌سازی نمی‌پردازیم؛ چارچوب‌های یادگیری ژرف انواع مختلفی از این الگوریتم‌ها را پیاده‌سازی و در اختیار قرار داده‌اند. توضیحات کاملی از این الگوریتم‌ها در [۳۸] آمده است.

### شرایط توقف آموزش

فرایند آموزش شبکه می‌تواند تا بی‌نهایت ادامه داشته باشد. بنابراین بایستی یک قانون برای توقف آن تعریف شود. گزینه‌های زیادی وجود دارد که تحت چه شرایطی آموزش را خاتمه دهیم. همچنین ترکیب این شرایط نیز امکان‌پذیر است. برخی از این شرایط عبارتند از [۴۱]:

۱. هنگامی که تعداد مشخصی دوره<sup>۱</sup> طی شود (هر دوره برابر تعداد تکرارهایی است که برای مرور کل مجموعه آموزش لازم است).
۲. هنگامی که خطای مجموعه ارزیابی (برای تعداد مشخصی دوره متوالی) کاهش پیدا نکند.
۳. هنگامی که تغییرات میزان خطای (برای تعداد مشخصی دوره متوالی) کمتر از یک حد آستانه شود.
۴. هنگامی که یک زمان مشخص از فرایند آموزش سپری شود.

### منظم‌سازی

مشکل رایجی که هنگام آموزش شبکه عصبی باید جلوگیری شود اثر بیش‌برازش<sup>۲</sup> است. بیش‌برازش یعنی با کاهش خطای روی مجموعه آموزش، خطای مجموعه‌های ارزیابی و آزمون به‌طور ناگهانی افزایش یابد [۳۸]. یک دلیل می‌تواند به اندازه کافی بزرگ نبودن اندازه مجموعه آموزش باشد. دلیل دیگر می‌تواند پیچیدگی بسیار بالای مدل باشد. پیچیدگی مدل با تعداد پارامترهای آن مشخص می‌شود. در پیچیدگی بالا ممکن است مدل، اختلال<sup>۳</sup>‌های موجود در ورودی‌ها را نیز لاحظ کند و متناسب یا برازنده آن شود. در نتیجه هنگام ارزیابی مدل روی داده‌های مجموعه آزمون، خطای بالا می‌رود [۴۱].

راهکارهای مختلفی برای مقابله با مسئله بیش‌برازش مدل پیشنهاد شده است. از این راهکارها تحت عنوان فنون منظم‌سازی<sup>۴</sup> یا تنظیم مدل یاد می‌شود. اغلب روش‌های شناخته شده، پارامترهای دارای مقادیر بالا را

<sup>1</sup>Epoch

<sup>2</sup>Overfitting

<sup>3</sup>Noise

<sup>4</sup>Regularization

## ۳-۲. یادگیری ژرف

که اثرات نوسانی شدیدی دارند، با تابعی جریمه می‌کنند [۴۰]. روش دیگر برای منظم‌سازی Dropout [۴۲] است که بسیار مؤثر و ساده بوده و در مدل‌های یادگیری ژرف معمولاً از آن استفاده می‌شود. در طول آموزش Dropout یک عصب را فقط با احتمال  $p$  (یک ابر پارامتر) فعال نگه می‌دارد و در غیر این صورت آن را صفر می‌کند. در نتیجه پیچیدگی مدل تنظیم می‌شود.

## ۴-۳-۲ شبکه عصبی مکرر

شبکه‌های روبه‌جلو در حل مسائلی که ترتیب ورودی در آنها مهم است، مثل ترجمه ماشینی و سایر وظیفه‌های NLP، نارسایی دارند. برای مثال در ترجمه ماشینی هر واژه به واژه‌های قبلی خود وابسته است. در این وظیفه‌ها ورودی به صورت یک توالی  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$  است. غالب وظایف حوزه پردازش زبان طبیعی بدین صورت هستند.

شبکه‌های عصبی مکرر کلاسی از شبکه‌های عصبی هستند که به صورت یک گراف جهت‌دار دارای دور بیان می‌شوند. به عبارت دیگر ورودی هریک از لایه‌های (پنهان یا لایه خروجی) افزون بر خروجی لایه قبل، شامل ورودی‌ای از گام زمانی<sup>۱</sup> قبل به صورت بازخورد نیز می‌شود. شکل ۵-۲ یک RNN را نشان می‌دهد که در آن لایه پنهان از گام‌های زمانی قبلی بازخورد گرفته است. در هر گام زمانی  $t$  (از  $t=1$  تا  $t=T$ ) یک بردار  $x^{(t)}$  از توالی ورودی پردازش می‌شود. معادلات گذر جلو شبکه در  $t$  عبارتند از [۳۸]:

$$z^{(t)} = Ux^{(t)} + Wh^{(t-1)} + b \quad (8-2)$$

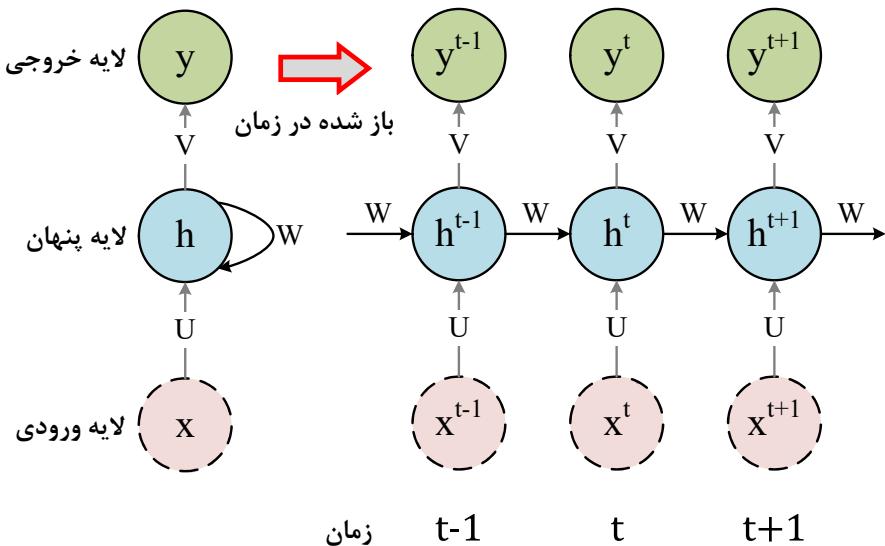
$$h^{(t)} = \sigma(z^{(t)}) \quad (9-2)$$

$$y^{(t)} = Vh^{(t)} + c \quad (10-2)$$

$$\hat{y}^{(t)} = softmax(y^{(t)}) \quad (11-2)$$

در روابط ۸-۲ تا ۱۱-۲،  $b$  و  $c$  بایاس و ماتریس‌های  $U$ ،  $V$  و  $W$  به ترتیب وزن یال‌های لایه ورودی

<sup>۱</sup>Time Step



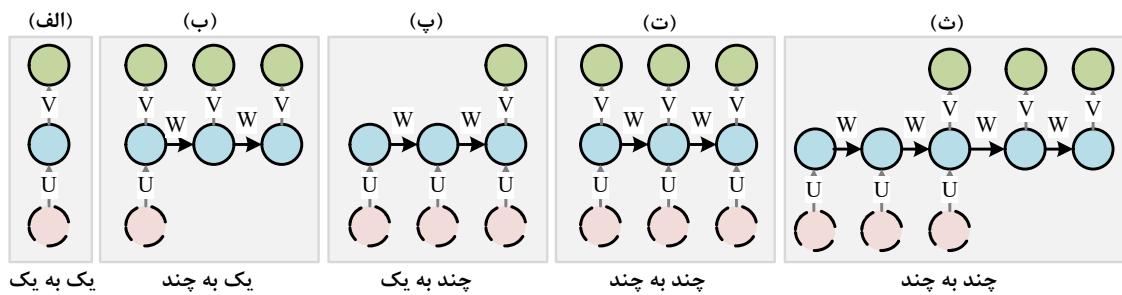
**شکل ۲-۵:** گراف محاسباتی مربوط به یک شبکه عصبی مکرر با یک لایه پنهان که یک توالی ورودی از مقادیر  $y = \langle y^{(1)}, y^{(2)}, \dots, y^{(n)} \rangle$  را به یک توالی خروجی از مقادیر  $x = \langle x^{(1)}, x^{(2)}, \dots, x^{(n)} \rangle$  نگاشت می‌کند. فرض شده است که خروجی  $y$  احتمال‌های نرمال نشده است، بنابراین خروجی نهایی شبکه یعنی  $\hat{y}$  از اعمال تابع بیشینه هموار روی  $y$  حاصل می‌شود. چپ: شبکه عصبی مکرر به صورت یال بازگشته (گراف جهت دار دارای دور). راست: همان شبکه به صورت باز شده در زمان، به نحوی که هر گره با برچسب مرحله زمانی مشخص شده است [۳۸].

به پنهان، پنهان به خروجی و پنهان به پنهان، پارامترهای شبکه هستند. در لایه خروجی به جای تابع انگیزش، تابع بیشینه هموار<sup>۱</sup> اعمال می‌شود. این تابع خروجی شبکه را به شکل یک توزیع احتمالی معتبر تبدیل می‌نماید و معمولاً در لایه خروجی مدل‌هایی که برای طبقه‌بندی استفاده می‌شوند، قرار می‌گیرد. تابع بیشینه هموار یک بردار  $k$ -تایی از اعداد حقیقی را به عنوان ورودی دریافت نموده و یک بردار  $k$ -تایی از مقادیر حقیقی در بازه  $[0, 1]$  را به عنوان خروجی می‌دهد به طوری که جمع مؤلفه‌های آن برابر یک خواهد بود (رابطه ۱۲-۲):

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}} \quad \text{for } i=1 \text{ to } k \quad (12-2)$$

در شکل ۲-۵، RNN با یک لایه پنهان نشان داده شده است. اما می‌توان RNN ژرف با چندین لایه پنهان نیز داشت. همچنین طول توالی‌های ورودی و خروجی می‌تواند بسته به مسئله مورد نظر متفاوت باشد.

<sup>۱</sup>Softmax



**شکل ۲-۶:** طرح‌واره‌ای از انواع حالت‌های مختلف شبکه عصبی مکرر براساس طول توالی ورودی و طول توالی خروجی. (الف): شبکه عصبی استاندارد، (ب): شبکه یک به چند، (پ): شبکه چند به یک، (ت) و (ث): شبکه‌های چند به چند [۴۰].

کارپتی<sup>۱</sup> [۴۰] RNN‌ها را از منظر طول توالی ورودی و طول توالی خروجی به چند دسته تقسیم‌بندی کرده است. شکل ۲-۶ این دسته‌بندی را نشان می‌دهد.

### آموزش شبکه عصبی مکرر

الگوریتم پساننتشار برای آموزش RNN هم قابل استفاده است. در واقع RNN همان شبکه عصبی روبه‌جلو است که یک بازخورد از وزن‌های گام زمانی قبل دارد. در نتیجه خط‌های خروجی به صورت روی‌به‌عقب از گام زمانی  $t$  منتشر می‌شود. بسته به طول توالی ورودی و میزان منابع محاسباتی این انتشار تا گام زمانی  $t = 1$  ادامه می‌یابد یا تحت محدودیت تعداد مشخصی گام<sup>۲</sup> متوقف می‌شود. نسخه‌ای از الگوریتم پساننتشار که می‌تواند به کل طول توالی ورودی اعمال شود، پساننتشار در زمان (BPTT<sup>۳</sup>) نام دارد. محدودیت منابع محاسباتی ایجاب می‌نماید که طول توالی ورودی مقدار مشخصی قرار داده شود یا پساننتشار در تعداد محدودی گام‌زمانی متوقف شود [۴۸].

### حافظه کوتاه‌مدت بلند

RNN در حالت استاندارد، برای توالی‌های طولانی که وابستگی بین ورودی‌هایی با فاصله زیاد وجود دارد، قادر به حفظ این وابستگی‌ها نیست؛ به بیان دیگر گرادیان در دوره‌های زمانی بلند مدت تمایل به ناپدید شدن<sup>۴</sup> یا انفجار<sup>۵</sup> (زیاد شدن بیاندازه) دارد. در حالت ناپدید شدن گرادیان، یادگیری مدل متوقف می‌شود. مدل

<sup>۱</sup>A. Karpathy (<http://karpathy.github.io/>)

<sup>۲</sup>Step

<sup>۳</sup>Back Propagation Through Time

<sup>۴</sup>Vanishing

<sup>۵</sup>Expllosion

حافظه کوتاه‌مدت بلند (LSTM<sup>۱</sup>) برای مقابله با مسئله بالا و افزایش قدرت به‌خاطر سپاری RNN ایجاد شده است. هسته این مدل سلول حافظه C است که در آن بر خلاف شبکه مکرر استاندارد که خروجی هر واحد تنها با اعمال یکتابع انگیزش ایجاد می‌شود، خروجی با محاسبات پیچیده‌تری تعیین می‌شود که به‌ویژه هدف آنها منظم‌سازی و حفظ قدرت گرادیان است. ایده اصلی در LSTM تخصیص مسیری مجزا علاوه بر یال بازخوردهی حالت پنهان، برای جریان حافظه است.

جزئیات عملکرد LSTM در [۳۸] توضیح داده شده است. امروزه به‌طور پیشفرض از LSTM در پیاده‌سازی RNN استفاده می‌شود [۴۳]. البته راهکارهای جایگزین دیگری مشابه LSTM، نیز مطرح شده‌اند مانند<sup>۲</sup> GRU<sup>۳</sup> [۱۸]، اما عملکرد آنها بهتر از LSTM نیست. مطالعه جامعی روی RNN و انواع آن برای یادگیری توالی‌ها در [۴۴] انجام شده است. کلیه مدل‌های یادگیری ژرف معرفی شده در این پایان‌نامه از LSTM در هسته خود استفاده می‌کنند.

## ۴-۲ مدل زبانی

در فصل ۱ به استفاده از LM در روش پیشنهادی خود، اشاره کردیم. LM یک مفهوم پایه در NLP است که امکان پیش‌بینی شانه بعدی در یک توالی را فراهم می‌کند. بهبیان دقیق‌تر LM عبارت است از یک توزیع احتمالی روی یک توالی از نشانه‌ها (اغلب واژه‌ها) که احتمال وقوع یک توالی داده شده را مشخص می‌کند. در نتیجه می‌توان بین چندین توالی داده شده برای مثال چند جمله، آن را که محتمل‌تر است، انتخاب کرد. برای توالی  $x^{(n)} < x^{(1)}, x^{(2)}, \dots, x^{(n)}$  به صورت زیر تعریف می‌شود [۴۵]:

$$p(x) = \prod_{t=1}^n p(x^{(t)} | x^{<t}) \quad (13-2)$$

در رابطه ۱۳-۲ هر جمله منفرد  $p(x^{(t)} | x^{<t})$  احتمال شرطی نشانه  $x^{(t)}$  به شرط وقوع (ظاهر شدن)  $t$  نشانه قبلی  $x^{<t}$  در توالی است که به آن زمینه<sup>۳</sup> یا تاریخچه<sup>۴</sup> نیز می‌گویند. در عمل محاسبه این احتمال به صورت رابطه ۱۳-۲ تقریباً غیر ممکن است؛ زیرا، نیازمند داشتن همه توالی‌های ممکن هستیم. مدل‌های سنتی n-gram برای غلبه بر چالش‌های محاسباتی، با استفاده از فرض مارکوف رابطه ۱۳-۲ را به درنظر گرفتن تنها  $1-n$  نشانه قبلی محدود می‌کنند. اگرچه در بسیاری از مسائل این مدل‌ها به خوبی پاسخ‌گو هستند؛

<sup>1</sup>Long-Short Term Memory

<sup>2</sup>Gated Recurrent Units

<sup>3</sup>Context

<sup>4</sup>History

اما، برای توالی‌های طولانی (بیشتر از ۴ یا ۵ نشانه) و نیز مشاهده نشده مناسب نیستند. در حالتی که نشانه فعلی پیش از این مشاهده نشده باشد، احتمال صفر به آن نسبت داده می‌شود که سبب صفر شدن احتمال پایانی می‌گردد. برای حل این مشکل، مدل‌های n-gram نیازمند اعمال فنون هموارسازی<sup>۱</sup> هستند [۳۹]. این فنون، راه حل‌هایی برای از بین بردن احتمال‌های صفر پیشنهاد می‌دهند. در هر صورت، مدل‌های n-gram با محدودیت‌های جدی روبرو هستند.

## ۱-۴-۲ مدل زبانی عصبی

می‌توان از RNN برای ایجاد مدل زبانی استفاده کرد. این مدل‌ها را مدل زبانی عصبی (NLM) می‌گویند. استفاده از RNN هر دو مشکل اشاره شده در بالا را برطرف می‌کند. یعنی هم امکان پیش‌بینی توالی‌های طولانی‌تر فراهم می‌شود و هم وقوع احتمالات صفر از بین می‌رود [۴۵]. شکل ۲-۷ یک معماری از RNN برای ساخت مدل زبانی در NLP را نشان می‌دهد. چون هدف مدل زبانی پیش‌بینی نشانه بعدی است، توالی  $x$  را در ورودی شبکه قرار داده و خروجی متناظر با آن را همان توالی  $x$  که یک واحد روبه‌جلو شیفت داده شده است، می‌گذارند. این روش آموزش شبکه Teacher Forcing می‌گویند [۳۸].

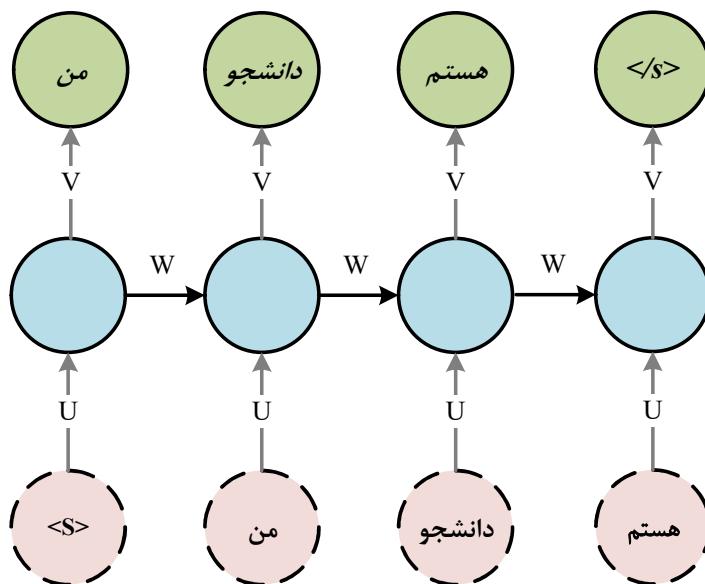
مدل‌های زبانی را مدل مولد نیز می‌نامند؛ زیرا یک توزیع احتمالی روی توالی‌های یک زبان می‌دهند که با نمونه‌برداری از آن می‌توان توالی‌های جدید تولید کرد. در NLP نشانه‌های هر توالی واژه‌ها هستند، اما می‌توان این مدل را در سطح کاراکتر نیز آموزش داد. ایده اصلی در این پایان‌نامه نیز همین است. هر فایل زبان و قواعد مخصوص به خود را دارد که در مشخصه‌های قالب فایل ذکر شده است. با ایجاد یک مدل زبانی برای این زبان در سطح کاراکتر، امکان پیش‌بینی نشانه بعدی از روی یک توالی از نشانه‌های داده شده فراهم می‌شود. با نمونه‌برداری از این مدل زبانی می‌توان داده‌های آزمون جدیدی ایجاد کرد.

## ۲-۴-۲ ارزیابی مدل زبانی

چگونه می‌توان میزان خوب بودن یک مدل زبانی را تعیین کرد؟ یک روش برای ارزیابی مدل زبانی، قرار دادن آن در یک وظیفه مشخص و اندازه‌گیری میزان دقت نتایج حاصله است. به این روش ارزیابی بیرونی<sup>۲</sup> می‌گویند [۳۹]. برای مثال در یک وظیفه تصحیح خودکار غلط‌های املایی، تعداد واژه‌های نادرستی که به درستی با واژه‌های صحیح جایگزین شده‌اند را شمارش می‌کنیم و بر تعداد کل واژه‌های غلط تقسیم می‌نماییم. ارزیابی

<sup>1</sup>Smoothing

<sup>2</sup>Extrinsic Evaluation



**شکل ۲-۷:** طرح‌واره‌ای از معماری یک مدل زبانی ایجاد شده با استفاده از RNN.  $< s >$  نشانه شروع توالی و  $< /s >$  نشانه خاتمه توالی در ابتدا و انتهای هر توالی موجود در مجموعه آموزش قرار داده می‌شود. بدین ترتیب، تولید توالی جدید پس از پیش‌بینی نشانه پایان، متوقف می‌گردد [۴۵].

بیرونی، زمانبر، پرهزینه و مختص به وظیفه اجرا شده خواهد بود. روش دیگر استفاده از ارزیابی درونی<sup>۱</sup> است [۳۹]. معیار سرگشتشگی<sup>۲</sup> با هدف ارزیابی مدل زبانی، به صورت زیر تعریف می‌شود [۴۶]:

$$\begin{aligned} PP_{LM}(x) &= \sqrt[n]{\prod_{i=1}^n \left( \frac{1}{p(x^{(i)} | x^{(1)}, \dots, x^{(i-1)})} \right)} \\ &= 2^{-\frac{1}{n} \sum_{i=1}^n \log_2 p(x^{(i)} | x^{(1)}, \dots, x^{(i-1)})} \end{aligned} \quad (14-2)$$

در رابطه ۱۴-۲،  $x$  توالی مورد ارزیابی است. همان‌طور که مشاهده می‌شود سرگشتشگی در ارتباط مستقیم با خطای آنتروپی متقاطع (رابطه ۲-۶) است که میزان اختلاف بین مدل و نمونه‌های مجموعه آزمون را نشان می‌دهد. بنابراین هرچه قدر میزان سرگشتشگی پایین‌تر باشد، مدل زبانی بهتر است. پایه توان و پایه لگاریتم در رابطه ۱۴-۲ می‌توانند مقادیری به غیر از ۲ انتخاب شود. اما در هر حالت هر دو پایه باستی یکسان باشند تا تساوی اول (بالا) و دوم (پایین) رابطه برقرار گردد. معمولاً از پایه لگاریتم طبیعی در این رابطه استفاده

<sup>۱</sup>Intrinsic Evaluation

<sup>۲</sup>Perplexity

می شود [۴۷].

برای درک بهتر نحوه ارزیابی توسط این معیار، توالی  $x = x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(n)}$  را در نظر می‌گیریم که مشکل از  $V$  نشانه متفاوت است.  $V$  مجموعه واژگان<sup>۱</sup> زبان است. در غیاب مدل زبانی (بدترین حالت)، تنها می‌توانیم ادعا کنیم که هر نشانه دستکم با احتمال  $\frac{1}{V}$  در توالی رخ می‌دهد (ضریب انشعاب<sup>۲</sup>). بدیهی است که این احتمال برای همه نشانه‌ها یکسان است. برای توالی  $x$  پس از جایگزینی احتمال آن در رابطه ۱۴-۲ داریم:

$$PP_{LM}(x) = \sqrt[n]{\prod_{i=1}^n \left(\frac{1}{V}\right)} = \sqrt[n]{V^n} = V$$

بنابراین در بدترین حالت میزان سرگشتگی برابر با اندازه مجموعه واژگان زبان است. حال چنان‌چه از مدل زبانی استفاده کنیم، احتمال نسبت داده شده به وقوع هر نشانه، نسبت به حالت عدم استفاده از مدل زبانی، افزایش و در نتیجه سرگشتگی کاهش خواهد یافت. در ارزیابی مدل‌های این پایان‌نامه از معیار سرگشتگی استفاده می‌کنیم.

## ۵-۲ خلاصه

در این فصل مفاهیم اولیه سه حوزه کلی مطرح در این پایان‌نامه یعنی آزمون نرم‌افزار، آزمون فازی و یادگیری ژرف را بیان کردیم. آزمون نرم‌افزار به عنوان یک مسئله تصمیم‌ناپذیر تعریف گردید و معیارهای پوشش برای ارزیابی آن معرفی شدند. آزمون فازی به عنوان یک فن مطرح در آزمون نرم‌افزار، معرفی و از سه دیدگاه مختلف مورد طبقه‌بندی و بحث قرار گرفت: نخست، از دیدگاه اطلاعات در دسترس از SUT به سه دسته آزمون جعبه سیاه، جعبه سفید و جعبه خاکستری تقسیم گردید. دوم، از دیدگاه روش‌های تولید داده آزمون به سه دسته مبتنی بر جایه‌جایی، مبتنی بر تولید و نیز ترکیبی تقسیم‌بندی شد. سوم، با توجه به دریافت بازخورد از نتیجه اجرا به دو دسته دارای بازخورد و بدون بازخورد تفکیک گشت. ابزار خودکارسازی آزمون فازی تحت عنوان کلی فازر معرفی و معماری آن تشریح شد. فازرها با توجه به طبقه‌بندی‌های فوق در بحث آزمون فازی و نیز با توجه به نوع ورودی SUT متفاوت هستند. فازرها قابل فایل مربوط به برنامه‌هایی با ورودی فایل هستند. برای اطلاعات بیشتر در زمینه آزمون نرم‌افزار خواننده را به [۲۷، ۱۳]<sup>۲</sup> ارجاع می‌دهیم. در زمینه آزمون فازی نیز خواننده را به [۲۹، ۲۴]<sup>۳</sup> رجوع می‌دهیم.

در بخش پایانی این فصل، یادگیری ژرف به عنوان زیرشاخه‌ای از یادگیری ماشینی با استفاده از شبکه‌های

<sup>1</sup>Vocabulary

<sup>2</sup>Branching Factor

عصبی مصنوعی، تفهیم شد. شبکه عصبی مورد استفاده در یادگیری ژرف را شبکه عصبی ژرف گویند. RNN نوع خاصی از شبکه‌های عصبی ژرف است که برای پردازش وظیفه‌های مبتنی بر توالی مثل مدل زبانی استفاده می‌شود. ساختار یک قالب فایل زبان مختص به خود را دارد و هدف از مطرح کردن اصول اولیه یادگیری ژرف استفاده از مدل‌های این فن، در یادگیری ساختار فایل، جهت تولید داده آزمون جدید است که در فصل ۴ به آن خواهیم پرداخت. الگوریتم‌های آموزش شبکه‌های عصبی ژرف ریاضیات گسترده‌ای دارد که به‌طور خلاصه به موارد مهم آن اشاره شد. جهت مطالعه مبسوط این الگوریتم‌ها خواننده را به [۳۸] ارجاع می‌دهیم. همچنین توضیح کاملی از روش‌های یادگیری توالی با استفاده از RNN‌ها در [۴۴] آمده است. در فصل بعد برخی از تازه‌ترین کارهای مرتبط با مفاهیم مطرح شده در این فصل توضیح داده می‌شوند.

## فصل ۳

### کارهای مرتبط

«یک شب تاریک و طوفانی بود. پاییز ۱۹۹۴، در آپارتمان خود در مادیسون نشسته بودم. آن شب من از طریق خط تلفن به سیستم‌های یونیکس دانشگاه متصل شدم. با بارش سنگین باران اختلال زیادی روی خط اتصالی بود که در اجرای فرمان‌های ارسالی من دخالت می‌کرد. رقابتی بین سرعت نوشتن یک فرمان قبل از آنکه اختلال آن را خراب کند وجود داشت. چیزی که مرا شگفت زده می‌کرد این حقیقت بود که اختلال سبب ایجاد خرابی و سقوط برنامه‌ها می‌شد و شگفت‌آورتر برنامه‌هایی بود که سقوط می‌کرد: ابزارهای رایج یونیکس!»

---

#### ✿ بارتون میلر، مبدع آزمون فازی

کارهای بسیاری برای بهبود آزمون فازی اولیه که داده‌های آزمون را به صورت تصادفی تولید می‌کرد [۳]، انجام شده است. فازرهای مبتنی بر تولید در برنامه‌های با ساختار ورودی پیچیده پوشش کد بیشتری نسبت به فازرهای مبتنی بر جابه‌جایی فراهم می‌کنند [۳۴]، اما کاملاً خودکار نیستند [۱۱]. در مقابل فازرهای مبتنی بر جابه‌جایی سعی کرده‌اند تا با استفاده از الگوریتم‌های تکاملی مثل ژنتیک داده آزمون‌های بهتری را برای جابه‌جایی انتخاب کنند. AFL [۱۹] نمونه‌ای موفق از فازرهای مبتنی بر جابه‌جایی است. در سال ۲۰۱۷، فنون یادگیری ماشینی به هر دوسته از فازرهای بالا اعمال شده‌اند. در فازرهای مبتنی بر تولید، برای یادگیری خودکار گرامر فایل [۱۱] و در فازرهای مبتنی بر جابه‌جایی برای پیش‌بینی بهترین مکان جابه‌جایی [۲۰].

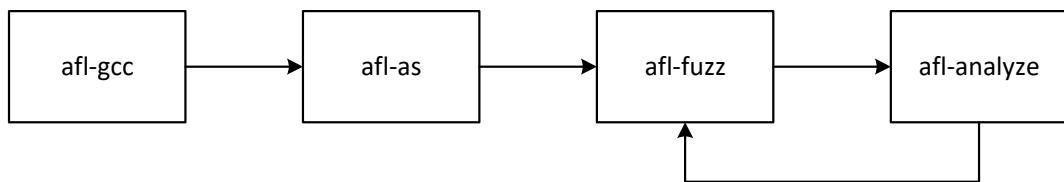
هریک از این کارها نواقص و محدودیت‌هایی دارند. در این فصل به معرفی، نقد و بررسی این کارها می‌پردازیم.

## ۱-۳ فازر AFL

[۱۹] AFL یک فازر قالب فایل جعبه خاکستری، با تولید داده آزمون مبتنی بر جابه‌جایی، دارای بازخورد، متن‌باز و رایگان است که توسط Michal Zalewski<sup>۱</sup> توسعه داده شده است. این فازر روی سیستم عامل‌های خانواده یونیکس قابل اجرا است. راهاندازی آن ساده بوده و واسط کاربری خوبی برای دنبال کردن جزئیات آماری فرایند آزمون فازی و خطاهای کشف شده دارد. در شکل ۱-۳ یک نمونه از اجرای این فازر را در عمل نشان داده‌ایم.

به‌طور پیش‌فرض AFL برای سنجش پوشش کد، به کد منبع SUT نیاز دارد. برنامه‌های نوشته شده به زبان‌های C، C++، Objective-C و Python در آزمون فازی جعبه سفید با AFL قابل استفاده هستند. همچنین نسخه‌هایی از AFL برای برنامه‌های نوشته شده به زبان‌های Go و Python توسط دیگران انتشار یافته است. AFL در آزمون فازی جعبه سیاه، برای ابزارگذاری و اخذ اطلاعات زمان اجرا، از ابزار QEMU [۳۳] استفاده می‌کند. این فازر به حدی موفق بوده که پژوهش‌های زیادی برای بهبود جنبه‌های مختلف آن انجام شده است. اما چنان‌چه خواهیم دید روی قالب فایل‌هایی با ساختار پیچیده نمی‌تواند به پوشش خوبی دست پیدا کند [۲۰].

<sup>۱</sup><http://lcamtuf.coredump.cx/>



شکل ۳-۲: مؤلفه‌های فازر AFL و ارتباط آنها با یکدیگر [۱۹].

```

american fuzzy lop 2.52b (mutool)

process timing
  run time : 55 days, 21 hrs, 46 min, 4 sec
  last new path : 0 days, 7 hrs, 13 min, 50 sec
  last uniq crash : 54 days, 5 hrs, 18 min, 1 sec
  last uniq hang : 55 days, 10 hrs, 9 min, 16 sec
cycle progress
  now processing : 5058 (46.09%)
  paths timed out : 22 (0.20%)
stage progress
  now trying : arith 8/8
  stage execs : 5.19M/7.75M (67.01%)
  total execs : 577M
  exec speed : 45.69/sec (slow!)
fuzzing strategy yields
  bit flips : 1431/38.3M, 275/38.3M, 123/38.3M
  byte flips : 8/4.78M, 6/2.58M, 21/2.58M
  arithmetics : 566/136M, 10/22.6M, 2/3.67M
  known ints : 18/14.5M, 116/67.0M, 126/87.5M
  dictionary : 0/0, 0/0, 170/101M
  havoc : 2424/1.58M, 0/0
  trim : 0.01%/171k, 46.03%
overall results
  cycles done : 0
  total paths : 11.0k
  uniq crashes : 8
  uniq hangs : 500+
map coverage
  map density : 7.19% / 17.87%
  count coverage : 5.46 bits/tuple
findings in depth
  favored paths : 857 (7.81%)
  new edges on : 1537 (14.01%)
  total crashes : 32 (8 unique)
  total tmouts : 1.90M (523+ unique)
path geometry
  levels : 2
  pending : 10.8k
  pend fav : 849
  own finds : 5298
  imported : n/a
  stability : 99.97%
[cpu000: 53%]
  
```

شکل ۳-۱: اجرای AFL بر روی ابزار mutool از نرم‌افزار MuPDF [۲۵]. این تصویر جزئیات اجرا بعد از گذشت ۵۵ روز از آغاز فرایند آزمون فازی را نشان می‌دهد. آزمون تا زمانی که کاربر *ctrl+z* را فشار ندهد، اجرا می‌شود.

## ۱-۱-۳ معماری AFL

شکل ۳-۲ مؤلفه‌های اصلی فازر AFL و ترتیب استفاده از آنها در هنگام آزمون فازی را نشان می‌دهد. در این معماری چهار مؤلفه مشاهده می‌شود:

afl-gcc • یک جایگزین برای gcc یا clang استاندارد است که برای کامپایل کد منبع SUT استفاده می‌شود و خروجی آن به afl-as ارسال می‌شود. در رویکرد جعبه سفید استفاده مرحله اول کامپایل

<sup>1</sup>Component

با afl-gcc است.

**afl-as**. کد کامپایل شده با afl-gcc را با تزریق کدهای اسembly ابزارگذاری می‌کند. ابزارگذاری به نحوی است که پوشش انشعباب یا پرش را ضبط می‌کند. خروجی afl-as فایل دودویی قابل اجرای SUT است که سپس توسط afl-fuzz استفاده می‌شود.

**afl-fuzz**. همانطور که انتظار می‌رود، هسته اصلی فازر است که عملیات فاز ورودی را انجام می‌دهد. این ابزار فایل دودویی و داده آزمون ورودی را دریافت و با توجه به اطلاعات دریافتی از afl-analyze فرایند آزمون را ادامه می‌دهد. این مؤلفه برای آزمون فازی جعبه سیاه، مستقیماً به کار می‌رود. همچنین مسئول چاپ واسط کاربری روی ترمینال است.

**afl-analyze**. اثر ورودی اجرا شده بر پوشش کد را بررسی می‌کند و اطلاعات مربوط به پوشش کد را به عنوان بازخورده به afl-fuzz می‌دهد تا برای جابه‌جایی‌های بهتر در ادامه استفاده کند.

حلقه بازخورده که در شکل ۲-۲ وجود دارد نشان دهنده تکاملی بودن فرایند آزمون فازی در AFL است. AFL در هسته خود از الگوریتم ژنتیک استفاده می‌کند. یک فایل ورودی جهش یافته، مفید و مورد توجه است اگر قسمت‌های جدیدی از کد دودویی را اجرا کرده باشد یا تعداد اجرای کدهای قبلی مشاهده شده را افزایش داده باشد. این ویژگی با عنوان Input Gain شناخته می‌شود [۲۰]. AFL، سپس این داده آزمون را به انتهای SUT صفت داده‌های آزمون، اضافه می‌کند. به این ترتیب حاصل فرایند آزمون فازی در AFL، علاوه بر اجرای و اندازه‌گیری پوشش کد، یک مجموعه داده آزمون جهش یافته با ویژگی‌های متمایز است.

برای مقایسه نحوه عملکرد AFL و یک فازر تصادفی، جزئیات الگوریتم‌های این دو فازر را مرور می‌کنیم. الگوریتم ۱-۳، فرایند فاز ورودی در یک فازر تصادفی و الگوریتم ۲-۳ همین فرایند را در AFL نشان می‌دهد. تابع *Mutate* یک بایت از ورودی را به صورت درجا، با استفاده از فنونی مثل وارون‌کردن بیت، وارون‌کردن بایت، چرخش بیت یا عملیات ریاضی و منطقی، جابه‌جا می‌کند. تابع *Execute* برنامه را با ورودی جابه‌جا شده اجرا و خرابی‌های احتمالی را گزارش می‌دهد. خطوط سایه زده شده در دو الگوریتم (خطوط ۱۰، ۱۴ و ۱۵) قسمت‌های متفاوت را نشان می‌دهد [۲۰].

## AFL ۲-۱-۳ مشکلات

آزمون فازی از لحاظ محاسباتی سنگین است. یعنی حتی یک Input Gain کوچک نیاز نیازمند هزارها تا میلیون‌ها جابه‌جایی است [۲۰]. از طرفی همه جابه‌جایی‌های یکسان نیستند. AFL با بهره‌گیری از بازخورد داده‌های آزمون بهتری را انتخاب می‌کند، اما همچنان آنها را به صورت تصادفی جهش می‌دهد یا جابه‌جا می‌کند.

## الگوریتم ۱-۳ ] Basic-Random Fuzzing [ ۲۰

**Input:** *Seeds*, Target program *P***Output:** *MaliciousInputs*

```

1   for Seed  $\in$  Seeds do
2     for iterations  $\leftarrow$  0 to limit do
3       input  $\leftarrow$  Seed
4       length  $\leftarrow$  len(Seed)
5       mutations  $\leftarrow$  RandInt(length)
6       for mut  $\leftarrow$  0 to mutations do
7         byte  $\leftarrow$  RandInt(length)
8         mutate(input, byte)
9       end
10      result  $\leftarrow$  Execute(P, input)
11      if result is crash then
12        Append input to MaliciousInputs
13      end
14
15
16    end
17 end

```

در نتیجه تعداد زیادی داده آزمون تکراری تولید می‌شود که لزوماً تأثیری بر بهبود آزمون ندارند. از طرفی در ساختارهای پیچیده، تغییر برخی قسمت‌ها سبب می‌شود تا داده آزمون ورودی نامعتبر شود و سریعاً توسط تجزیه‌گر مردود اعلام گردد. در نتیجه تعداد زیادی داده آزمون هدر رفته خواهیم داشت. AFL افزوده<sup>۱</sup> [ ۲۰ ] مسئله انتخاب تصادفی مکان‌های جایه‌جایی را مورد مطالعه و راهکارهایی برای هوشمندسازی آن ارائه داده است (بخش ۲-۳).

محدودیت دیگر AFL قابلیت حمل<sup>۲</sup> پایین آن است. AFL به خاطر استفاده از فراخوانی‌های سیستمی سیستم عامل Linux تنها در توزیع‌های آن قابل استفاده است. نسخه‌ای از AFL، تحت عنوان WinAFL<sup>۳</sup> برای سیستم عامل ویندوز توسعه داده شده است که البته سرعت پایین و سربار بالایی دارد. WinAFL برای افزایش

<sup>1</sup>Augmented<sup>2</sup>Portability<sup>3</sup><https://github.com/ivanfratric/winafl>

## الگوریتم ۲-۳ AFL Fuzzing [۲۰]

**Input:**  $Seeds$ , Target program  $P$ **Output:**  $MaliciousInputs$ 

```

1 for  $Seed \in Seeds$  do
2   for  $iterations \leftarrow 0$  to  $limit$  do
3      $input \leftarrow Seed$ 
4      $length \leftarrow \text{len}(Seed)$ 
5      $mutations \leftarrow \text{RandInt}(length)$ 
6     for  $mut \leftarrow 0$  to  $mutations$  do
7        $byte \leftarrow \text{RandInt}(length)$ 
8        $\text{Mutate}(input, byte)$ 
9     end
10     $result, cov \leftarrow \text{Execute}(P, input)$ 
11    if  $result$  is crash then
12      Append  $input$  to  $MaliciousInputs$ 
13    end
14    if  $\text{HasInputGain}(cov)$  then
15      Append  $input$  to  $Seeds$ 
16    end
17  end
18 end

```

سرعت پیشنهاد می‌کند که تنها یک تابع از برنامه که هدف اصلی آزمون فازی است، ابزارگذاری شود.

به دلیل مشکلاتی که اشاره شد، اجرای AFL و هرگونه فازر مشابه آن، روی نرم‌افزارهایی مثل PDF‌خوان‌ها که ساختار ورودی آنها پیچیده است به پوشش کد به مراتب کمتری دست می‌یابد که حتی با افزایش زمان آزمون نیز نتایج بهبود چندانی نمی‌کند. در این پایان‌نامه یک روش ترکیبی برای تولید داده آزمون ارائه می‌دهیم که پوشش کد بهتری نسبت به فازرهای مبتنی بر جایه‌جایی محض برای ساختارهای پیچیده فراهم کرده و در عین حال ویژگی‌های مثبت جایه‌جایی در فایل‌های دودویی را نیز به کار می‌بندد.

## ۲-۳ فازر AFL افزوده

AFL افزوده [۲۰] تلاش می‌کند با استفاده از فنون یادگیری ماشینی مکان‌های مناسبی را برای جابه‌جایی بایت‌ها پیدا کند. چارچوب ارائه شده در این کار شامل فازر AFL و یک مدل است که مکان‌های مفید برای جابه‌جایی را مشخص می‌کند. در طول اجرا فازر ابتدا مدل را برای اخذ آدرس مکان‌های جهش پرس‌جو می‌کند و جابه‌جایی‌های فازر را روی مکان‌های بازگردانیده شده متمرکز می‌کند.

برای آموزش مدل فایل ورودی از مجموعه دانه اولیه، فایل جهش یافته و میزان پوشش کد هر دو فایل پس از اجرا توسط SUT مورد نیاز است. در فرایند آموزش اگر پوشش کد فایل جهش یافته و فایل والد یکسان باشد (یعنی میزان Input Gain صفر باشد)، مکان‌های جابه‌جا شده مکان‌های امیدبخشی نخواهند بود و چنان‌چه پوشش کد فایل جابه‌جا شده افزایش داشته باشد (یعنی میزان Input Gain بزرگ‌تر از صفر باشد)، آنگاه جابه‌جایی‌ها امیدبخش محسوب می‌شوند. با تعریف یکتابع خطای که بسته به مقدار Input Gain امتیازهایی به هریک از دو حالت بالا نسبت می‌دهد، مدل در حالت بانتظارت آموزش می‌بیند. ورودی مکان‌های جهش در فایل و خروجی میزان امتیاز کسب شده بر اثر هریک از این مکان‌ها است. صورت‌بندی ریاضی مطالب عنوان شده، به‌طور مفصل در [۲۰] ذکر شده است.

یک ویژگی مثبت این مقاله آموزش و استفاده از چندین مدل متنوع یادگیری ژرف و نیز آزمون چندین برنامه هدف در انجام آزمایش‌ها است که سبب می‌شود تأثیر مدل‌های مختلف را بتوان با یکدیگر مقایسه کرد. برای ایجاد مجموعه آموزش، فازر AFL با تعداد ۱۸۰ دانه اولیه برای هر SUT به مدت ۲۴ ساعت اجرا گردیده و اطلاعات لازم ضبط شده‌اند.

الگوریتم ۳-۳، فرایнд فاز ورودی و انجام آزمون فازی در AFL افزوده را نشان می‌دهد. خطوط سایه زده شده (خطوط ۲، ۱۱، ۱۲ و ۱۳) قسمت‌های اضافه شده به الگوریتم اصلی AFL هستند. چون جابه‌جایی‌های AFL در حالت عادی به صورت تصادفی است (خطوط ۷ و ۸ در الگوریتم ۲-۳ و خطوط ۸ و ۹ در الگوریتم ۳-۳)، تغییر انجام شده در AFL افزوده بدین صورت است که ابتدا مکان‌های مناسب جابه‌جایی برای یک ورودی را توسط مدل پیش‌بینی می‌کند، سپس اجازه می‌دهد تا AFL ورودی را جابه‌جا کند. حال چنان‌چه مجموع شباهت جابه‌جایی‌های AFL با جابه‌جایی‌های گزارش شده که مطلوب مدل است، کمتر از یک حد آستانه  $\alpha$  باشد (خط ۱۲ الگوریتم ۳-۳)؛ این داده آزمون تولیدی برای اجرا ارسال نمی‌گردد. محاسبه شباهت دو رشته بیتی نیز با استفاده از ترکیب عطفی آنها (خط ۱۲ الگوریتم ۳-۳) به راحتی امکان‌پذیر است. چنان‌چه دو رشته مشابه نباشند مجموع بیت‌های ترکیب عطفی آنها کمتر از مجموع بیت‌های هریک از آنها خواهد بود و بالعکس.

## الگوریتم ۳-۳ Augmented-AFL Fuzzing

**Input:**  $Seeds$ , Target program  $P$

**Output:**  $MaliciousInputs$

```

1 for  $Seed \in Seeds$  do
2    $bytemask \leftarrow \text{QueryModel}(Seed)$ 
3   for  $iterations \leftarrow 0$  to  $limit$  do
4      $input \leftarrow Seed$ 
5      $length \leftarrow \text{len}(Seed)$ 
6      $mutations \leftarrow \text{RandInt}(length)$ 
7     for  $mut \leftarrow 0$  to  $mutations$  do
8        $byte \leftarrow \text{RandInt}(length)$ 
9        $\text{Mutate}(input, byte)$ 
10    end
11     $diff \leftarrow input \oplus Seed$ 
12    if  $\sum diff \wedge bytemask < \alpha$  then
13      Continue
14    end
15     $result, codecov \leftarrow \text{Execute}(P, input)$ 
16    if  $result$  is crash then
17      Append  $input$  to  $MaliciousInputs$ 
18    end
19    if  $\text{HasInputGain}(codecov)$  then
20      Append  $input$  to  $Seeds$ 
21    end
22  end
23 end
```

### ۱-۲-۳ مشکلات AFL افزوده

الگوریتم AFL افزوده روی چندین قالب فایل و چندین برنامه مختلف، مورد آزمایش قرار گرفته است؛ از جمله قالب‌های فایل XML و PDF. کمترین بهبود گزارش شده بازهم مربوط به قالب فایل PDF و نرم‌افزار MuPDF است. علت آن هم ساختار پیچیده و هم حجم بالای فایل‌های PDF است. میزان درصد پوشش کد گزارش شده برای نرم‌افزار MuPDF AFL اصلی ۱۱/۶۳ و توسط AFL افزوده در بهترین مدل ۱۱/۸۰ بوده که افزایش ناچیز ۱۷٪ درصدی را نشان می‌دهد. در بخش نتایج این مقاله همچنین هیچ ارزیابی روی دقت مدل‌های مختلف آموزش داده شده انجام نشده و به یک نتیجه‌گیری کلی بسته شده است.

یک تأکید مهم که در مستندات AFL به آن اشاره شده است کوچک نگاه داشتن فایل‌های مجموعه دانه اولیه است (توصیه شده است که از فایل‌هایی با حجم زیر ۱ کیلوبایت استفاده شود)؛ زیرا، در صورتی که از فایل‌های با حجم بالایی استفاده شود سرعت فازر بهشدت کاهش می‌یابد. در واقع جابه‌جایی‌های بسیاری باقیستی روی یک دانه اعمال گردد تا یک داده آزمون جدید ایجاد شود. این در حالی است که برخی ساختارها مثل PDF ذاتاً حجم بالایی دارند. مثلاً یک فایل PDF تنها حاوی عبارت Hello World حجمی در حدود ۱ کیلوبایت دارد. برای این ساختارها، تولید یک فایل جدید از ابتدا، مثلاً از روی یک مدل، سرعت تولید داده آزمون را افزایش می‌دهد. استفاده از یک روش ترکیبی یعنی ثابت نگه داشتن بخش‌هایی از فایل و تولید بخش‌های دارای اهمیت در ساختار یک فایل سرعت تولید داده را حتی بیشتر از تولید یک فایل از ابتدا هم افزایش می‌دهد. روش ارائه شده در این پایان‌نامه یک روش ترکیبی است.

### ۳-۳ یادگیری و فاز

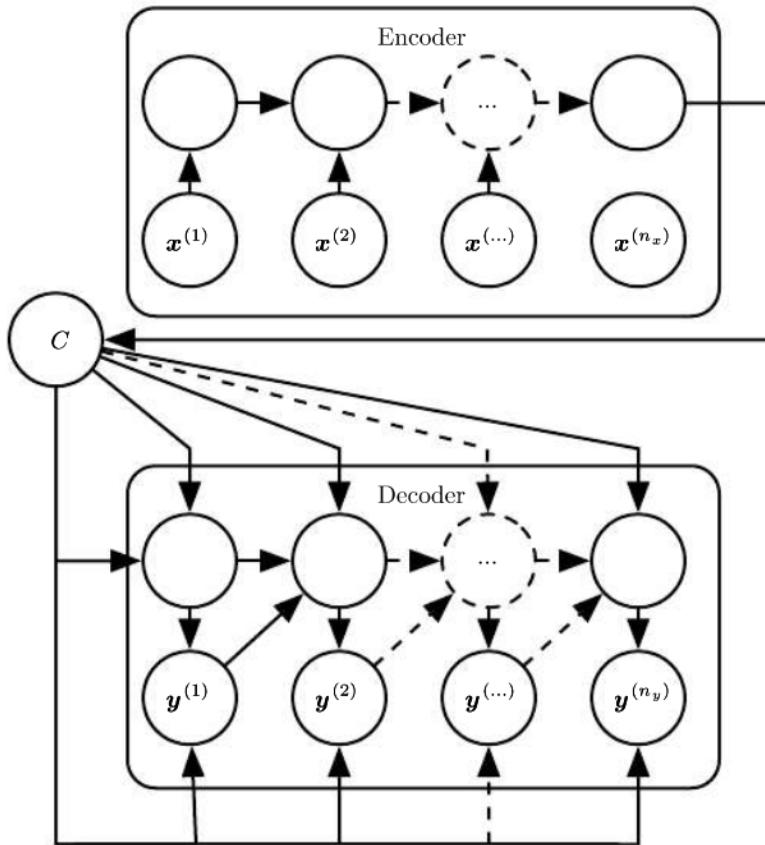
مقاله یادگیری و فاز<sup>۱</sup> [۱۱] روشی برای تولید داده آزمون جهت استفاده در آزمون فازی بر مبنای RNN و مدل کدگذار-کدگشا<sup>۲</sup> [۱۷، ۱۸] ارائه کرده است. در این مقاله ساختار فایل PDF و مرورگر وب Edge<sup>۳</sup> شرکت مایکروسافت، برای آزمون انتخاب شده‌اند. ایده اصلی یادگیری یک مدل مولد زبان روی مجموعه‌ای از ویژگی‌های اشیای PDF با داشتن مجموعه‌ای از نمونه‌های اولیه است. ساختار قالب فایل PDF و اشیای داده‌ای آن، در پیوست آ توپیچ داده شده است.

مدل کدگذار-کدگشا [۱۸، ۱۷] از دو RNN تشکیل شده است. یک شبکه کدگذار که یک توالی ورودی با بعد متغیر را به یک نمایش بعد ثابت تبدیل می‌کند (بردار زمینه) و یک شبکه کدگشا که یک توالی ورودی بعد

<sup>1</sup>Learn&Fuzz

<sup>2</sup>Encoder-Decoder

<sup>3</sup>مرورگر جدید شرکت مایکروسافت که همراه با سیستم عامل ویندوز ۱۰ معرفی شد.

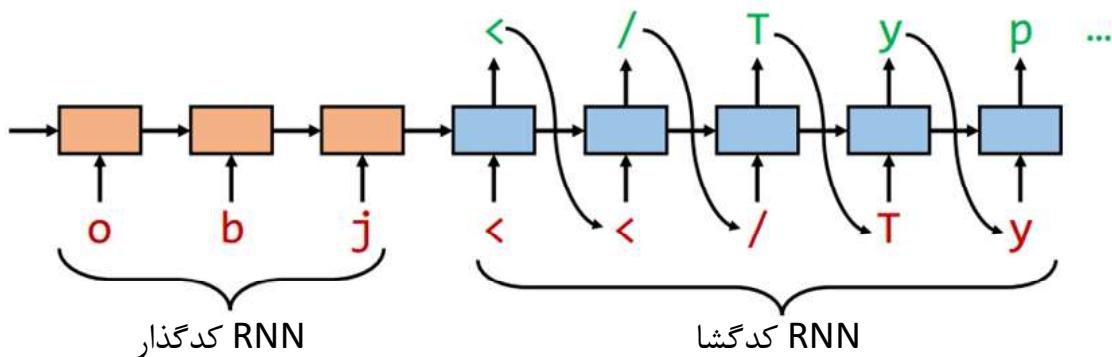


**شکل ۳-۳:** مثالی از معماری مدل کدگذار-کدگشا، متشکل از دو RNN و بردار زمینه  $C$ . این مدل برای یادگیری تولید توالی خروجی  $< y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(n_y)} >$  از روی بردار زمینه، که نماینده توالی ورودی  $< x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(n_x)} >$  است، به کار می‌رود [۳۸].

ثابت را می‌گیرد و به یک توالی ورودی با بعد متغیر تولید می‌کند. شبکه کدگشا توالی‌های خروجی را با استفاده از کاراکتر خروجی پیش‌بینی شده که در مرحله‌زمانی  $t$  به عنوان کاراکتر ورودی برای مرحله‌زمانی  $t+1$  تولید شده است، تولید می‌کند. یک بازنمایی از معماری این مدل در شکل ۳-۳ نشان داده شده است. این مدل نخستین بار در وظیفه ترجمه ماشینی استفاده شده است [۱۸]. با توصیف بیان شده، این معماری امکان یادگیری توزیع شرطی ( $p(y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(n)}) | < x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(n)} >$ ) را فراهم می‌کند.

### ۳-۳-۱ آموزش مدل کدگذار-کدگشا

مدل کدگذار-کدگشا در روش یادگیری و فاز، با استفاده از تکه اشیای PDF که هر یک به صورت توالی‌ای از کاراکترها در نظر گرفته شده است، آموزش می‌بیند. برای آموزش، ابتدا همه فایل‌های حاوی اشیای PDF



[۱۱] شکل ۳-۴: مدل RNN کدگذار-کدگشا برای تولید اشیای داده‌ای PDF

به‌شکل یک توالی از کاراکترها به یکدیگر الحاق می‌شوند که منجر به ایجاد یک توالی بزرگ از کاراکترها به صورت  $s^{(1)}, s^{(2)}, \dots, s^{(n)}$  خواهد شد. سپس توالی  $\hat{s}$  به چند زیر توالی آموزشی با طول ثابت  $d$  شکسته می‌شود به نحوی که  $i$ -امین نمونه آموزشی عبارت است از:  $\hat{s}[i * d : (i + 1) * d]$  که  $\hat{s}[l : u]$  یک زیرتوالی از  $\hat{s}$  بین اندیس‌های  $l$  و  $u$  را نشان می‌دهد. در ادامه، توالی خروجی برای هر توالی آموزشی  $t_i$  عبارت است از توالی ورودی که یک واحد به سمت راست شیفت داده شده است؛ یعنی:  $\hat{s}[(i + 1) * d : (i + 1) * d + 1] = o_{t_i}$ . مدل سپس به صورت انتهاه‌انتها<sup>۱</sup> آموزش می‌بیند تا یک مدل مولد روی مجموعه همه توالی‌های ایجاد شده، یادگیری شود. شکل ۳-۴ مدل ارائه شده در این مقاله را نشان می‌دهد. از آنجایی که مدل بحث شده در حالت بدون نظارت آموزش می‌بیند، برچسب‌های تولید شده به‌طور صریح برای مشخص کردن این که مدل یادگرفته شده چه قدر خوب عمل می‌کند، آزموده نشدن. به جای آن چندین مدل آموزش داده می‌شود که در تعداد دوره‌ها (بخش ۳-۲) متفاوت هستند. مدل در ۱۰، ۲۰، ۳۰، ۴۰ و ۵۰ دوره آموزش داده شده است. در تنظیمات شبکه ارائه شده در این مقاله، یادگیری روی هر دوره حدود ۱۲ دقیقه طول می‌کشد و بنابراین مدل با ۵۰ عدد حدوداً ۱۰ ساعت زمان نیاز دارد تا کامل شود. شبکه به صورت RNN با ۲ لایه مخفی که هر لایه ۱۲۸ سلول LSTM دارد، است.

### ۳-۳-۳ تولید داده‌های جدید

از مدل یادگیری شده برای تولید اشیای جدید PDF استفاده شده است. راهبردهای مختلفی برای تولید اشیا، بسته به راهبرد نمونه‌برداری که برای نمونه‌برداری از توزیع یادگیری شده به کار می‌رود، وجود دارند. در این مقاله، با یک پیشوند<sup>۲</sup> از توالی "obj" (که آغاز یک شی را مشخص می‌کند) شروع و سپس مدل را پرس و جو

<sup>1</sup>End to End

<sup>2</sup>Prefix

### فصل ۳. کارهای مرتبط

کرده تا یک توالی از کاراکترهای خروجی تولید گردد، تا زمانی که مدل "endobj" را که مشخص کننده انتهای یک شی است، تولید نماید. سپس این مقاله، سه راهبرد مختلف را برای تولید اشیای جدید از روی مدل، استفاده کرده است [۱۱]:

● **NoSample**. در این راهبرد تولید، کاراکتر پیش‌بینی شده با بالاترین احتمال به عنوان کاراکتر بعدی انتخاب می‌شود که در واقع انتخابی حریصانه<sup>۱</sup> است. این راهبرد برای تولید اشیای PDF‌ای که خوش‌شکل<sup>۲</sup> و سازگار<sup>۳</sup> هستند، کاملاً نتیجه‌بخش است ولی تعداد اشیائی را که می‌توان تولید کرد، محدود می‌کند. با دادن یک پیشوند مثل "jz ۰obj" بهترین توالی از کاراکترهای بعدی به صورت یکتا مشخص می‌شود و بنابراین این راهبرد همان شی را نتیجه می‌دهد. این محدودیت مغایر با اهداف آزمون فازی بوده و مانع مفید بودن استفاده از آن در آزمون فازی می‌شود.

● **Sample**. در این راهبرد تولید، به جای انتخاب محتمل‌ترین کاراکتر پیش‌بینی شده، از توزیع یادگرفته شده، برای نمونه‌برداری کاراکتر بعدی در توالی‌ای که پیشوند آن داده شده است، استفاده می‌شود. راهبرد نمونه‌برداری قادر به تولید مجموعه گوناگونی از اشیا با ترکیب الگوهای مختلفی که از اشیای موجود یادگرفته شده است، خواهد بود. به دلیل نمونه‌برداری اشیای تولید شده، تضمینی نیست که آنها به میزان راهبرد قبلی، خوش‌شکل باشند، که این ویژگی از منظر آزمون فازی مفید است.

● **SampleSpace**. این راهبرد ترکیب دو راهبرد قبلی است. SampleSpace توزیع احتمالی برای تولید کاراکتر بعدی را تنها زمانی که پیشوند توالی کنونی، با فضای خالی خاتمه می‌یابد، نمونه‌برداری می‌کند. برای میان توکن<sup>۴</sup>‌ها (یعنی پیشوندهایی که با فضای خالی خاتمه نمی‌یابند)، بهترین کاراکتر را انتخاب می‌کند (راهبرد NoSample) است که در مقایسه با راهبرد Sample، ورودی‌های خوش‌شکل‌تری تولید می‌شود. چون نمونه‌برداری در پایان کاراکترهای فضای خالی صورت می‌گیرد؛ نمونه‌های جدید نیز در این راهبرد، ساخته خواهند شد.

## ۳-۳-۳ اعمال آزمون فازی

برای فاز داده‌های آزمون تولید شده، مقاله الگوریتم جدیدی تحت عنوان SampleFuzz (الگوریتم ۳-۴)، ارائه کرده است. SampleFuzz توزیع یادگیری شده  $D(x, \theta)$ ، احتمال فازینگ یک کاراکتر  $t_{fuzz}$  و احتمال

<sup>1</sup>Greedy

<sup>2</sup>Well-formed

<sup>3</sup>Consistent

<sup>4</sup>Token

آستانه  $t$  که تصمیم می‌گیرد آیا کاراکتر پیش‌بینی شده اصلاح شود یا نه، را به عنوان ورودی می‌پذیرد. سپس برای توالی خروجی  $seq$ ، الگوریتم  $D(x, \theta)$  را نمونه‌برداری می‌کند تا کاراکتر بعدی یعنی  $c$  و احتمال آن ( $p(c)$ ) را در یک مرحله زمانی مشخص  $t$  به دست آورد. اگر احتمال  $p(c)$  از آستانه فراهم شده توسط کاربر ( $p_t$ ) بالاتر باشد؛ یعنی، اگر مدل مطمئن باشد که کاراکتر بعدی به احتمال زیاد  $c$  است، الگوریتم تصمیم می‌گیرد تا یک کاراکتر دیگر  $c'$  که احتمال کمینه ( $p(c')$ ) را دارد، جایگزین  $c$  کند. البته این اصلاح (فاز) تنها در صورتی که عدد تصادفی تولید شده  $p_fuzz$  از ورودی  $t_fuzz$  مقدار بیشتری داشته باشد، انجام می‌شود، که بدین ترتیب اجازه می‌دهد تا کاربر بتواند بر روی احتمال (درصد) کاراکترهای فاز شده کنترل داشته باشد.

---

### الگوریتم ۴-۳ SampleFuzz [۱۱]

---

**Input:**  $D(x, \theta)$ ,  $t_fuzz$ ,  $p_t$

**Output:**  $seq$

```

1 seq  $\leftarrow$  "obj"
2 while  $\sim seq.EndsWith("endobj")$  do
3    $c, p(c) \leftarrow sample(D(seq, \theta))$  /* Sample c from the learnt distribution */
4    $p_fuzz \leftarrow Random(0, 1)$  /* Random variable to decide whether to fuzz */
5   if  $p_fuzz > t_fuzz \wedge p(c) > p_t$  then
6      $c \leftarrow argmin_{c'}\{p(c') \sim D(seq, \theta)\}$  /* Replace c by c' (with lowest
      likelihood) */
7   end
8    $seq \leftarrow seq + c$ 
9   if  $len(seq) > MaxLen$  then
10     $seq \leftarrow "obj"$  /* Reset the sequence */
11  end
12 end
13 Return  $seq$ 

```

---

الگوریتم SampleFuzz اطمینان می‌یابد که طول شی با عدد  $MaxLen$  محدود شود؛ اما، شرط حلقه *while* الگوریتم تضمین نمی‌کند که همواره خاتمه یابد. با این حال، نویسنده‌گان گزارش کرده‌اند که در آزمایش‌های انجام شده در عمل، الگوریتم همواره پایان یافته است. چیدمان آزمایش‌ها و نیز تحلیل جامع نتایج در [۱۱] آمده است که نشان از برتری الگوریتم SampleFuzz در مقایسه با روش تصادفی و افزایش نسبی

پوشش کد در سطح دستورات برنامه است.

### ۴-۳-۳ مشکلات روش یادگیری و فاز

مقاله یادگیری و فاز برای نخستین بار از یادگیری ماشینی در تولید داده آزمون فازی استفاده کرده است. در عین حال روش ارائه شده در این مقاله دارای مشکلات و کاستی‌هایی است که عبارتند از:

۱. همواره با یک پیشوند ثابت شروع به تولید داده جدید می‌کند که این سبب می‌شود تنوع کمتری حاصل شود. در نتیجه ناچار به پیچیده کردن فرایند نمونه برداری از مدل می‌شود. همچنین این پیشوند کوتاه است و حاوی اطلاعات بعدی یک شی داده‌ای نیست. می‌توان مدل‌هایی با قابلیت پذیرفتن پیشوندهای متفاوت ایجاد کرد.

۲. مدل کدگذار-کدگشا معمولاً برای نگاشت توالی‌های با طول و دامنه‌های متفاوت به یکدیگر، استفاده می‌شود. بارزترین مثال وظیفه ترجمه ماشینی است که هدف آن نگاشت جمله‌های یک زبان به زبان دیگر است [۱۷]. در این وظیفه، طول جمله زبان مقصد لزوماً برابر طول معادل همان جمله در زبان مبدأ نیست. به همین خاطر نیازمند معماری پیچیده کدگذار-کدگشا هستیم. وظیفه یادگیری خودکار ساختار فایل بیشتر به ایجاد یک مدل زبانی روی الفبای زبان فایل ورودی شباهت دارد که در نتیجه استفاده از یک مدل زبانی عصبی در این وظیفه، منطقی‌تر و احتمالاً نتیجه‌بخش‌تر به نظر می‌رسد. چنان‌چه در روش یادگیری و فاز هم می‌بینیم که ساختار و وظایف شبکه کدگذار و کدگشا به خوبی تفکیک و تفہیم نگردیده است و صرفاً به پیچیده‌تر شدن مدل انجامیده است.

۳. فقط داده‌های متنی فایل تولید و فاز شده‌اند در حالی که یک فایل بازیزی در حالت کلی حاوی ترکیبی از داده‌های متنی و غیرمتنی است. می‌توان یک روش ترکیبی برای این منظور ارائه داد.

۴. الگوریتم فاز ارائه شده، همان‌طور که دیدیم، ممکن است هیچ‌گاه پایان نیابد یا اجرای الگوریتم برای تولید یک نمونه خیلی طولانی شود که خوب به نظر نمی‌رسد. می‌توان با تعریف شرایطی از این اتفاق جلوگیری کرد. خاتمه‌یافتن در هر صورت، یک ویژگی است که در تعریف اصطلاح الگوریتم وجود دارد و از این بابت این مسئله بسیار مهم تلقی می‌شود.

۵. معماری مدل پیشنهادی به نحوی است که توالی‌های تولید شده برای آموزش در برگیرنده همه اطلاعات و وابستگی‌های ساختار فایل نیستند. در هنگام تولید توالی‌های اموزشی هر بار به اندازه  $d$  واحد از روی مجموعه داده پرس می‌کنیم. اگر  $d$  کوچک باشد، توالی‌های کوچکی خواهیم داشت که لزوماً

شامل یک وابستگی طولانی نیستند و اگر  $d$  بزرگ انتخاب شود، تعداد وابستگی‌های کمتری تشخیص داده می‌شود. به این نکته باقیستی توجه کرد که وابستگی‌های فایل‌های با قالب مختلف، یکسان نیستند. می‌توان راهکارهای بهتری در نحوه تولید ورودی و خروجی شبکه یافت.

۶. در هر بار اجرا فقط یک نمونه را تولید می‌کند و بنابراین نمونه‌های تولید شده در چندین اجرای متواالی کاملاً مستقل از یکدیگر هستند. ممکن است چند نمونه متواالی بهم وابستگی داشته باشند در این صورت باقیستی بتوان به نحوی این پارامتر را در تولید نمونه‌ها گنجاند.

۷. و بالآخره در مقاله یادگیری و فاز تنها از یک مدل یادگیری ژرف برای تولید داده آزمون استفاده شده است که می‌توان با انتخاب مدل‌های مختلف یا تغییر در ابرپارامترهای در مورد تأثیر نوع مدل نیز بحث کرد و آزمایش‌های جدید را تجربه نمود.

## ۴-۳ دیگر کارهای مرتبط

تعداد دیگری کار در زمینه تولید داده آزمون در فازهای و بهویژه در زمینه یادگیری ساختار فایل انجام شده است. در فصل اول، بهطور خلاصه به برخی از آنها اشاره کردیم [۱۵، ۱۶]. بیشتر این روش‌ها بسیار پیچیده بوده، درک و پیاده‌سازی آنها مشکل است و سربار زیادی را به عملیات آزمون فازی یا عملیات پیش‌پردازش، عملیات پیش از انجام آزمون فازی، تحمیل می‌کنند. در ضمن اغلب آنها در نهایت محدود به ساختارهای ورودی خاصی می‌شوند. تمرکز ما بر روی روش‌هایی است که کاملاً عملیاتی بوده و قابل استفاده بر روی نرم‌افزارهای دنیای واقعی هستند.

در آزمایشگاه تحقیقاتی مهندسی معکوس دانشگاه علم و صنعت ایران، تعدادی زیادی پروژه، کار پژوهشی و پایان‌نامه در ارتباط با آزمون فازی و تولید داده آزمون، نگارش شده است<sup>۱</sup>. یعقوبی [۴۸] یک روش خودکار تولید داده آزمون با استفاده از الگوریتم ژنتیک برای آزمون فازی جعبه سیاه مرورگرهای وب، ارائه داده است. هدف الگوریتم در این فرایند تکاملی، تشخیص مجموعه‌ای از برچسب‌های HTML است که حداکثر درهم ریختگی حافظه و کاهش سرعت اجرایی مرورگر را موجب شوند. در واقع دسته برچسب‌های HTML به عنوان صفحات ورودی هر مرورگر، طوری ساخته می‌شوند که در هر مرحله نسبت به دسته برچسب‌های قبل، میزان مصرف حافظه اصلی را بالا ببرند.

<sup>۱</sup> فهرست کاملی از کارهای مرتبط انجام شده، از طریق وبسایت رسمی آزمایشگاه به نشانی <http://parsa.iust.ac.ir> قابل دسترسی هستند. همچنین فهرستی سازماندهی شده و کامل در ارتباط با منابع مختلف آزمون فازی در نشانی <https://github.com/seccfigo/Awesome-Fuzzing> وجود دارد.

امکان بهره‌گیری از این روش برای آزمون فازی و شناسایی خطاهای آسیب‌پذیری نرم‌افزارهایی که کد منبع آنها در دسترس نیست نیز وجود دارد. اما تمرکز این روش روی تولید برچسب‌های HTML بوده و سرعت همگرایی آن کند است. علاوه بر این، تنها هدف الگوریتم رنتیک در اینجا، افزایش حافظه تعریف شده است در حالی که می‌دانیم صرف افزایش حافظه منجر به کشف خطای نمی‌شود. در سیستمی با حافظه پایین ممکن است سیستم عامل برنامه را پیش از سقوط متوقف سازد یا برنامه فقط بر اثر کمبود حافظه نتواند یک ورودی به اندازه کافی بزرگ را پردازش کند. در ساختارهای پیچیده که تجزیه‌گر سختگیری دارند، تولید داده به این روش بسیار زمانبر است؛ زیرا، در عمل خیلی از داده‌های آزمون تولیدی مورد پردازش قرار نمی‌گیرند و حافظه‌ای به آنها تخصیص داده نمی‌شود.

امینی [۴۹] یک روش تکاملی با هدف حل مشکل بزرگی فضای ورودی، برای تولید داده آزمون ارائه داده است. در روش‌هایی مشابه این روش مسئله تولید داده آزمون به صورت یک مسئله جستجو تعریف می‌گردد و سپس از الگوریتم‌های اکتشافی برای یافتن یک حل بهینه استفاده می‌شود. در راهکار امینی، زمانی که یک داده آزمون یک الگوی آسیب‌پذیر<sup>۱</sup> را شناسایی می‌کند، داده آزمون بعدی به‌گونه‌ای تولید می‌شود که منجر به افشاری آن آسیب‌پذیری گردد. در نتیجه احتمال کشف آسیب‌پذیری‌های شناسایی شده بهبود خواهد داشت. الگوی آسیب‌پذیر در سطح کد منبع برنامه قابل شناسایی بوده و بنابراین این روش در فازهای جعبه سفید قابل استفاده است. البته می‌توان آن را به فازهای جعبه خاکستری هم تعمیم داد. روش مذکور روی برنامه‌های دنیای واقعی ارزیابی نشده است. همچنین یافته‌های آزمایش‌های آن، با هیچ روش هوشمند تولید داده آزمونی، مقایسه نگردیده است.

## ۳-۵ خلاصه

جدول ۱-۳، یک جمع‌بندی از مزایا و معایب سه کار اصلی معرفی شده در این فصل [۲۰، ۱۹، ۱۱] را در کنار یکدیگر نشان می‌دهد. همچنین فازر FileFuzz [۲]، که یک فازر قالب فایل تصادفی و بسیار اولیه است به عنوان یک فازر مبنایی برای مقایسه، در جدول آورده شده است. البته تا به امروز، فازرهای بسیار زیادی در زمینه آزمون فازی قالب فایل توسعه داده شده‌اند که مجال بررسی همه آنها در این پایان‌نامه نبوده و در اینجا صرفاً آن دسته از کارهایی را بیان کردیم که قصد داریم در روش پیشنهادی خود مشکلات موجود در آنها را تا حد امکان برطرف کنیم. برخی از این فازرهای مانند [۱۰] به صورت داخل سازمانی و یا تجاری ارایه شده‌اند و کد منبع آنها در دسترس نیست. عدم دسترسی به کد منبع یک فازر و در نتیجه عدم امکان تغییر در پیمانه‌های مختلف آن را می‌توان انگیزه‌ای برای توسعه فازرهای جدید دانست.

<sup>۱</sup>Vulnerable Pattern

## جدول ۱-۳: مقایسه کارهای مرتبط

فارز	مشخصه‌ها	مزایا	معایب
[۲] FileFuzz	• مبتنی بر جابه‌جایی • عدم نیاز به اطلاع داشتن به صورت کاملاً تصادفی از ساختار ورودی و ساختار • پوشش کد بسیار پایین و غیر قابل قبول، برای بیشتر نرم‌افزارها پس از آزمون	• سادگی پیاده‌سازی • تعیین مکان جابه‌جایی‌ها	SUT
[۱۹] AFL	• مبتنی بر جابه‌جایی بهره‌گیری از الگوریتم رنتیک • قابل استفاده به صورت تولید تعداد زیادی داده آزمون هدر رفته • سرعت پایین فاز در فایل‌های با حجم بیشتر از ۱ کیلوبایت	• بیشینه کردن پوشش کد با به صورت تصادفی • قابل استفاده به صورت جعبه‌سیاه، جعبه‌خاکستری و آزمون هدر رفته	• تعیین مکان جابه‌جایی‌ها
AFL افزوده [۲۰]	• مبتنی بر جابه‌جایی • مبتنی بر بازخورد • جعبه‌خاکستری	• پوشش کد پایین برای جابه‌جایی‌ها از طریق یک نرم‌افزارهای با ساختار مدل ارزش‌دهی به بایت‌ها ورودی پیچیده	• تعیین مکان (آدرس نسبی) نرم‌افزارهایی با ساختار مدل ارزش‌دهی به بایت‌ها
Learn&Fuzz [۱۱]	• مبتنی بر تولید • جعبه سفید	• یادگیری گرامر فایل و تولید داده‌های آزمون جدید دو دویی • شروع تولید داده آزمون با یک پیشوند ثابت کدگشنا	• یادگیری گرامر فایل و حذف کامل داده‌های دو دویی • شروع تولید داده آزمون با یک پیشوند ثابت کدگشنا
		• تعیین مکان فاز داده آزمون توسط مدل یادگیری شده الگوریتم فاز	• تعیین مکان فاز داده آزمون توسط مدل یادگیری شده الگوریتم فاز
		• تعیین مقدار فاز شده داده مجموعه داده و دیگر آزمون	• تعیین مقدار فاز شده داده در دسترس نبودن فاز، مجموعه داده و دیگر آزمون
		پارامترهای مهم مربوط به آموزش مدل	پارامترهای مهم مربوط به آموزش مدل

به طور خلاصه، در این فصل ابتدا فازر قالب فایل AFL [۱۹] را به عنوان یک فازر با روش تولید داده آزمون مبتنی بر جابه‌جایی مورد مطالعه قرار دادیم و دیدیم که به علت جابه‌جایی تصادفی بایت‌های فایل، تعداد زیادی داده آزمون بیهوده برای ساختارهای پیچیده ایجاد می‌کند. سپس AFL افزوده [۲۰] را توضیح دادیم از که فنون یادگیری ماشینی برای تعیین محل مکان‌های جابه‌جایی استفاده می‌کند. ایده اصلی در این روش، آن است که در قالب‌های فایل، معمولاً فقط قسمت‌های کوچکی از فایل برای تجزیه‌گر مهم هستند و در نتیجه بیشتر جابه‌جایی آن‌ها منتهی به اجرای مسیرهای جدید SUT می‌گردد. این روش نیز روی قالب پیچیده‌ای مانند PDF بهود ناچیزی داشته است. با این حال روی قالب‌های کاملاً دودویی مثل قالب فایل PNG در زمان برابر به پوشش کد بهتری دست یافته است.

در ادامه فصل، روش یادگیری و فاز [۱۱] را توضیح دادیم که از مدل کدگذار-کدگشا برای تولید و فاز داده آزمون استفاده می‌کند. ایده اصلی این روش یادگیری خودکار گرامر برای یک قالب فایل متنی و سپس تولید داده با روش مبتنی بر تولید است. این روش با یک روش تصادفی که در ابتدای فصل نیز مطرح شد، مقایسه شده و نتایج بهود داشته است. روش یادگیری و فاز روی ابزار متن بسته مرورگر Edge شرکتマイکروسافت و توسط محققین همین شرکت مورد آزمایش قرار گرفته و هیچ‌گونه کدمنبع و مجموعه داده‌ای از آن منتشر نشده است. در مقابل اما فازر AFL متن‌باز و رایگان است. AFL افزوده نیز به تازگی توسط توسعه‌دهنگان آن منتشر شده است.

در پایان این فصل مروری اجمالی بر دو کار پیشین انجام شده در زمینه آزمون فازی و تولید خودکار داده آزمون در آزمایشگاه مهندسی معکوس دانشگاه علم و صنعت ایران، که این پایان‌نامه در آنجا نگارش یافته است، داشتیم. هر دو کار [۴۸، ۴۹] از الگوریتم ژنتیک برای تولید داده‌های آزمون استفاده کرده بودند که مزايا و معایب هریک نیز بیان شد. در این پایان‌نامه ما از الگوریتم‌های یادگیری ژرف برای تولید خودکار داده آزمون استفاده خواهیم کرد. با توجه به مزايا و معایبی که در کلیه روش‌های مطرح در این فصل شناسایی و معرفی کردیم، در فصل بعد یک روش ترکیبی که بتواند از ویژگی‌های خوب جابه‌جایی تصادفی فایل‌های دودویی در کنار تولید فایل‌های متنی براساس مدل، استفاده کند، ارائه خواهیم کرد. هدف بهود برخی کاستی‌های شناسایی شده در کارهای مطرح در این فصل و حل مسائل اصلی اشاره شده در فصل اول است.

## فصل ۴

### روش پیشنهادی

«اولین قانون هر فناوری مورد استفاده در یک کسب و کار این است که إعمال خودکارسازی به یک عملیات کارآمد، کارآمدی را افزایش می‌دهد. دومین قانون این است که إعمال خودکارسازی به یک عملیات ناکارآمد، ناکارآمدی را افزایش می‌دهد.»

#### ✳ بیل گیتس

هدف، در روش پیشنهادی همان‌طور که پیش از این هم گفته شد، ایجاد یک مدل برای یادگیری خودکار گرامر قالب فایل و سپس استفاده از آن در تولید داده‌های آزمون جدید است. معماری مدل یادگیرنده ساختار فایل، نحوه آموزش، مجموعه داده لازم برای آموزش مدل، نحوه تولید داده‌های آزمون و در نهایت چگونگی انجام آزمون فازی، مباحثی هستند که در این فصل به آنها خواهیم پرداخت. در ابتدا به بیان کلیات روش پیشنهادی پرداخته و سپس هریک از مراحل را با جزئیات تشریح می‌کنیم.

#### ۱-۴ کلیات

شكل ۱-۴ یک روندnamی ساده از کلیات مراحل انجام روش پیشنهادی ما را نشان می‌دهد. در ساختار برخی قالب‌های فایل پیچیده مانند PDF، بسیاری از قسمت‌ها به صورت متنی هستند. اما ممکن است قسمت‌های دودویی هم وجود داشته باشد. در روش پیشنهادی خود ما ابتدا کلیه قسمت‌های متنی فایل‌های موجود در

دانه‌های اولیه را استخراج، سپس این قسمت‌ها را به یکدیگر الحاق می‌کنیم. حاصل این کار ایجاد یک پیکره<sup>۱</sup> بزرگ است که در بطن خود مشخصه‌های قالب یک فایل را دارد. در ادامه این مجموعه را مطابق با ضوابط فنون یادگیری ماشینی به سه مجموعه آموزش، ارزیابی و آزمون تقسیم‌بندی با نسبت‌های مشخص می‌کنیم. سپس یک مدل مولد را ایجاد و آن را روی داده‌های مجموعه آموزش، با تعریف یک روش یادگیری، آموزش می‌دهیم. در نهایت از این مدل برای ایجاد داده‌های آزمون جدید و اعمال آزمون فازی استفاده خواهیم کرد. اما چون مدل صرفاً قادر به تولید داده‌های متنی است در هنگام ایجاد یک داده آزمون، با فنوئی که در ادامه خواهیم گفت، قسمت‌های دودویی را نیز به داده آزمون تزریق می‌نماییم.

## ۱-۱-۴ تمایز داده‌های متنی و دودویی

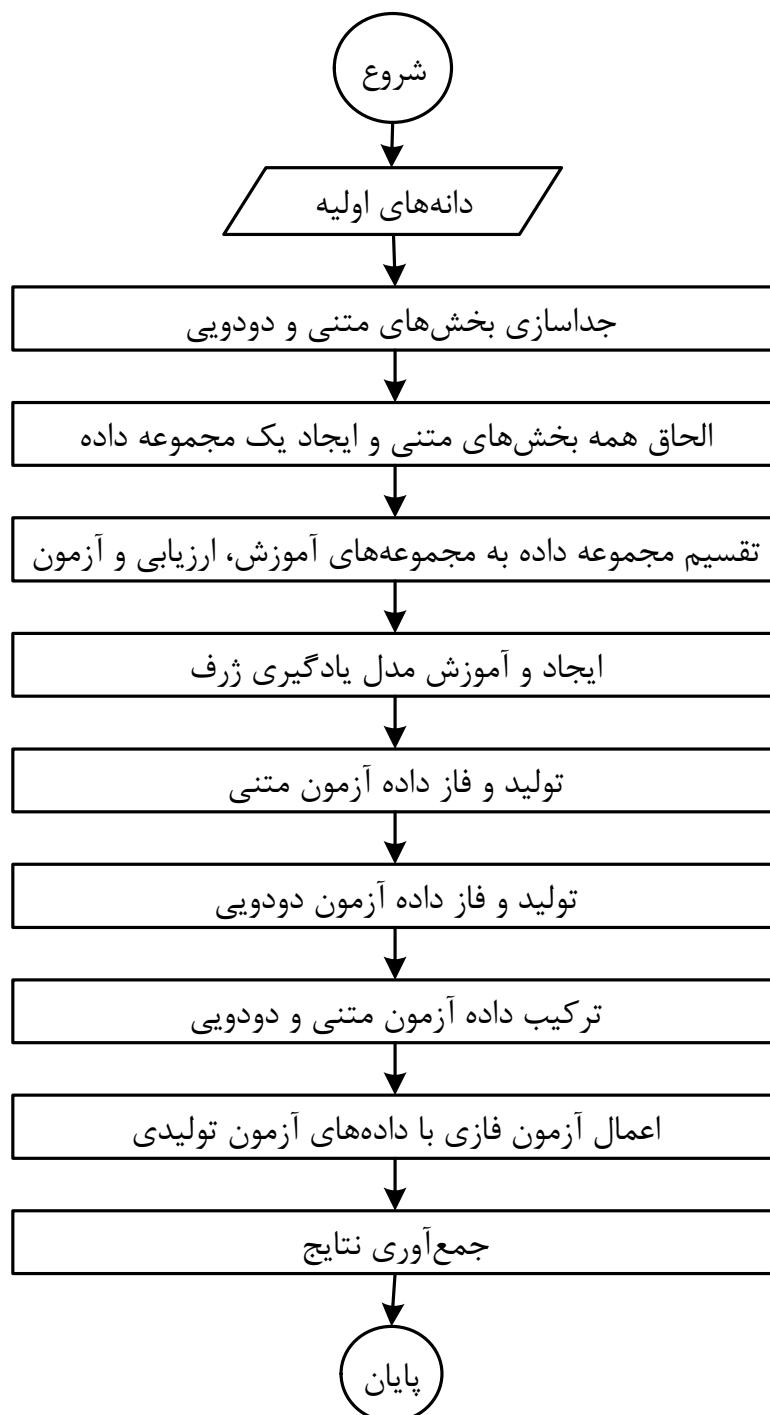
گفتیم مدل مولد را روی بخش متنی قالب فایل ایجاد می‌کنیم. اما مفهوم دقیق متنی و دودویی و معیار جداسازی آنها چیست و دلیل تأکید ما بر این موضوع از کجا نشئت می‌گیرد؟ منظور از داده‌های متنی در اینجا مجموعه کدهای ASCII<sup>2</sup> است. هنگامی که در سطح بیت یک فایل را بررسی می‌کنیم تمایزی میان واژه‌های متنی و دودویی نیست؛ زیرا همه چیز با مفهوم صفر و یک بیان می‌شود که دودویی است. اما اگر فایل را به صورت یک توالی از بایت‌ها در نظر بگیریم، با توجه به اینکه هر بایت ۲۵۵ حالت ممکن خواهد داشت، می‌توانیم آن فایل را یک زبان مشکل از تکرار حداکثر ۲۵۵ نشانه بدانیم. کدهای ASCII حتی این تعداد را به ۱۲۸ نشانه که کاراکترها و نشانه‌های متداول زبان انگلیسی هستند، محدود می‌کنند. مشکل بخش‌های دودویی آن است که اغلب در سطح بیت تفسیر می‌شوند و یادگیری وابستگی‌های آنها با شبکه‌های عصبی، که داده‌های محدودی را در زمان آموزش می‌بینند، امکان‌پذیر نیست. از طرفی هنگامی که نسبت کوچکی از داده‌ها به صورت دودویی در میان قسمت‌های متنی ظاهر می‌شوند می‌توان آنها را با روش‌های جابه‌جایی تصادفی فاز کرد.

ما در روش پیشنهادی خود داده‌های دودویی را از قالب فایل حذف کرده اما به جای آن یک توکن خاص موسوم به توکن دودویی<sup>۳</sup> را جایگزین می‌کنیم. بدین ترتیب در فرایند آموزش مدل توزیع آماری نحوه ظاهر شدن قسمت‌های دودویی نیز فراگیری می‌شود. حال در فرایند تولید داده‌های آزمون جدید مدل مکان‌های احتمالی ظاهر شدن قسمت دودویی را با پیش‌بینی وقوع توکن مربوطه، تعیین می‌کند. سپس توکن را با یک قسمت دودویی که به صورت تصادفی (یا هر روش دیگر) فاز شده است جایگزین می‌کنیم. یعنی وارون عملی که پیش از فرایند آموزش انجام دادیم را پس از فرایند تولید، انجام می‌دهیم. با کلیت گفته شده در اینجا یک روش ترکیبی تولید خودکار داده آزمون خواهیم داشت.

<sup>1</sup>Corpus

<sup>2</sup>American Standard Code for Information Interchange

<sup>3</sup>Binary Token



شکل ۴-۱: روندnamای روش پیشنهادی در حالت کلی.

## ۴-۱-۲ تمايز داده و فراداده

در فصل ۱، تمايز داده و فراداده به عنوان يک مسئله در توليد داده آزمون مطرح گردید. مدل مطرح در روش پیشنهادی اين امكان را فراهم می‌کند تا بتوانيم به صورت احتمالي داده و فراداده را از يكديگر تشخيص دهيم. از آنجايي که فرادادهها در ساختاري يک فایل از پيش تعريف شده و تقربياً ثابت هستند، تكرار آنها به مراتب بيشتر از تكرار دادهها است که محتويات يک فایل را تشکيل می‌دهند. مدل بدین ترتيب در توزيع آماري يادگيري شده احتمال بيشتری را به فرادادهها نسبت می‌دهد. با تعين يک آستانه مشخص برای هر يک از انواع داده و فراداده در هنگام توليد داده آزمون قادر به تصميم‌گيری در مورد نوع آنها هستيم. ما نتيجه اين تصميم را در نحوه فاز داده آزمون جديد بهكار مى‌بريم، بدین صورت که دو الگوريتم فاز با اهداف فاز فراداده برای مرحله تجزيه و فاز داده برای مرحله پرداخت فایل ارائه خواهيم داد.

## ۴-۱-۳ مثال انگيزشي

تمرکز اصلی ما در روش پیشنهادی، بر روی پیمانه توليد داده آزمون در يک فازر قالب فایل است. شکل ۴-۲، مراحل توليد داده آزمون از طریق روش پیشنهادی را روی قالب فایل PDF، نشان می‌دهد. سه مرحله اصلی در این شکل وجود دارد که به ترتیب با شماره‌های يک تا سه، شماره‌گذاری شده‌اند. مطابق این مراحل، ابتدا پیکره‌ای از فایل‌های از پیش موجود و معتبر جمع آوری گردیده و به عنوان مجموعه داده انتخاب می‌شوند. در مرحله اول، یعنی پیش‌پردازش، هر يک از فایل‌های ورودی پیمایش شده، بخش‌های متنی آن جدا و به يكديگر الحق می‌شوند. همچنین بخش‌های دودویی داخل فایل، نیز به صورت مجزا، پس از جدا شدن، نگهداری می‌شوند. در فایل PDF، بخش‌هایی دودویی بین کلیدواژه‌های stream و endstream می‌آيند که در پیش‌پردازش آنها را با توکن دودویی stream جایگزين می‌کنیم. خروجی اين مرحله دو مجموعه است. يک مجموعه داده متنی و يک مجموعه داده دودویی. در مجموعه داده متنی، مكان قسمت‌های دودویی با توکن دودویی مشخص شده است.

در مرحله دوم، يک مدل زبانی عصبی برروی مجموعه داده حاصل از جداسازی بخش‌های متنی پیکره آموزش داده می‌شود. مانند هر وظیفه يادگیری ماشینی در این مرحله ابتدا مجموعه داده ورودی به سه مجموعه آموزش، آزمون و ارزیابی تقسیم می‌گردد. خروجی این مرحله يک مدل مولد است که قادر به تولید داده‌های جدید با پیروی از توزيع حاکم بر داده‌های مجموعه آموزش بوده و کاراکتر به کاراکتر داده‌های جدید را تولید می‌کند.

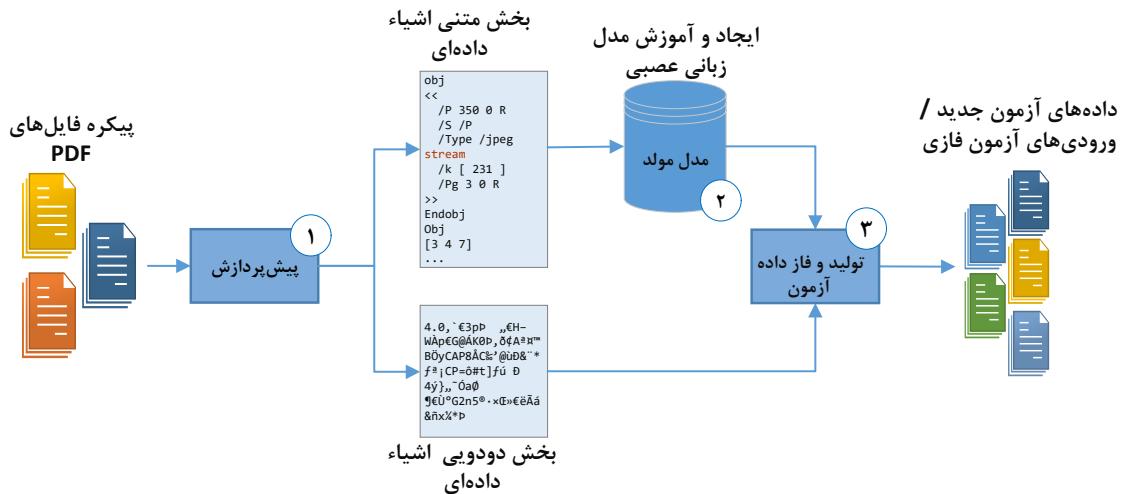
مرحله سوم با روش‌هایي اقدام به تولید داده‌های آزمون جدید، در اينجا فایل‌های PDF، می‌کند. در اين مرحله بخش‌های متنی فایل جدید از طریق مدل مولد، یعنی با روش مبنی بر تولید، ایجاد می‌گردد. عمل

بدشکل کردن داده‌های متنی نیز همزمان با تولید آنها صورت می‌گیرد. بدین صورت که برخی از کاراکترها بعد از تولید، بر اساس شرایطی که در ادامه فصل توضیح داده می‌شوند، با کاراکترهای دیگری جایگزین می‌شوند. در همین مرحله، بخش‌هایی دودویی نیز در صورت پیش‌بینی توسط مدل مولد، با یک روش مبتنی بر جابه‌جایی، تولید و در مکان مناسب خود قرار می‌گیرند. منظور از پیش‌بینی در اینجا، تولید توکن دودویی stream در توسط مدل مولد است. در تولید بخش دودویی یک داده آزمون از بخش‌های دودویی پیکره اولیه که در مرحله یک جداسازی شده است به عنوان دانه اولیه استفاده می‌کنیم. در واقع آنها را با یکی از عملگرهای جابه‌جایی که در فصل ۲ به آن اشاره شد به سادگی جابه‌جا کرده و یک داده آزمون دودویی جدید و فاز شده می‌سازیم.

مرحله سوم در مجموع تعدادی داده آزمون را با روشی ترکیبی، تولید می‌کند. این داده‌های آزمون سپس می‌توانند به عنوان داده‌های آزمون در فرایند آزمون فازی استفاده شوند. همچنین می‌توان از آن به عنوان دانه اولیه برای فازرهای قالب فایل مبتنی بر جابه‌جایی نیز استفاده کرد. برای ارزیابی روش پیشنهادی خود که در فصل بعد به آن خواهیم پرداخت، در انتهای این فصل یک فازر قالب فایل ساده را مطرح می‌کنیم. این فازر قادر است تا پس از تولید داده‌های آزمون به روش شرح داده شده آنها را به عنوان ورودی به SUT تزریق کرده و خطاهای احتمالی را در صورت وجود گزارش کند. مرحله پیش‌پردازش بسیار ساده بوده و نیاز به توضیح خاصی ندارد. این مرحله ممکن است برای قالب‌های فایل مختلف، کاملاً متفاوت باشد. مثلاً برای قالب‌های فایل متنی مثل XML و HTML پیش‌پردازش تنها الحاق کردن کلیه فایل‌های موجود در پیکره است. مراحل دوم و سوم، یعنی ایجاد و آموزش مدل مولد و تولید و فاز داده آزمون، به ترتیب در بخش ۴-۲ و بخش ۳-۴ توضیح داده شده‌اند. همچنین معماری فازر قالب فایل پیشنهادی در بخش ۴-۴ بیان گردیده است.

## ۲-۴ مدل

یک فایل را می‌توان نمونه تولید شده توسط زبانی که قالب آن فایل را بیان می‌کند دانست. با این فرض می‌توان یک مدل زبانی برای هر قالب فایلی ایجاد کرد. مدل زبانی عصبی عملکرد بهتری نسبت به مدهای زبانی سنتی دارد. با توجه به ذات مبتنی بر توالی بایت‌های یک فایل، وابستگی بایت فعلی به یک توالی از بایت‌های قبلی و نیز تجزیه ترتیبی (چپ به راست و بالا به پایین) آن توسط تجزیه‌گر SUT در بسیاری از موارد، در روش پیشنهادی خود، از RNN برای ایجاد مدل زبانی و یادگیری ساختار فایل استفاده می‌کنیم. در فصل ۲ دیدیم که RNN برای یادگیری وظایف مبتنی بر توالی ابداع شده است. به طور دقیق‌تر در هنگام آموزش از یک مدل RNN چند به یک، با معماری شکل ۶-۲ (پ) استفاده خواهیم کرد و در هنگام تولید داده جدید با فراخوانی همین مدل به صورت متوالی، مدلی از نوع شکل ۶-۶ (ت) خواهیم داشت.



شکل ۴-۲: مراحل یادگیری ساختار فایل، تولید و فاز داده آزمون برای فایل PDF.

هدف RNN در اینجا ایجاد یک مدل زبانی عصبی روی ساختار فایل با دیدن یک پیکره از نمونه‌های آموزشی است. مطابق آنچه گفتیم، مدل زبانی مدلی مولد است، یعنی پس از آموزش، می‌توان از آن برای تولید داده‌های آزمون جدید استفاده کرد، که در ادامه نحوه این کار را توضیح خواهیم داد. در این پایان‌نامه دو مدل با معماری مختلف بر مبنای RNN می‌سازیم. در هسته عصب‌های RNN هر دو مدل از سلول LSTM استفاده می‌کنیم که قابلیت یادگیری توالی‌های با طول زیاد را دارد. مدل اول LSTM یکسویه<sup>۱</sup> نام دارد که معماری آن مشابه شکل ۵-۲ است. مدل دوم LSTM دوسویه<sup>۲</sup> است.

LSTM دوسویه توالی ورودی را به دو صورت روبه‌جلو<sup>۳</sup> و روبه‌عقب<sup>۴</sup> می‌بیند. در واقع LSTM دوسویه از دو LSTM یکسویه تشکیل شده است. یکی از آنها توالی را از چپ به راست پردازش می‌کند و دیگری از راست به چپ. در نتیجه هر گذر جلو LSTM دو خروجی خواهد داشت. یک تابع ادغام<sup>۵</sup> برای ادغام خروجی هریک از LSTM‌های روبه‌جلو و روبه‌عقب به کار می‌رود. تابع ادغام می‌تواند هر تابعی که قادر به ترکیب دو بردار با طول‌های یکسان است، درنظر گرفته شود؛ از جمله: جمع، ضرب، الحق و غیره. ما از تابع ادغام جمع برای مدل پیشنهادی خود استفاده می‌کنیم که درایه‌های نظیر به نظیر هر یک از بردارهای خروجی را با یکدیگر جمع می‌زنند.

علاوه بر نوع مدل LSTM، برای LSTM یکسویه مدل‌هایی با تعداد عصب‌های متفاوت و نیز تعداد

<sup>1</sup>Unidirectional

<sup>2</sup>Bidirectional

<sup>3</sup>Forward

<sup>4</sup>Backward

<sup>5</sup>Merge Function

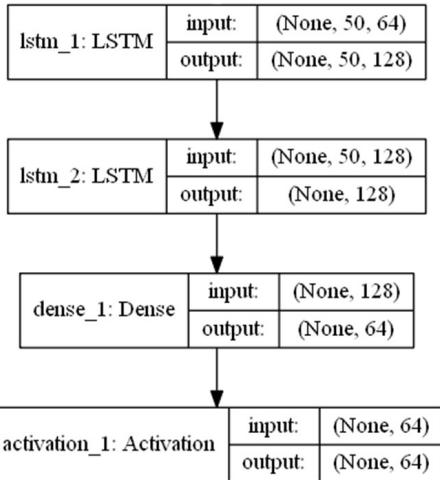
**جدول ۴-۱:** مدل‌های یادگیری ژرف طراحی شده، پارامترها و ابرپارامترهای آنها.

شماره (نام) مدل	نوع (معماری) مدل	تعداد لایه پنهان	تعداد عصب در هر لایه	تعداد پارامترها
۱۲۷۵۸۴	۱ Unidirectional LSTM (Many to One)	۱۲۸	۱	
۲۳۸۶۵۶	۲ Unidirectional LSTM (Many to One)	۱۲۸	۲	
۸۷۰۴۶۴	۲ Unidirectional LSTM (Many to One)	۲۵۶	۲	
۴۶۹۰۵۶	۴ Bidirectional LSTM (Many to One)	۱۲۸	۴	

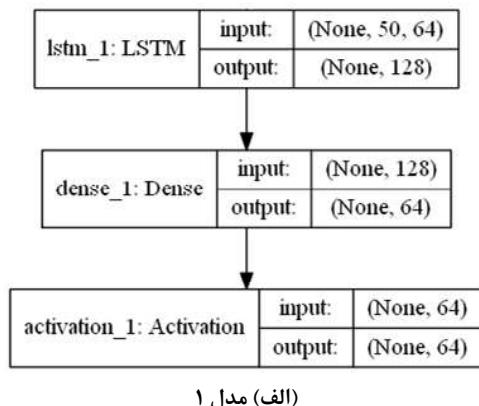
لایه‌های پنهان متفاوت را ساخته‌ایم. هدف از این کار مشاهده تأثیر مدل‌های با پیچیدگی مختلف در یادگیری ساختار فایل و تولید داده‌های آزمون و آزمایش این فرضیه است که آیا مدل‌های پیچیده‌تر، مدل‌های ساده‌تر را شکست می‌دهند؟ یعنی موفق به افزایش پوشش کد و یا شناسایی خطاها بیشتری می‌شوند یا خیر. جدول ۴-۱ مدل‌های طراحی شده برای استفاده در آزمایش‌های روش پیشنهادی و مشخصه‌های هریک را نشان می‌دهد. یکی از مهم‌ترین مشخصه‌ها تعداد پارامترهای هر مدل است. گراف محاسباتی هریک از مدل‌های جدول ۴-۱ در شکل‌های ۴-۳-۴ الف تا ۴-۳-۴ ت آمده است. اندازه ماتریس‌های ورودی و خروجی در گراف مرتبط با تعداد لایه‌های پنهان و نیز تعداد عصب‌ها در هر لایه است که در ادامه بیشتر توضیح داده خواهد شد. همچنین در ادامه پایان‌نامه، هریک از مدل‌های معرفی شده، با شماره ذکر شده برای آنها در جدول ۴-۱ شناخته می‌شوند.

**۴-۲-۱ آموزش مدل**

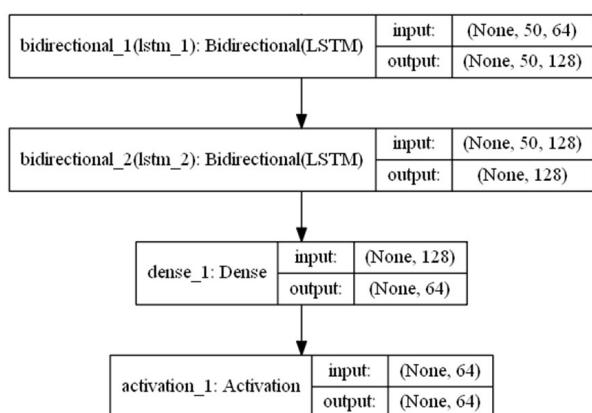
فرایند آموزش همه مدل‌های جدول ۴-۱ یکسان است. در واقع برای آموزش هر مدل تنها نیاز داریم که ورودی و خروجی شبکه عصبی ژرف را مشخص نماییم. پس از استخراج مجموعه داده اولیه که حاوی یک پیکره متواالی از کاراکترهای در بردارنده فایل است، آن را به مجموعه‌های آموزش، آزمون و ارزیابی تقسیم می‌کنیم. چون خروجی مدل مولد در هر بار اجرا یک توالی خواهد بود، طول توالی‌های مجموعه داده معیار مناسبی برای تقسیم‌بندی آن به مجموعه‌های سه‌گانه یاد شده است به نحوی که هر مجموعه توزیع یکسانی از فراوانی طول‌های مختلف باشد. از مجموعه‌های آموزش و ارزیابی در هنگام آموزش مدل و از مجموعه آزمون در هنگام تولید داده‌های جدید از روی مدل، استفاده خواهیم کرد. تفکیک مجموعه داده به سه مجموعه



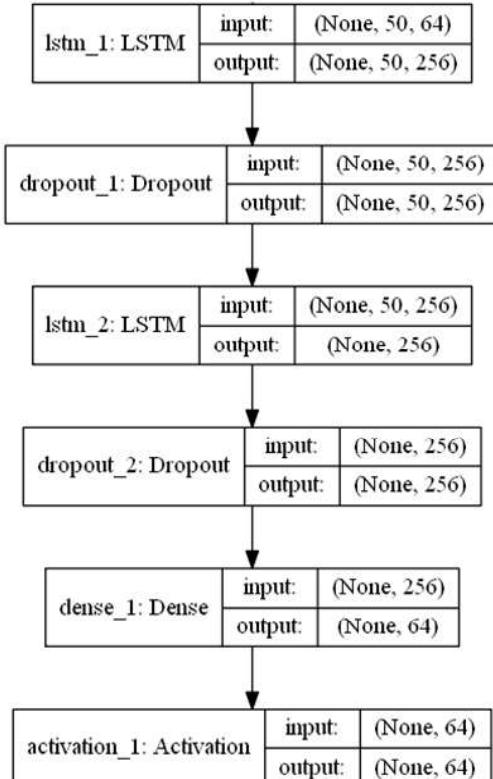
۲) مدل ۲



الف) مدل ۱



ت) مدل ۳



پ) مدل ۴

شکل ۳-۴: گراف محاسباتی مدل‌های عصبی ژرف جدول ۴-۱. اعداد داخل مستطیل‌ها اندازه ماتریس‌های ورودی و خروجی هر لایه را نشان می‌دهند. گراف‌ها با استفاده از توابع کتابخانه Keras [۵۰] تولید شده‌اند.

آموزش، ارزیابی و آزمون در وظایف یادگیری ماشینی یک امر بدیهی است، اما استفاده از آن برای آزمون فازی و تولید داده‌های آزمون جدید تا قبل از این، انجام نشده و چنان‌چه در ادامه خواهیم دید، ایده جدیدی است. در واقع ما از داده‌های مجموعه آزمون برای تولید داده‌های آزمون جدید استفاده خواهیم کرد.

هر مجموعه بدین ترتیب حاوی تعدادی فایل است و هر فایل یک توالی از کاراکترهای ASCII است. شروع و پایان هریک از این توالی‌ها پیش از فرایند آموزش، با توکن‌های شروع و پایان که نشانه‌های مجزایی (بیرون از مجموعه داده) هستند برچسب‌گذاری می‌شود. البته بسیاری از قالب‌های فایل چنین توکن‌هایی را به عنوان بخشی از ساختار خود دارند. برای مثال فایل‌های HTML با برچسب  $<html>$  شروع و با برچسب  $</html>$  خاتمه می‌یابند. فایل‌های XML و PHP نیز چنین هستند. در این حالت نیازی به افزودن توکن‌های شروع و پایان جدید نیست و می‌توان از همین توکن‌ها استفاده کرد. دلیل افزودن توکن‌های شروع و پایان در این مرحله، به نحوه ساخت مدل‌های زبانی عصبی باز می‌گردد. در واقع این مدل‌ها با دیدن توکن آغازین پیش‌بینی را شروع و با تولید توکن پایانی، پیش‌بینی را خاتمه می‌دهند.

به منظور آموزش هر مدل چند به یک و ایجاد یک مدل زبانی عصبی، ابتدا با الحاق محتوای فایل‌های مجموعه آموزش، یک توالی بزرگ از کاراکترها به شکل  $<(n)^s, (n)^s, \dots, (n)^s> = S$  ایجاد می‌کنیم. سپس این توالی را به تعدادی توالی ورودی کوچکتر با طول ثابت  $d$  می‌شکنیم طوری که  $i$ -امین توالی ورودی برابر است با:

$$x_i = S[i * j : (i * j) + d] \quad (1-4)$$

در رابطه ۱-۴،  $s[l, u]$  برشی از توالی  $s$  بین شاخص‌های  $l$  و  $u$  بوده و  $j$  میزان پرش به جلو در انتخاب توالی ورودی بعدی از توالی اصلی را نشان می‌دهد. ابرپارامتر  $d$  در اینجا، در واقع همان بردار زمینه در مدل زبانی است. انتخاب  $d$  به پارامترهایی مانند فاصله وابستگی‌های بین بایت‌های مختلف در یک فایل و طول فایل‌های مجموعه داده، بستگی دارد. وقتی از LSTM به عنوان بلوک‌های سازنده RNN در ساخت NLM استفاده شود، می‌توان وابستگی‌های طولانی‌تر را به خوبی یاد گرفت. در عمل و برای وظایف مدل زبانی، این مقدار را بین ۴۰ تا ۱۰۰ نشانه در نظر می‌گیرند.

گفتیم شبکه عصبی در حالت بانتظارت آموزش می‌بیند؛ یعنی، برای هر ورودی یک برچسب خروجی نیاز است. در اینجا کاراکتر بعدی، که هدف پیش‌بینی آن است، را به عنوان برچسب خروجی برای هر توالی ورودی نسبت می‌دهیم. بنابراین برچسب خروجی برای  $i$ -امین توالی ورودی برابر خواهد بود با:

$$Y_{x_i} = S[(i * j) + d + 1] \quad (2-4)$$

پس از تولید همه نمونه‌های آموزشی و برچسب‌های متناظر آنها، مدل به صورت انتهایه‌انتها آموزش داده می‌شود؛ مشابه روش یادگیری و فاز [۱۱]. . یعنی، داده‌های خام مجموعه آموزش، بدون استخراج هیچ‌گونه ویژگی خاصی به شبکه داده می‌شوند و در زمان استفاده نیز، شبکه داده‌های نهایی را بدون نیاز به هیچ‌گونه پس‌پردازشی تولید می‌کند. در این حالت مدل قادر پیش‌بینی مقدار احتمال شرطی  $p(x^{(i+d+1)} | x^{(i+d)}, \dots, x^{(i)})$  نشان داده شده است. همچنین شکل ۴-۴، یک فایل HTML و سه‌توالی آموزشی اول تولید شده برای آن توسط الگوریتم ۱-۴ را به همراه موقعیت قرارگیری آنها در شبکه نشان می‌دهد. توالی ورودی کلی در این مثال، همه کاراکترهای فایل HTML و پارامترهای  $d$  و  $j$  به ترتیب برابر ۳ و ۱ قرار داده شده‌اند. مقادیر در نظر گرفته شده کاملاً فرضی هستند.

شبکه عصبی با مقادیر عددی کار می‌کند. بنابراین با ایستی یک نمایش عددی برای هر کاراکتر در مجموعه آموزش در نظر گرفت. یک روش مرسوم در مدل‌های مبتنی بر کاراکتر استفاده از بردارسازی One-hot یا ۱ از  $k$  است. در این روش یک بردار به طول اندازه مجموعه واژگان برای هر کاراکتر درنظر می‌گیرند که تنها یکی از شاخص‌های آن برابر ۱ است. بدین ترتیب برای نمایش  $V$  کاراکتر،  $V$  بردار متفاوت خواهیم داشت. در اینجا نیز از همین روش استفاده می‌کنیم.

خروجی شبکه نیز که به‌شکل یک کاراکتر تعریف شده است در واقع یک بردار One-hot است. در فرایند آموزش تنها یکی از شاخص‌های این بردار یک است. اما در مرحله آزمون بردار تولیدی حاوی مقادیر حقیقی مختلفی در بازه  $(-\infty, +\infty)$  برای هر شاخص است. با اعمالتابع ییشینه هموار (رجوع شود به رابطه ۱۲-۲)، در خروجی این لایه، می‌توان خروجی را به یک توزیع احتمالی معتبر تبدیل کرد.

#### الگوریتم ۱-۴ ProduceTrainingSamples

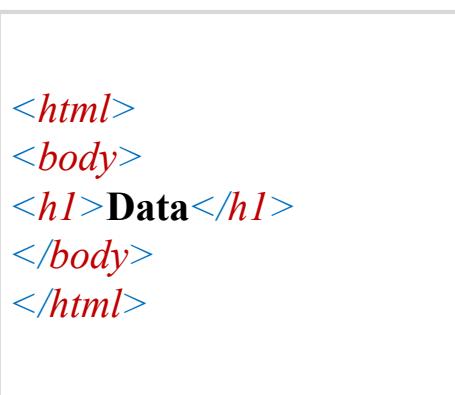
**Input:** Total sequence  $S$ , Input sequences lenght  $d$ , Jump step  $j$

**Output:** Input sequences  $x$ , Output labels  $Y$

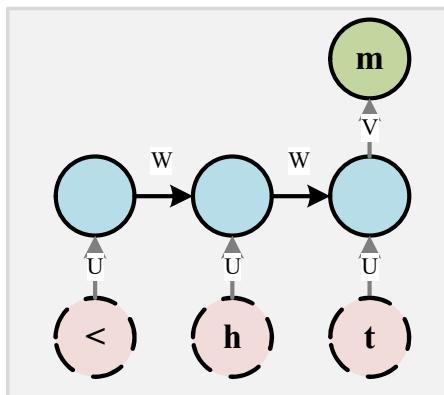
```

1  $x \leftarrow \text{List}()$                                      /* Make an empty list */
2  $Y \leftarrow \text{List}()$                                      /* Make an empty list */
3 for  $i \leftarrow 1$  to  $\text{Len}(S) - d$  do
4    $x[i] \leftarrow S[i * j : (i * j) + d]$ 
5    $Y[i] \leftarrow S[(i * j) + d + 1]$ 
6 end
7 Return  $x, Y$ 

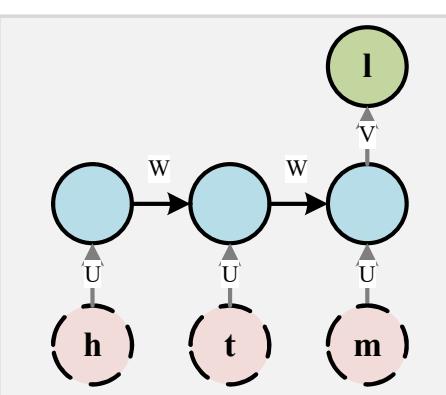
```



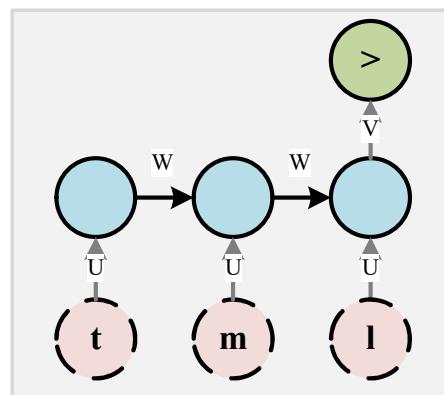
(الف) یک فایل HTML



(ب) توالی آموزشی اول



(پ) توالی آموزشی دوم



(ت) توالی آموزشی سوم

**شکل ۴-۴:** مثالی از نحوه آموزش مدل‌های چند به یک روی یک فایل HTML ساده. تنها سه توالی آموزشی اول در این شکل نشان داده شده است. اما الگوریتم ۴-۱ کل فایل را به توالی‌های آموزشی تقسیم می‌کند.

## ۲-۲-۴ تولید داده‌های جدید

پس از اتمام فرایند آموزش، مدل برای تولید داده‌های جدید مورد پرس‌وجو قرار می‌گیرد. در این مرحله ابتدا یک پیشوند ثابت از کاراکترهای شروع کننده یک توالی از فایل‌های مجموعه آزمون به طول  $d$ ، که توکن شروع در ابتدای آن است، به شبکه خورانیده می‌شود<sup>۱</sup>. شبکه توزیع احتمالی کاراکتر  $1 + d$  را به عنوان خروجی تولید می‌کند که شامل یک بردار One-hot از مقادیر احتمال‌های تمام کاراکترهای دیده شده هنگام آموزش است. سپس یک کاراکتر از این توزیع احتمالی، انتخاب شده و به پیشوند ورودی الحاق می‌گردد. در این حالت طول پیشوند  $1 + d$  است. در مرحله بعد سمت چپ‌ترین کاراکتر پیشوند حذف شده و  $d$  کاراکتر باقی مانده دومرتبه به شبکه داده می‌شود تا توزیع احتمالی کاراکتر  $2 + d$  حاصل شود. این روند تا زمان تولید توکن پایانی که خاتمه یافتن یک توالی را پیش‌بینی می‌کند، ادامه خواهد داشت. در [۱۱] سه راهبرد تولید داده جدید از توزیع احتمالی پیش‌بینی شده توسط مدل معرفی شده که در بخش ۲-۳-۲ آنها را دیدیم.

در اینجا ما از راهبرد نمونه‌برداری که روشی مرسوم برای تولید داده از یک توزیع آماری است استفاده می‌کنیم. یعنی هر بار از توزیع احتمالی پیش‌بینی شده به عنوان یک توزیع توزیع چندجمله‌ای<sup>۲</sup> نمونه‌برداری numpy نماییم. البته راهبردهای جدیدی را نیز معرفی خواهیم کرد. در زبان پایتون و با استفاده از بسته numpy می‌توان به صورت زیر از یک توزیع چندجمله‌ای نمونه‌برداری کرد:

---

```

1 import numpy as np
2 sample = np.random.multinomial(n, prob_vals, size=None)

```

---

برنامه ۴-۱: نمونه‌برداری از توزیع چندجمله‌ای در پایتون.

### پارامتر تنوع

در [۱۱] انتخاب حریصانه به دلیل وجود پیشوند ثابت راهبرد ناکارمدی معرفی شده است. در روش پیشنهادی ما، مدل پیشوندهای متغیری را می‌پذیرد. بنابراین می‌توانیم از راهبرد حریصانه هم استفاده نماییم. با این حال تعداد داده‌های تولیدی در این راهبرد محدود به تعداد نمونه‌های مجموعه آزمون است که البته تعداد کمی نخواهند بود.

<sup>1</sup> دقت شود که ویژگی فایل‌های مجموعه آزمون آن است که قبلاً در فرایند آموزش توسط مدل مشاهده نشده‌اند و این امر تأثیر مستقیمی در ارزیابی خوب بودن مدل و تنوع داده‌های تولید شده توسط مدل دارد. بدون وجود مجموعه آزمون نمی‌توانیم معیارهایی مانند سرگشتشگی را به درستی حساب کنیم. به همین سبب جداسازی مجموعه آزمون را تأکید می‌کنیم.

<sup>2</sup>Multinomial Distribution

در مقابل راهبرد حریصانه، نمونه‌برداری باعث ایجاد اشیای متنوع می‌شود که لزوماً خوش‌شکل نیستند. به همین دلیل پارامتری به نام تنوع<sup>۱</sup> را با هدف کنترل داشتن برروی تنوع داده‌های تولیدی مطرح می‌کنیم. تنوع  $D$  به صورت یک عدد حقیقی در بازه  $(0, +\infty)$  تعریف می‌گردد. در هنگام آزمون مدل، مقادیر حقیقی پیش‌بینی شده توسط مدل، ابتدا بر  $D$  تقسیم می‌شوند و سپستابع بیشینه همواره به آن اعمال می‌شود تا بردار خروجی حاصل گردد. در نتیجه هرچه قدر مقدار  $D$  کوچک‌تر (نزدیک به صفر) انتخاب شود، نمونه‌برداری به راهبرد حریصانه نزدیک شده، داده‌های خوش‌شکل‌تری تولید می‌گردد اما از تنوع آن کاسته می‌شود. بالعکس هرچه قدر مقدار  $D$  بزرگ‌تر از یک انتخاب شود، اختلاف بین مقادیر پیش‌بینی شده کم، تنوع داده‌های تولید شده زیاد و متقابلاً احتمال بدشکل بودن آنها افزایش می‌یابد.

#### n-فهرست بهتر

یک راهبرد دیگر تولید داده جدید، استفاده از n-فهرست بهتر است. در این راهبرد در هر بار پیش‌بینی مدل پس از اعمال تابع بیشینه هموار، تعداد  $n$  احتمال بالای بردار خروجی انتخاب و نگهداری می‌شود. سپس با هریک از این احتمال‌ها پیش‌بینی مرحله بعد انجام می‌شود. برای انتخاب n-فهرست بهتر بین دو پیش‌بینی مقادیر احتمال‌های هر دو فهرست در یکدیگر ضرب و  $n$  حاصل ضرب بالاتر انتخاب می‌شود. پس از  $B$  مرحله،  $n$  پیشوند  $B$  تایی خواهیم داشت که پیشوند با بالاترین احتمال پیشوند خروجی خواهد بود. این راهبرد البته زمان و حافظه مصرفی بالایی نیاز دارد. لذا مقادیر  $n$  و  $B$  بایستی کوچک انتخاب شوند. در ترجمه ماشینی معمولاً از این راهبرد نیز استفاده می‌شود. ما در این پایان‌نامه و در آزمایش‌های خود، راهبرد بالا را ارزیابی نکردیم، اما می‌توان از آن در عمل استفاده کرد.

## ۳-۴ الگوریتم‌های فاز عصبی

این بخش را می‌توان هسته روش پیشنهادی و نوآوری‌های ارایه شده در این پایان‌نامه دانست. در اینجا توضیح می‌دهیم که داده‌های آزمون چگونه با روشی خودکار و ترکیبی تولید و فاز می‌شوند. هنگامی که داده‌های آزمون را با استفاده از مدل‌های بخش ۴-۲ و راهبردهای معرفی شده در بخش ۴-۲ تولید می‌کنیم یک تنوع ذاتی در داده‌ها وجود خواهد داشت و در هر صورت داده‌های آزمون جدید محسوب می‌شوند. اما اهداف یادگیری قالب فایل و آزمون فازی قالب فایل در تضاد با یکدیگر هستند. هدف الگوریتم یادگیری ایجاد داده‌های خوش‌شکل است که بتواند از تجزیه‌گر اولیه عبور و مسیرهای عمیق را اجرا کند و هدف آزمون فازی بدشکل

<sup>۱</sup>Diversity

## ۴-۳. الگوریتم‌های فاز عصبی

کردن ورودی است به نحوی که تجزیه‌گر و دیگر قسمت‌های کد اجرا شده را به حالت خرابی ببرد. در این بخش، ما الگوریتم‌های فاز عصبی را با هدف ایجاد یک مصالحه بین دو هدف قبلی و تولید نهایی داده آزمون برای یک فازر قالب فایل، معرفی می‌کنیم.

فاز کردن داده آزمون همزمان با تولید آن از روی مدل صورت می‌گیرد و مرحله مجزایی نیست. الگوریتم‌های ۴-۲ و ۴-۳ نحوه تولید داده آزمون که در بخش ۲-۴ بحث شد و فاز همزمان آن را نشان می‌دهند. هر دو الگوریتم به عنوان ورودی مدل یادگیری شده  $M$ ، پیشوند شروع توالی  $P$  (حاوی توکن شروع)، تنوع نمونه‌برداری  $D$ ، نخ فاز  $FR$ ، توکن پایان  $ET$  و توکن دودویی  $BT$  را پذیرفته و داده آزمون  $TD$  را به عنوان خروجی باز می‌گردانند.

حلقه اصلی هر یک از الگوریتم‌ها، تا زمانی که  $ET$  تولید نشده باشد ادامه می‌یابد. چنان‌چه طول توالی تولید شده از یک حد آستانه  $MaxLen$  بیشتر شد ولی  $ET$  کماکان تولید نشده باشد، آنگاه  $ET$  به صورت خودکار به انتهای  $TD$  اضافه می‌شود تا الگوریتم از حلقه تکرار بیرون آید. بدین ترتیب مطمئن می‌شویم که الگوریتم همراه خاتمه خواهد یافت. سپس در صورتی که مدل حضور یک داده دودویی در  $TD$  را پیش‌بینی کرده باشد (یعنی  $BT$  در توالی  $TD$  پیدا شود)، ابتدا یک بخش دودویی جایگزین  $BT$  شده و سپس این بخش با روش‌های جابه‌جایی تصادفی یا هر روش دلخواه دیگر به صورت مجزا فاز می‌شود و در نهایت داده آزمون نهایی توسط الگوریتم بازگردانیده می‌شود. ما در عمل برای خط ۱۹ هر دو الگوریتم، عملگر معکوس کردن یک بایت را در نظر گرفتیم. بدین ترتیب که درصد ثابتی از بایت‌های دودویی را در هر بخش دودویی انتخاب شده، به صورت تصادفی انتخاب نموده و مقادیر بیت‌های هر بایت انتخابی را معکوس می‌کنیم.

## ۴-۳-۱ فاز عصبی داده

تفاوت اصلی در الگوریتم‌های فاز عصبی داده و فاز عصبی فراداده در هنگام اعمال عملیات فاز است. در بخش ۲-۱-۴ گفتیم مدل به داده‌ها احتمال کمتری نسبت می‌دهد. بنابراین در الگوریتم ۴-۲، هرگاه احتمال کاراکتر پیش‌بینی شده از یک حد آستانه  $\alpha$  کمتر باشد، متوجه می‌شویم که کاراکتر جزو بخش داده فایل بوده که در این حالت آن را با کاراکتری که کمترین احتمال را دارد جایگزین می‌کنیم. با این اميد که کم احتمال‌ترین کاراکتر، ناخواسته‌ترین کاراکتر نیز بوده و ممکن است منجر به خرابی شود.

خط ۷ الگوریتم ۴-۲ بیان می‌دارد که برای فاز بایستی شرایط دیگری علاوه بر شرط فوق برقرار باشد. از جمله عدد تصادفی  $p_{fuzz}$  که لازم است از  $FR$  یعنی نخ فاز کمتر شود. هرچه قدر مقدار  $FR$  بزرگ‌تر انتخاب شود شرط  $FR < p_{fuzz}$  برای اعداد بیشتری تحقق می‌یابد. بدین ترتیب آزمون‌گر می‌تواند روی درصد کاراکترهای فاز شده کنترل داشته باشد. همچنین می‌خواهیم که مجموعه کاراکترهای توالی‌های  $ET$  و

$BT$  فاز نشود؛ زیرا از آنها در قسمت‌های بعد استفاده می‌کنیم. لذا دو شرط دیگر به شرط‌های خط ۷ الگوریتم ۴-۲ اضافه می‌شوند. بنابراین فاز کاراکتر پیش‌بینی شده تنها در حالتی که ترکیب عطفی هر ۴ شرط درست باشد، صورت می‌گیرد.

تغییر یک بایت یا یک کاراکتر از بخش‌های داده یک فایل، معمولاً بدشکل بودن خاصی را ارائه نمی‌دهد. به طور شهودی تصور کنید که به جای عدد ۹۲۵ عدد ۱۲۵ قرار داده شود یا هر البته هر کاراکتر دیگری در موقعیت رقم ۱. در بد شکل کردن داده هدف ما جایگزین کردن یک مقدار مرزی به جای داده موجود است. مثلاً برای مورد گفته شده جایگذاری عددی به فرم ۹۹۹...۹ خوب به نظر می‌رسد. در واقع در فاز عصبی داده، به دنبال روشی برای انتشار فاز به کاراکترهای بعدی نیز هستیم. وقتی یک کاراکتر فاز شد، دو راهکار برای تولید کاراکترهای بعدی پیش‌رو داریم. راهکار اول تولید کاراکترهای بعدی، مستقل از کاراکتر فاز شده است؛ یعنی اینکه مدل در تولید کاراکتر  $t$  ام هیچ آگاهی از فاز شدن کاراکتر مرحله  $1-t$  ام خود نداشته باشد. برای این منظور کافی است تا کاراکتر اولیه تولید شده توسط مدل، یعنی کاراکتر قبل از فاز و جایگزین شدن، را به پیشوند شروع توالي  $P$  اضافه کنیم. بدین ترتیب مدل مستقل از اینکه کاراکتر فاز شده است یا نه به تولید داده‌های مبتنی بر گرامر (داده‌های خوش‌شکل) ادامه می‌دهد. راهکار دوم تولید کاراکترهای بعدی، وابسته به کاراکتر فاز شده است؛ یعنی اینکه اثر کاراکتر فاز شده به کاراکترهای تولید شده در مراحل بعد سرایت کند. برای این منظور کاراکتر فاز شده را به پیشوند شروع توالي  $P$  اضافه کرده و مدل را به سمت بد شکل کردن کاراکترهای بعدی متماطل می‌کنیم. این راهکار در الگوریتم فاز عصبی داده به دلیل شرح داده شده استفاده گردیده است. خط ۱۰ الگوریتم ۴-۲، کاراکتر فاز شده یعنی کاراکتر  $c$  را به پیشوند  $P$  اضافه می‌کند. در همین خط برای ثابت ماندن طول پیشوند که در واقع ورودی مدل است، کاراکتر اول یعنی قدیمی‌ترین کاراکتر را از پیشوند حذف می‌کنیم. در کل این خط عمل انتشار فاز به جلو را پیاده‌سازی می‌کند.

## ۴-۳-۴ فاز عصبی فراداده

در الگوریتم فاز عصبی فراداده (الگوریتم ۴-۳)، می‌خواهیم کاراکترهای فراداده را تغییر دهیم. برای این منظور بررسی می‌کنیم که احتمال کاراکتر پیش‌بینی شده توسط مدل یعنی ( $c$ )  $p$  بیشتر از حد آستانه  $\beta$  باشد، در این صورت کاراکتر با کمترین احتمال را جایگزین می‌کنیم. در اینجا اما به دنبال تغییر تنها یکی از کاراکترها در هر توکن فراداده هستیم. لذا در خط ۱۱ الگوریتم ۴-۳ تغییری در پیشوند بعدی مدل ایجاد نمی‌کنیم تا مدل به همان روند تولید داده خوش‌شکل خود ادامه دهد بدون اینکه متوجه فاز شدن کاراکتر پیش‌بینی کرده گام زمانی قبلی خود باشد. در واقع در فاز عصبی فرا داده، انتشار فاز به جلو را انجام نمی‌دهیم. ایده ما برای این کار این است که در اغلب مواقع یک تغییر کوچک، مثلاً تغییر یک بایت در قسمتی که قالب فایل را بیان

می‌کند سبب گمراه شدن تجزیه‌گر می‌شود. این عمل درست در خلاف مورد داده است که یک بایت معمولاً تغییری ایجاد نمی‌کند و این مقادیر مرزی (بیش از حد کوچک، بیش از حد بزرگ، خالی یا صفر) هستند که مسبب خرابی می‌شوند. مقادیر داده معمولاً به صورت یک توالی از بایت‌ها ادامه می‌یابند.

در الگوریتم ۴-۳ پارامتر نرخ فاز  $FR$  همچنان وجود دارد، اما دو شرطی که بررسی کننده وجود توکن‌های  $BT$  و  $ET$  هستند را حذف کردیم تا علاوه بر آسان‌تر کردن شرط فاز بتوانیم آنها را نیز به عنوان بخشی از قالب فایل، فاز کنیم؛ زیرا، تعداد خطاهایی که در تجزیه‌گر رخ می‌دهند، معمولاً بیشتر هستند. در نهایت خطوط سایه زده شده در الگوریتم ۴-۳ (خطوط ۷، ۸ و ۱۱) محل‌های تفاوت این الگوریتم با الگوریتم فاز عصبی داده (الگوریتم ۲-۴) را نشان می‌دهد.

حد آستانه  $MexLen$  که حداقل طول داده آزمون تولیدی را در الگوریتم‌های ۲-۴ و ۴-۳ مشخص می‌کند نیز به صورت یک عدد صحیح تصادفی در بازه  $(a, b)$  مشخص می‌شود. مقادیر این بازه می‌تواند براساس میانگین طول فایل‌های مجموعه آزمون، انتخاب شود. بدین صورت که کران‌های آن برابر اختلاف واریانس طول از میانگین باشند. مقادیر ابرپارامترهای موجود در هر دو الگوریتم باقیستی توسط آزمون‌گر مطابق قالب فایل مورد آزمون و نکته‌های بیان شده، تنظیم گردد. در فصل ۵ که ارزیابی روش پیشنهادی را مطرح می‌کنیم، مقادیر در نظر گرفته شده را برای قالب فایل PDF بیان می‌کنیم.

## ۴-۴ پیاده‌سازی

جزئیات پیاده‌سازی روش پیشنهادی در پیوست ب آمده است. برای پیاده‌سازی مدل‌های یادگیری ژرف از کتابخانه سطح بالای یادگیری ژرف Keras [۵۰] که به زبان پایتون نوشته شده است استفاده کردیم. این کتابخانه مجموعه‌ای مفید از توابع و API‌ها برای ساخت انواع مختلف شبکه‌های عصبی را در اختیار قرار می‌دهد؛ اما، برای اجرا نیاز به یک چارچوب سطح پایین دارد که ما Tensorflow [۴۷] را بدین منظور انتخاب کردیم. Tensorflow همچنین یک ابزار به نام Tensorboard دارد که از آن می‌توان برای مصورسازی گراف‌های محاسباتی و نیز مشاهده نمودارهای خطأ و دقت در فرایندهای آموزش و آزمون استفاده کرد. برای آموزش مدل‌ها ما از تابع خطای CE (رابطه ۶-۲ در بخش ۳-۲) و بهینه‌ساز Adam [۵۱] با نرخ‌های یادگیری  $1 \times 10^{-3}$  و  $1 \times 10^{-4}$  استفاده کردیم.

هدف این پایان‌نامه همان‌طور که پیش از این گفته شد، ارائه یک روش تولید خودکار داده آزمون است که در بخش‌های گذشته این فصل بحث شد. اما تولید داده آزمون به تهایی کافی نیست و برای ارزیابی لازم است تا یک فاזר قالب فایل با همه پیمانه‌های شکل ۲-۲ در اختیار داشته باشیم. برای این منظور نیاز به دو پیمانه تزریق کننده مورد آزمون و پایش SUT نیز داریم. اغلب نرم‌افزارهایی که یک فایل را به عنوان ورودی

## الگوریتم ۲-۴ DataNeuralFuzz

**Input:** Learnt model  $M$ , Sequence prefix  $P$ , Diversity  $D$ , Fuzzing rate  $FR$ , End token  $ET$ , Binary token  $BT$

**Output:** Test data  $TD$

```

1  $TD \leftarrow P$ 
2  $MaxLen \leftarrow \text{RandInt}(a, b)$ 
3 while not EndsWith( $TD, ET$ ) do
4    $predicts \leftarrow \text{Predict}(M(P))$ 
5    $c, p(c) \leftarrow \text{Sample}(predicts, D)$       /* Sample c from the learnt model */
6    $p\_fuzz \leftarrow \text{Random}(0, 1)$            /* Decide whether to fuzz */
7   if  $p\_fuzz < FR \wedge p(c) < \alpha \wedge c \notin \text{Chars}(BT) \wedge c \notin \text{Chars}(ET)$  then
8      $c \leftarrow \text{argmin}_{c'}\{p(c') \in predicts\}$       /* Fuzz c by c' where c' is the
       lowest likelihood */
9   end
10   $TD \leftarrow TD + c$ 
11   $P \leftarrow P[1:] + c$  /* Propagate fuzz to prefix and next generated data */
12  if  $\text{Len}(TD) > MaxLen$  then
13     $TD \leftarrow TD + ET$ 
14    Break
15  end
16 end
17 if  $BT \in TD$  then
18    $TD \leftarrow \text{AddBinaryPart}(TD)$ 
19    $TD \leftarrow \text{MutateBinaryPart}(TD)$ 
20 end
21 Return  $TD$ 

```

## الگوریتم ۳-۴ MetadataNeuralFuzz

**Input:** Learnt model  $M$ , Sequence prefix  $P$ , Diversity  $D$ , Fuzzing rate  $FR$ , End token  $ET$ , Binary token  $BT$

**Output:** Test data  $TD$

```

1  $TD \leftarrow P$ 
2  $MaxLen \leftarrow \text{RandInt}(a, b)$ 
3 while not EndsWith( $TD, ET$ ) do
4    $predicts \leftarrow \text{Predict}(M(P))$ 
5    $c, p(c) \leftarrow \text{Sample}(predicts, D)$       /* Sample  $c$  from the learnt model */
6    $p\_fuzz \leftarrow \text{Random}(0, 1)$            /* Decide whether to fuzz */
7   if  $p\_fuzz < FR \wedge p(c) > \beta$  then
8      $c' \leftarrow \text{argmin}_{c''} \{p(c'') \in predicts\}$     /* Fuzz  $c$  by  $c'$  where  $c'$  is the
       lowest likelihood */
9   end
10   $TD \leftarrow TD + c'$ 
11   $P \leftarrow P[1:] + c$            /* Don't propagate fuzz to prefix */
12  if  $\text{Len}(TD) > MaxLen$  then
13     $TD \leftarrow TD + ET$ 
14    Break
15  end
16 end
17 if  $BT \in TD$  then
18    $TD \leftarrow \text{AddBinaryPart}(TD)$ 
19    $TD \leftarrow \text{MutateBinaryPart}(TD)$ 
20 end
21 Return  $TD$ 

```

می‌پذیرند یک واسط خط فرمان نیز دارند که می‌توان از این طریق یک فایل را به آنها داد. در غیر اینصورت نیز می‌توان یک برنامه کوچک برای تزریق ورودی نوشت. به‌طور کلی در آزمون فازی قالب فایل تزریق ورودی ساده است. برای پایش نیز ابزارهای مستقلی وجود دارد که می‌توان از آنها استفاده کرد. ما آزمون فازی را تحت سیستم عامل ویندوز انجام دادیم<sup>۱</sup> و ابزار Application Verifier<sup>۲</sup> [۳۶] که در بخش ۲-۲ معرفی شد را برای پایش انتخاب کردیم که مایکروسافت استفاده از آن را در SDL توصیه کرده است.

فایل اجرایی SUT را می‌گیرد و در هر بار اجرای SUT یک فایل حاوی وقایع را با یک شماره ترتیبی ثبت می‌کند. در صورتی که برنامه دچار خطای زمان‌اجرا (شامل یکی از انواع خطاهای فساد حافظه، دسترسی غیر مجاز وغیره) شود، نوع خطأ در این فایل ثبت و ضبط می‌گردد. سپس می‌توان برنامه را به‌همراه داده آزمون مسبب خطأ (که از روی شماره ترتیبی فایل وقایع قابل تشخیص است) در یک محیط اشکال‌زدا مانند WinDbg مجدداً اجرا و ضمن مشاهده خطأ، محل دقیق آن را آشکار ساخت.

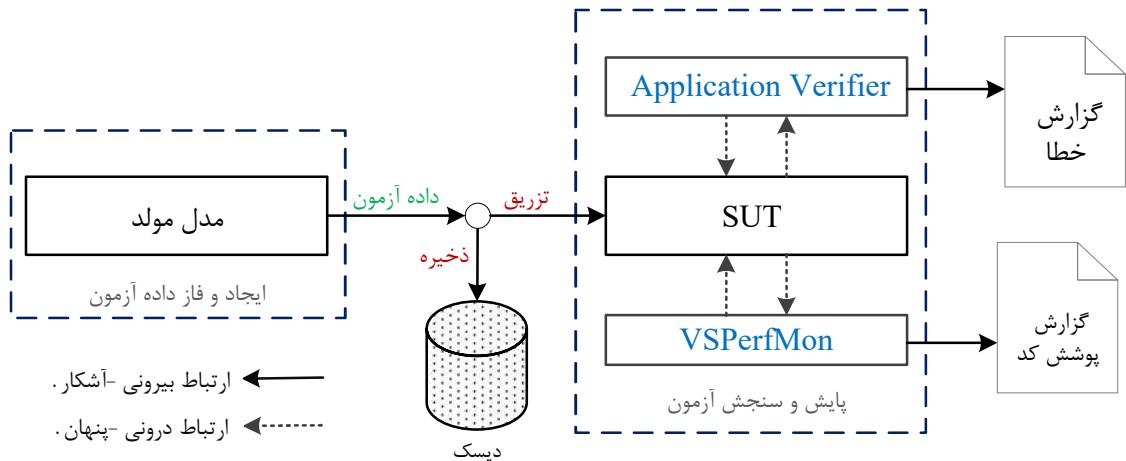
از سازوکاری که در بالا توضیح داده شد برای آزمون فازی نرمافزار MuPDF که یک نرمافزار پر استفاده است و قالب پیچیده PDF را به عنوان ورودی می‌پذیرد، در آزمایش‌ها استفاده کرده‌ایم. ما همچنین پوشش کد این نرمافزار را با استفاده ابزار VSPerfMon اندازه‌گیری کردیم تا عملکرد تولید کننده داده آزمون را سنجیم.

شکل ۴-۵ یک طرح‌واره کلی از معما ری (مؤلفه‌ها و ارتباط بین آنها) فاز پیشنهادی ما در این پایان‌نامه را نشان می‌دهد. با تعویض پیمانه پایش، این فاز در سیستم‌های عامل دیگر نیز قابل اجرا خواهد بود. این فاز را می‌توان در گروه فازرهای جعبه خاکستری، مبتنی بر روش ترکیبی تولید داده آزمون و بدون حلقه بازخورد به شمار آورد. VSPerfMon برای اندازه‌گیری پوشش کد در حالت جعبه‌سفید استفاده می‌شود که در صورت عدم وجود کد منبع SUT باقیستی آن را با یکی از ابزارهای سنجش پوشش کد بازیزی در بخش ۲-۱-۳ جایگزین کرد.

## ۴-۵ خلاصه

مدلهای زبانی عصبی در سطح کاراکتر قابلیت یادگیری یک توزیع آماری روی یک توالی از کاراکترها را دارند. در این فصل ما ساختار یک فایل را به صورت یک توالی از کاراکترها (بایت‌ها) مدل کرده و با ارائه چندین مدل زبانی عصبی و چندین راهبرد تولید داده‌های جدید از این مدل‌ها، یک روش تولید خودکار داده آزمون ارائه دادیم که قابل استفاده در فازرهای قالب فایل مختلف است. ما همچنین دو الگوریتم با عنوان‌های فاز داده عصبی و فاز فراداده عصبی ارائه کردیم. هر کدام از این الگوریتم‌ها بدشکل‌سازی داده آزمون ورودی

<sup>۱</sup> هرچند که روش پیشنهادی کاملاً مستقل از سیستم عامل است. برنامه تولید خودکار داده آزمون به زبان پایتون نوشته شده و روی سکوهای مختلف قابل اجرا است.



**شکل ۴-۵:** معماری فازر قالب فایل پیشنهادی. فازر به صورت کاملاً پیمانه‌ای پیاده‌سازی شده است. پیکان‌ها ارتباطات بین مؤلفه‌ها را نشان می‌دهند. ارتباط بیرونی، سازوکار ارتباطی است که توسط ما پیاده‌سازی شده یا قابل تغییر است و ارتباط درونی سازوکار ارتباطی است که توسط پیمانه‌های شخص ثالث استفاده می‌شود.

را با هدف خاصی انجام می‌دهند: اولی با هدف جابه‌جایی داده‌های یک فایل به منظور ایجاد خطأ در هنگام پرداخت آنها و دومی با هدف جابه‌جایی فراداده‌های یک فایل برای اینجا خطأ در تجزیه‌گر اولیه ساختار فایل تحت فاز.

در بخش پایانی فصل نیز یک فازر قالب فایل را طراحی و پیاده‌سازی کردیم که از آن می‌توان برای آزمون فازی برنامه‌هایی با ورودی فایل استفاده کرد. خروجی این فازر علاوه بر گزارش خطاهای یافته شده حین فرایند آزمون، گزارش میزان پوشش کد و ذخیره کلیه داده‌های آزمون تولید شده نیز است. معماری فازر پیشنهادی کاملاً پیمانه‌ای بوده و با تعویض پیمانه پایش، امکان استفاده از آن در سیستم‌های عاملی به‌جز ویندوز فراهم می‌شود. فازر معرفی شده یک فازر جعبه خاکستری و مجهز به یک روش تولید داده آزمون ترکیبی است. در فصل ۵، به ارزیابی روش پیشنهادی و بررسی یافته‌های حاصل از آن می‌پردازیم.

## فصل ۵

### ارزیابی روش پیشنهادی

«من شکست نخورده‌ام. من فقط ۱۰،۰۰۰ راه پیدا کردم که کار نمی‌کند.»

#### توماس ادیسون

ما روش پیشنهادی خود را روی قالب فایل PDF به عنوان یک قالب فایل با ساختار پیچیده و ترکیبی و نرم‌افزار متن‌باز و رایگان MuPDF [۲۵] مورد ارزیابی قرار دادیم. در این فصل به تشریح چیدمان آزمایش‌ها، معیارهای ارزیابی، یافته‌های حاصل از آزمایش‌ها و نتیجه‌گیری از آنها می‌پردازیم. هدف علاوه بر اثبات کارایی و خوب بودن روش پیشنهادی، شناسایی و تفکیک پارامترهای مهم در هنگام استفاده از فنون یادگیری ماشینی در آزمون فازی و تولید خودکار داده آزمون است. استفاده از یادگیری ماشینی در آزمون فازی حوزه پژوهشی بدیع و ناشناخته‌ای است. بنابراین در این فصل معیارهایی را که در هر روش پیشنهادی باقیستی مورد ارزیابی قرار بگیرند، نیز مشخص خواهیم نمود. سپس روش پیشنهادی خود را با آنها مورد ارزیابی قرار می‌دهیم.

#### ۱-۵ مورد مطالعاتی

نرم‌افزار MuPDF [۲۵] مجموعه غنی از کتابخانه‌ها، API‌ها و ابزارهای کاربردی برای کار با فایل‌های PDF است. این نرم‌افزار مشتمل بر سه قسمت API، ابزار mutool و ابزار PDF‌خوان است. ابزار mutool برای ایجاد و تغییر فایل‌های PDF استفاده می‌شود. ابزار PDF‌خوان نیز برای نمایش فایل‌های PDF به کار می‌رود. بسته نرم‌افزاری کامل، به زبان C نوشته شده و در سیستم عامل‌های ویندوز، لینوکس و Mac و همچنین اندروید قابل استفاده است. به دلیل قابلیت حمل بالا و کاربرد زیادی که دارد، پیدا کردن خطا در آن حائز اهمیت است. در

آزمایش‌ها ما ابزار PDF‌خوان نسخه ویندوز را مورد آزمون قرار دادیم. منظور از عبارت MuPDF در ادامه این فصل همان ابزار PDF‌خوان است.

MuPDF البته در نسخه‌های جدید قالب‌های فایل دیگری را نیز پشتیبانی می‌کند؛ از جمله قالب‌های فایل XPS و EPub. اما قالب فایل PDF بدون شک رایج‌ترین قالب اسناد و انتشارات الکترونیکی است که بیشتر مردم از آن استفاده می‌کنند. از این‌رو تضمین کیفیت و امنیت PDF‌خوان‌ها بهویژه در تلفن‌های همراه اهمیت دو چندانی دارد. افزون بر آنچه گفته شد قالب فایل PDF و نرم‌افزار MuPDF تمامی مسائلی را که در بخش ۱-۲ بدان اشاره کردیم، یعنی ساختار پیچیده ورودی، ساختار پیچیده کد و غیره، دارند و از طرفی در فصل ۳ دیدیم که فازرهای مبتنی بر جابه‌جایی مانند AFL و AFL افزوده نتواسه بودند به پوشش کد خوبی حین آزمون فازی این نرم‌افزار دست‌یابند. بنابراین انتخاب آنها به عنوان مورد مطالعاتی به دلایل بالا و نیز در جهت فراهم شدن امکان مقایسه با کارهای پیشین، قابل توجیه است. آزمایش‌ها برای هر قالب فایل و هر نرم‌افزار دیگری به نحوی که در ادامه این فصل توضیح داده می‌شود قابل اجرا است.

ساختار قالب فایل PDF در پیوست ۱ به‌طور کامل بررسی شده است. بخش عمدۀ این ساختار را اشیای متنی تشکیل می‌دهند. از مدل‌های معرفی شده در بخش ۲-۴ برای یادگیری ساختار اشیای متنی استفاده می‌کنیم؛ زیرا یادگیری فایل PDF به صورت یکجا با توجه به اینکه قسمت‌هایی از آن برای آدرس‌دهی استفاده می‌شوند امکان‌پذیر نیست. اما هر PDF یا MuPDF یا هر PDF خوان دیگری فایل‌های کامل PDF را به عنوان ورودی می‌پذیرد به همین لازم است تا اشیای تولید شده را به یک فایل تبدیل کنیم. برای این کار اشیای جدید را با استفاده از سازوکار توضیح داده شده در پیوست آ در مورد بروزرسانی یک فایل PDF، به یک فایل معتبر از پیش موجود، که آن را میزبان<sup>۱</sup> می‌نامیم، اضافه می‌کنیم. در نتیجه یک فایل PDF جدید در اختیار خواهیم داشت که ساختار آن معتبر بوده و برخی اشیای آن تغییر داده شده و بازنویسی شده‌اند.

## ۲-۵ معیارهای ارزیابی

در تولید خودکار داده آزمون به روش یادگیری ژرف، تعداد زیادی پارامتر وجود دارد که می‌توان تأثیر آنها را در فرایند آزمون فازی سنجید. اما مهم‌ترین هدف در فرایند آزمون فازی همان‌گونه که پیش از این هم ذکر شد یافتن خطای SUT است. هدف مهم بعدی افزایش میزان پوشش کد برای نیل هر چه بیشتر به هدف اول است. هدف دیگر که مختص به این روش است یادگیری هر چه بهتر ساختار فایل ورودی است. براساس اهداف اشاره شده، معیارهای ارزیابی زیر را در این فصل لحاظ کرده‌ایم:

- خطای دقت مدل‌های یادگیری ژرف. شامل خطای دقت هر مدل روی مجموعه‌های آموزش و ارزیابی،

<sup>1</sup>Host

میزان و نحوه تغییر آنها با گذشت زمان. این معیارها توسط کد نوشته شده به ابزار Tensorboard گزارش شده و در حین فرایند آموزش یا در پایان آن قابل مشاهده هستند.

- سرگشتگی. برای ارزیابی میزان خوب بودن مدل زبانی در پیش‌بینی نشانه بعدی توالی داده شده. معیار سرگشتگی از رابطه ۲-۱۴ و با جایگذاری خطای مدل روی مجموعه ارزیابی، برای آن مدل محاسبه می‌گردد.

- پوشش کد. در هر اجرای SUT، تعداد خطوط برنامه و نیز تعداد بلوک‌های اولیه‌ای که اجرا شده‌اند، شمارش می‌شود. با داشتن کل خطوط و بلوک‌های اولیه درصد پوشش کد نیز قابل محاسبه است. برای یک مجموعه آزمون، یعنی چندین بار اجرای متوالی SUT پوشش کد عبارت خواهد بود از اجتماع پوشش کد تک‌تک هریک از اجرایها. بنابراین منظور از اصطلاح کلی پوشش کد در ادامه پوشش بلوک‌های پایه است.

- خطا و آسیب‌پذیری. SUT تحت ابزار پایش Application Verifier اجرا می‌شود که خطاها احتمالی را ثبت و شماره‌گذاری می‌کند. سپس می‌توان با تحلیل این خطاها در یک محیط اشکال زدا، ضمن تعیین مکان و علت خطا، امکان قابل بهره‌برداری بودن آنها را بررسی کرد.

آزمایش‌ها باستی به‌گونه‌ای طراحی شوند که هریک از آنها تأثیر یک پارامتر خاص را بر روی معیارهای بالا، به‌طور واضح نمایان سازد؛ اما، پارامترها کدامند؟ ما در این بخش، چندین پارامتر و مقادیر قابل آزمایش برای هریک را شناسایی و معرفی می‌کنیم. فهرستی از این پارامترها و مقادیری که می‌توانند بپذیرند در جدول ۵-۱ ذکر شده است. همان‌طور که مشاهده می‌شود برخی از پارامترها مقادیری پیوسته و برخی دیگر مقادیری گسسته می‌پذیرند. بر این باور هستیم که هر روش تولید خودکار داده آزمون با استفاده از یادگیری ماشینی روی چنین پارامترهایی بنا می‌شود، لذا شناسایی و تفکیک این پارامترها پیش از هرگونه طرح آزمایشی لازم و ارزشمند است.

نوع مدل در سطر اول جدول ۵-۱ برای هر مدل، خود دارای آبرپارامترهایی مانند تعداد عصب، لایه، نرخ یادگیری و غیره است که در فصل قبل به آنها اشاره شد. بنابراین جدول ۵-۱ یک دید جامع از همه پارامترهایی که تا اینجا معرفی شدند، در اختیار می‌گذارد. در بخش بعد چیدمان آزمایش‌ها خود را براساس این پارامترها شرح خواهیم داد.

**جدول ۵-۱ :** پارامترهای مؤثر در تولید خودکار داده آزمون با روش‌های یادگیری ماشینی و مقادیر قابل آزمایش برای آنها.

ردیف	پارامتر	مقادیر قابل آزمایش
۱	نوع مدل	• انواع مدل‌های یادگیری ژرف
۲	راهبرد	• حریصانه • نمونه برداری
۳	تنوع ( $D$ )	• ترکیبی (حریصانه + نمونه برداری) • چند فهرست بهتر • $(0, \infty)$
۴	روش فاز	• بدون فاز • تصادفی
۵	الگوریتم فاز	• مبتنی بر مدل • ترکیبی • فاز داده • فاز فراداده
۶	تعداد دوره آموزش	• $\{1, 2, 3, \dots\}$
۷	نوع فایل	• متنی • متنی + دودویی
۸	استفاده از میزبان	• استفاده • عدم استفاده
۹	تعداد میزبان	• $\{1, 2, 3, \dots\}$
۱۰	میزان تغییر میزبان	• ثابت • متغیر

جدول ۵-۲: مشخصه‌ها و زمان آموزش مدل‌های جدول ۴-۱ در فرایند آموزش.

۱	۲	۳	۴	پارامتر / شماره مدل
۵۰	۵۰	۵۰	۵۰	طول توالی‌های ورودی ( $d$ )
۱	۱	۳	۳	گام پرش ( $j$ )
۵۰	۵۰	۵۰	۵۰	تعداد دوره
۹:۳۰'	۵:۴۵'	۱:۴۵'	۱:۰۰'	زمان تقریبی یک دوره (ساعت: دقیقه)
۵/۴۱	۹/۹۹	۲/۷۶	۱/۲۴	حجم تقریبی مدل (مگابایت)

## ۳-۵ چیدمان آزمایش‌ها

برای آموزش مدل‌های یادگیری معرفی شده در بخش ۴-۴ (جدول ۴-۱) و تولید داده‌های آزمون از یک سیستم فیزیکی با پردازنده گرافیکی NVidia GTX 1080، پردازنده مرکزی Intel Core i7 و ۲۰ گیگابایت حافظه اصلی به همراه سیستم عامل Ubuntu 16.04 x64 استفاده کردیم. آزمون‌های فازی را نیز روی ماشین مجازی با پردازنده Intel Core i7 و ۸ گیگابایت حافظه اصلی به همراه سیستم عامل Windows 10 x64 انجام دادیم. ما همچنین از نسخه نهایی MuPDF 1.11(2017-04-11) در زمان انجام آزمایش‌ها یعنی نسخه Final برای آزمون فازی استفاده کردیم. با توجه به نهایی بودن نسخه انتظار می‌رود خطاهای نسخه تا حد زیادی، در مقایسه با نسخه‌های آلفا، بتا و RC برطرف شده باشند و لذا شناسایی خطای کار سخت‌تری خواهد بود. مشخصه‌های کامل مدل‌ها در فرایند آموزش به همراه تعداد دوره و زمان آموزش هر دوره، در جدول ۵-۵ آمده است.

پیچیدگی مدل‌های جدول ۵-۵ با افزایش شماره، افزایش می‌یابد. یعنی مدل ۲ از مدل ۱ پیچیده‌تر در نظر گرفته شده است. همان‌طور که در جدول هم مشخص است مدل‌های پیچیده‌تر زمان آموزش طولانی‌تری دارند. برای مدل‌های پیچیده‌تر منطقی است که نمونه‌های آموزشی بیشتری داشته باشیم. در نتیجه برای مدل‌های ۳ و ۴ گام پرش را ۱ و برای مدل‌های ۱ و ۲ گام پرش را ۳ در نظر گرفتیم. همچنین برای مدل شماره ۳ از روش منظم‌سازی Dropout  $p = 0.3$  [۴۲] استفاده کردیم. در نهایت هر مدل را دست‌کم برای ۵۰ دوره روی داده‌های مجموعه آموزش، آموزش دادیم.

## ۱-۳-۵ مجموعه داده

آموزش موفق شبکه‌های عصبی ژرف مستلزم داشتن مجموعه داده به اندازه کافی بزرگ و مناسب است. در یادگیری ماشینی هرچه قدر تعداد داده‌ها بیشتر باشد، یادگیری بهتر انجام خواهد شد. هنوز در بسیاری از زمینه‌ها

مجموعه داده کافی برای آموزش وجود ندارد و این یکی از محدودیت‌های استفاده از شبکه‌های عصبی ژرف است. برای یادگیری آماری ساختار اشیای فایل PDF پیکره بزرگی از فایل‌های PDF جمع‌آوری کردیم؛ زیرا چنین پیکره‌ای از قبل وجود نداشت. بخشی از این پیکره شامل مجموعه داده‌های آزمون PDF خوان<sup>۱</sup> Mozilla است که در مرورگر وب Firefox و دیگر پروژه‌های این شرکت استفاده می‌شود و به صورت رایگان در دسترس است. بخش دیگری از آن PDF‌های استفاده شده در فازرهای دیگر مثل AFL است و بالآخره بخشی بزرگی نیز از طریق وب جمع‌آوری شد به‌ نحوی که تنوع خوبی از لحاظ اندازه و محتوا داشته باشد. پیکره نهایی در حال حاضر شامل ۶۱۰۰ فایل PDF مختلف و طبقه‌بندی شده است که ما آن را تحت نام IUST PDF Corpus<sup>۲</sup> منتشر کردہ‌ایم و از آن می‌توان در موارد دیگر نیز استفاده کرد.

از فایل‌های داخل این پیکره که اندازه آنها بین ۱ تا ۹۰۰ کیلوبایت و تا حداقل ۷۹۳۵ کیلوبایت متغیر است، در حدود ۴۱۵۳ تعداد شیء داده‌ای PDF با طول‌های گوناگون استخراج شد که به عنوان مجموعه داده جهت یادگیری ساختار اشیای داده‌ای PDF در نظر گرفته شده‌اند. این اشیاء سپس همگی در یک فایل الحاق شدند که حجم فایل نهایی در حدود ۷۰ مگابایت است. از این تعداد در حدود ۱۳۷۱۵۷ شیء حاوی جریان‌های دودویی بودند که محتوای همه این جریان‌ها پس از شناسایی با توکن دودویی stream جایگزین شدند؛ زیرا، همان‌طور که قبل از این هم اشاره کردیم نیازی به شرکت دادن آنها در فرایند آموزش نیست و بعداً مجلداً آنها را اضافه می‌کنیم.

## ۲-۳-۵ پیش‌پردازش داده‌ها

تعداد اشیای استخراج شده بسیار زیاد هستند. قبل از انجام عملیات یادگیری مدل بایستی تعدادی از این اشیا را به نحوی از مجموعه داده حذف کرد. برای این منظور عملیات پیش‌پردازشی طی مراحل مختلف روی مجموعه داده انجام دادیم. مهمترین ویژگی در دسترس در تنظیم ابرپارامترهای مدل یادگیری ژرف در اینجا طول اشیای داده‌ای PDF است که هم در فرایند آموزش و هم در فرایند تولید نقش تأثیرگذاری دارد. لذا داشتن توزیعی همگن از اشیای PDF بر حسب طول آنها فرضیه‌ای است که در اینجا در نظر گرفته شده است و در عملیات پیش‌پردازش نیز معیار طول اشیای داده‌ای بوده است. ویژگی طول اشیاء از این منظر مهم است که شبکه LSTM بر روی توالی‌های بسیار طولانی عملکرد ضعیفی دارد و بهتر است چنین توالی‌هایی را در نظر نگیریم. مراحل پیش‌پردازش داده‌ها به ترتیب در زیر ذکر شده است.

### ۱. هر شیء داده‌ای یک موجودیت مستقل درنظر گرفته شده و ابتدا مجموعه داده بر حسب طول اشیای

<sup>1</sup><https://github.com/mozilla/pdf.js/tree/master/test/pdfs>

<sup>2</sup>[https://github.com/m-zakeri/iust\\_deep\\_fuzz/tree/master/dataset](https://github.com/m-zakeri/iust_deep_fuzz/tree/master/dataset)

داخل آن مرتب گردید. سپس کلیه اشیاء داده‌ای به‌شکل

obj

/

endobj

و

obj

null

endobj

از مجموعه داده اولیه حذف شدند؛ زیرا، حاوی بار اطلاعاتی مفیدی نیستند.

۲. پس از اعمال مرحله ۱ تعداد اشیای مجموعه داده به ۴۹۴۹۷۹ کاهش یافت. اکنون به‌طور تقریبی هر صدک از مجموعه داده شامل حدود ۴۹۵۰ شیء داده‌ای PDF است. برای حذف هرچه بیشتر داده‌های پرت صدک‌های اول و آخر را نیز از مجموعه داده حذف کردیم. به‌این ترتیب در پایان این مرحله تعداد ۴۸۵۰۸۰ شیء داده‌ای PDF باقی ماند.

۳. مرحله اصلی پیش‌پردازش خارج کردن داده‌های outlier و extreme value با استفاده از معیار *iqr*<sup>۱</sup> توسط ابزار WEKA<sup>۲</sup> است. ورودی این مرحله مجموعه داده مرحله ۲ است و خروجی آن حاوی تعداد ۴۷۷۱۰۴ شیء داده‌ای PDF است. برای تعیین دو مقدار نامبرده یعنی outlier و extreme value به ترتیب از ضرایب ۳ برای ضریب outlier و ۶ برای ضریب extreme value استفاده شد؛ که مقادیر پیش‌فرض WEKA برای پاک‌ساز *iqr*<sup>۲</sup> هستند.

۴. در این مرحله هدف تقسیم مجموعه داده به سه بخش مجموعه آموزش، مجموعه ارزیابی و مجموعه آزمون است. از آنجایی که یادگیری مدل ما در حالت بدون نظارت انجام می‌شود<sup>۳</sup>، از مجموعه آزمون فقط برای انتخاب پیشوند جهت تولید اشیای جدید استفاده خواهد شد. به‌عبارت دیگر مدل روی مجموعه آموزش، آموزش داده می‌شود و در هنگام تولید اشیای جدید PDF مقادیر اولیه از مجموعه آزمون برداشته می‌شود. برای این منظور ۲۵ درصد از کل تعداد اشیای PDF را برای مجموعه آزمون و ۷۵ درصد را برای مجموعه آموزش/ ارزیابی در نظر گرفتیم. برای ایجاد این تفکیک نیز از فیلتر

<sup>1</sup><https://www.cs.waikato.ac.nz/ml/weka/>

<sup>2</sup>Filter

<sup>3</sup> برچسب‌ها از خود مجموعه آموزش برداشته می‌شوند.

**جدول ۳-۵:** نحوه و درصدهای تقسیم مجموعه داده به مجموعه‌های آموزش، ارزیابی و آزمون.

مجموعه	تعداد شیء	درصد
آموزش	۲۶۸۳۷۱	۵۶/۲۵
ارزیابی	۸۹۴۵۷	۱۸/۷۵
آزمون	۱۱۹۲۷۶	۲۵
جمع	۴۷۷۱۰۴	۱۰۰

موجود در ابزار WEKA با دانه اولیه ۱۰۱ (کوچکترین عدد اول سه رقمی) و مقدار  $fold = 3$  (یعنی بعد از درهم آمیزی<sup>۱</sup>) و تقسیم کل مجموعه داده به ۴ قسمت با نسبت‌هایی برابر با مجموعه داده اولیه، قسمت سوم آن برای مجموعه آزمون کنار گذشته شود) استفاده شد. در نهایت تعداد ۳۵۷۸۲۸ شی در مجموعه‌های آموزش/ ارزیابی و تعداد ۱۱۹۲۷۶ شی داده‌ای هم در مجموعه آزمون قرار گرفتند.

۵. در تکمیل مرحله ۴، نیاز است تا مجموعه آموزش/ ارزیابی جدا شده، حاوی ۳۵۷۸۲۸ شیء خود به دو مجموعه مجزای آموزش و ارزیابی تقسیم شود. برای این منظور این مجموعه آموزش/ ارزیابی به ۴ قسمت تقسیم گردید و قسمت چهارم به عنوان مجموعه ارزیابی در نظر گرفته شد. سه قسمت اول هم مجموعه آموزش را تشکیل می‌دهند. درصد تعداد اشیاء هریک از مجموعه‌های آموزش، ارزیابی و آزمون در جدول ۳-۵ آمده است.

۶. آخرین مرحله پیش‌پردازش نرم‌افزاری<sup>۲</sup> متن است. در این مرحله برخی کاراکترهای بسیار کم تکرار (فراوانی نسبی نزدیک صفر) حذف یا با کاراکترهای مناسبی جایگزین شدن. در واقع طی این مرحله اندازه مجموعه واژگان کاراکترها از ۹۶ به ۶۴ تقلیل یافت. علت انجام این مرحله، آن است که کاراکترهای با تعداد کم داده پرت محسوب می‌شوند و به دلیل اینکه در مجموعه آموزش تعداد تکرار آنها پایین است عملأً احتمال بسیار ناچیزی به آنها تخصیص داده می‌شود که در واقع ضرورتی به این کار نیست. در عین حال با انجام این کار، حجم مکعب داده‌ها کمتر شده و فرایند آموزش سریع‌تر انجام می‌شود.

<sup>1</sup>Shuffle

<sup>2</sup>Normalization

### ۳-۵-۳ انتخاب فایل‌های میزبان

در [۱۱] انتخاب فایل‌های میزبان به صورت انتخاب سه فایل با کمترین حجم از پیکره انجام گرفته است. در واقع این انتخاب بدون هدف خاص و کاملاً تصادفی صورت گرفته است. پیرو روش آنها ما نیز برای آزمایش‌های خود سه فایل را به عنوان فایل‌های میزبان انتخاب کردیم، با این تفاوت که برای انتخاب فایل‌های میزبان ابتدا میزان پوشش کد را به صورت مجزا برای همه فایل‌های IUST PDF Corpus به دست آورده و سپس سه فایل با بیشترین میزان پوشش کد، کمترین میزان پوشش کد و نیز میزان پوشش کد میانگین، به ترتیب با نام‌های host3\_avg و host2\_min و host1\_max به عنوان فایل‌های میزبان انتخاب شدند. هدف از این کار بررسی تأثیر روش پیشنهادی بر بهبود میزان پوشش کد در فایل‌های با اختلاف پوشش کد زیاد است و سپس انجام آزمون فازی براساس بهترین میزبان است. ما همچنین برای هر آزمایش پوشش کد مجموع هر سه میزبان را با ادغام پوشش کدهای هریک از آنها تحت عنوان host123 گزارش کردیم. این کار کمک می‌کند تا بتوانیم نتایج دو آزمایش کاملاً جدا را در حالت کلی با یکدیگر مقایسه کنیم.

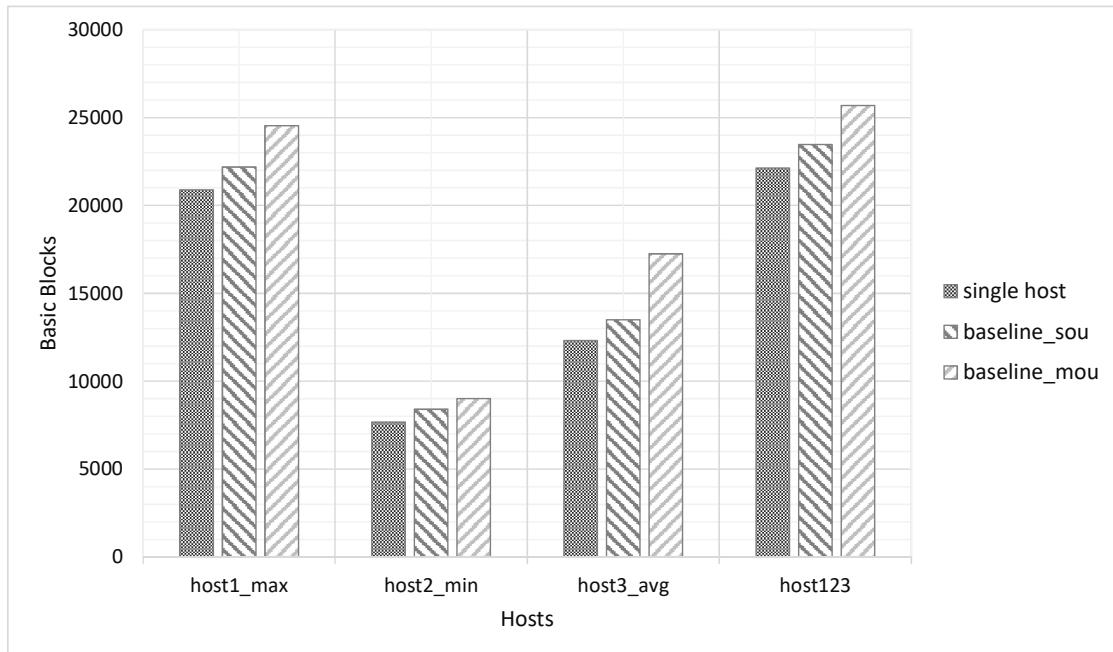
### ۴-۳-۵ پوشش کد مبنای

برای آنکه بتوانیم در ک معنادار و مقایسه‌پذیری از پوشش کدهای به دست آمده در هر آزمایش داشته باشیم. ابتدا پوشش کد مبنای را مطرح، اندازه‌گیری و گزارش می‌کنیم. پوشش کد مبنای، پوشش کد در غیاب استفاده از روش پیشنهادی برای تولید خودکار داده آزمون است. برای این منظور، ابتدا از مجموعه آزمون، تعداد ۱۰۰۰ شیء داده‌ای PDF استخراج و در دو حالت به فایل‌های میزبان تزریق می‌نماییم:

- بروزرسانی یک شیء<sup>۱</sup> (SOU<sup>1</sup>): برای هر فایل میزبان در هر مرحله یک شیء را انتخاب و آخرین شیء میزبان را با آن بازنویسی می‌کنیم، یعنی در واقع تغییر تنها یک شیء از فایل میزبان در هر مرحله بدون توجه به اندازه و تعداد اشیای فایل میزبان.
- بروزرسانی چندین شیء (MOU<sup>2</sup>): تغییر در صد ثابتی از اشیای فایل میزبان در هر مرحله. در این روش ما برای host1\_max تعداد  $\frac{1}{\Delta}$ ، برای host2\_min تعداد  $\frac{1}{\Delta}$  و برای host3\_avg تعداد  $\frac{1}{\Delta}$  اشیای هر فایل را به صورت تصادفی انتخاب و با اشیای انتخابی از مجموعه آزمون بازنویسی کردیم. یعنی در این روش هر میزبان را به نسبت بیشتری تغییر می‌دهیم، به‌امید اینکه این تغییرات سبب افزایش پوشش کد شود. می‌توان اشیاء را با روش‌هایی غیر از روش تصادفی نیز برای بازنویسی انتخاب کرد؛ به عنوان مثال

<sup>1</sup>Single Object Update

<sup>2</sup>Multiple Objects Update



**شکل ۵-۱:** پوشش کد برای هریک از میزبان‌ها و پوشش کدهای مبنای.

انتخاب براساس ترتیب نزولی یا صعودی ID هر شیء یا براساس بزرگی آدرس آنها. در این پایان‌نامه ما فقط روش تصادفی را آزمایش کردیم.

به این ترتیب سه مجموعه ۱۰۰۰ فایلی از حالت اول و سه مجموعه ۱۰۰۰ فایلی از حالت دوم حاصل می‌گردد که آنها را به ترتیب<sup>۱</sup> baseline\_sou و<sup>۲</sup> baseline\_mou می‌نامیم. پوشش کد هر مجموعه و نیز پوشش کد اجتماع هر سه میزبان به عنوان یک مقدار پایه برای آزمایش‌های بعدی اندازه‌گیری و تعیین شدند. نمودار شکل ۵-۱ مقادیر پوشش کد مبنای را در مقایسه با پوشش کد هریک از میزبان‌ها به صورت تنها نشان می‌دهد. لازم به ذکر است که هر یک از میزبان‌ها پیش از بروزرسانی‌های افزایشی برای تولید داده جدید بررسی شدند و معتبر بودن ساختار آنها تأیید گردید. لذا، صرف انجام این کار به منزله آزمون فازی نخواهد بود؛ زیرا، هنوز داده‌ها را بدشکل (فاز) نکرده‌ایم. هرچند ممکن است در این فرایند نیز خطای قابل شناسایی باشد.

### مشاهدات

- میزان پوشش کد هریک از baseline از پوشش کد فایل میزبان به تنها یک بیشتر است. یعنی تغییر میزبان‌ها منجر به اجرای بلوک‌های پایه جدید و احتمالاً مسیرهای اجرایی جدیدی، شده است.

<sup>1</sup>Baseline for Single Object Update

<sup>2</sup>Baseline for Multiple Object Update

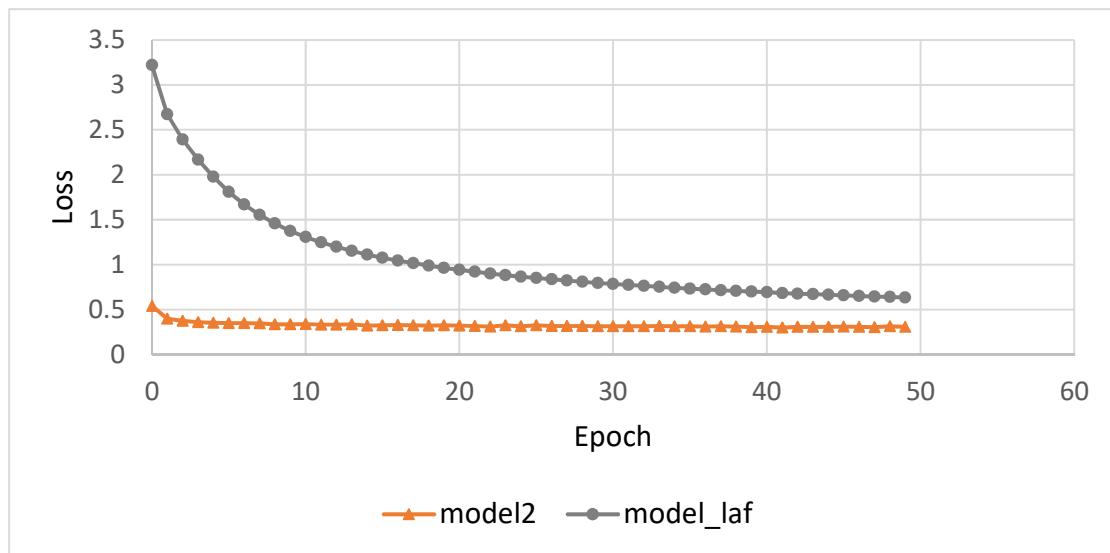
- میزان پوشش baseline‌ها متناسب با میزان‌ها است. یعنی به عنوان مثال host1\_max بیشترین پوشش کد را در هر سه حالت تنها، baseline\_sou و baseline\_mou داشته است. این بدین معنی است که انتخاب فایل میزان بسیار حائز اهمیت است و بخش قابل توجهی از پوشش کد در هر حالت مربوط به فایل میزان است.
- میزان پوشش کد baseline\_mou در تمامی میزان‌ها از baseline\_sou بیشتر است. این بدین معنی است که با افزایش میزان تغییر فایل شاهد افزایش میزان پوشش کد بوده‌ایم.
- بیشترین پوشش کد مربوط به اجتماع پوشش کدهای baseline\_mou در host123 است که نشان می‌دهد هر کدام از میزان‌ها مجموعه دستورات متفاوتی را اجرا کرده‌اند.
- درنهایت اعداد و ارقام پوشش کد در محدوده ۲۵ هزار بلوک پایه بوده که نشان می‌دهد MuPDF نرم‌افزاری پیچیده است.

## ۴-۵ آزمایش‌ها، یافته‌ها و مقایسه نتایج

در این بخش آزمایش‌های صورت گرفته و نتایج حاصل از آن را بیان می‌کنیم. همچنین نتایج را با سایر کارهای مرتبط مقایسه خواهیم کرد. چون SUT مورد استفاده در یادگیری و فاز [۱۱] برای آزمون دردسترس نبود، ما روش یادگیری و فاز را نیز پیاده‌سازی و آن را روی ابزار MuPDF آزمایش کردیم. بهاین ترتیب توانستیم مقایسه معناداری بین روش پیشنهادی خود با این روش، به عنوان مرتبط‌ترین کار انجام شده داشته باشیم. نتایج نشان می‌دهد که روش پیشنهادی ما در معیار پوشش کد و همچنین در معیارهای خطأ، دقت و سرگشتگی بهبود داشته است و توانسته روش یادگیری و فاز، که در آزمایش‌ها با نام laf آن را نشان می‌دهیم، را شکست دهد. روش پیشنهادی ما همچنین نسبت به AFL و AFLافزوده بهبودهایی داشته است که خواهیم دید. چون مجموعه داده در مدل‌های یادگیری ژرف بسیار مهم است و روی نتایج تأثیر می‌گذارد در هنگام آموزش مدل laf مجموعه داده را مطابق مقیاس ارائه شده در [۱۱] لحاظ کردیم. بدین زمان آموزش کمتر برای هر دوره در حدود ۱۵ دقیقه به طول انجامید. سایر پارامترها مانند طول توالی‌های آموزشی  $d$  را که در مقاله به آن اشاره نشده بود نیز مشابه مقادیر آن در روش پیشنهادی قرار دادیم.

جدول ۴-۵: سرگشتگی، دقت و خطای مدل‌های مختلف روی مجموعه‌های آموزش و ارزیابی.

معیار / مدل	۱	۲	۳	۴	laf
سرگشتگی	۱/۴۴۰	۱/۳۹۱	۱/۳۳۵	۱/۳۵۰	۱/۸۶۰
بیشینه دقت آموزش	۰/۸۸۶	۰/۹۰۲	۰/۸۹۳	۰/۹۰۹	۰/۸۲۰
بیشینه دقت ارزیابی	۰/۸۸۴	۰/۸۹۵	۰/۹۰۴	۰/۹۰۵	۰/۸۰۰
کمینه خطای آموزش	۰/۳۵۳	۰/۲۹۸	۰/۳۲۴	۰/۲۷۶	۰/۶۲۳
کمینه خطای ارزیابی	۰/۳۶۵	۰/۳۳۰	۰/۲۸۹	۰/۳۰۰	۰/۷۲۴



شکل ۵-۲: نمودار تغییرات خطای مدل‌های ۲ و laf در دوره‌های ۱ تا ۵۰.

### ۴-۵-۱ سرگشتگی، خطای و دقت مدل‌ها

جدول ۴-۵ سرگشتگی، خطای و دقت مدل‌های چهارگانه پیشنهادی و مدل یادگیری و فاز را بعد از گذشت ۵۰ دوره، روی مجموعه‌های آموزش و ارزیابی نشان می‌دهد. سرگشتگی مطابق رابطه ۲-۱۴ و خطای نیز از رابطه ۲-۶ برای مدل‌ها محاسبه شده است که در فصل ۲ آنها را بیان کردیم. دقت نیز معیاری است که توسط Keras برای هر مدل حساب می‌شود. همچنین شکل ۴-۵ روند تغییرات خطای در فرایند آموزش را برای مدل ۲ و مدل laf نشان می‌دهد. مدل ۲ در این نمودار به این دلیل انتخاب گردیده که بیشترین میزان شباهت را از لحاظ معماری و مقادیر آبرپارامترها با مدل laf دارد.

## مشاهدات

- سرگشتگی و خطای همه مدل‌های پیشنهادی از مدل یادگیری و فاز کمتر و دقت آنها از یادگیری و فاز بیشتر است. یعنی مدل زبانی عصی با معماری توضیح داده شده در بخش ۴-۲، در یادگیری ساختار فایل از مدل کدگذار-کدگشا [۱۱] بهتر عمل کرده است.
- بیشترین دقت در بین تمامی مدل‌ها مربوط به مدل ۴ است. مدل ۴ LSTM دوسویه است. شبکه LSTM دوسویه هنگام آموزش علاوه بر کاراکترهای قبلی، کاراکترهای بعدی را نیز در نظر می‌گیرد. بنابراین توانسته است به دقت بیشتر دست یابد. کمترین میزان سرگشتگی نیز متعلق به مدل ۳ است. این مدل همان‌طور که در جدول نیز مشخص است دارای کمترین خطای ارزیابی است. البته اختلاف بین مقادیر مختلف در همه مدل‌های پیشنهادی کم است که نشان می‌دهد همه مدل‌ها قادر به یادگیری و درک ساختار فایل بوده‌اند. سرگشتگی بیشینه، حالت بدون استفاده از مدل یادگیری، برای تنظیمات ما برابر ۶۴ (اندازه بردار واژگان) خواهد بود.
- در شکل ۵-۲، منحنی خطای مدل ۲ در همه دوره‌ها، از منحنی خطای مدل laf پایین‌تر است. البته دوره‌ها زمان متفاوتی دارند بنابراین مقایسه نظریه‌نظری آنها ممکن است جالب به نظر نیاید. با این حال در بازه زمانی مساوی از شروع فرایند آموزش نیز این وضعیت فوق برقرار است. برای مثال پایان دوره ۱ برای مدل ۲ مصادف با پایان دوره ۱۲ برای مدل laf خواهد بود که باز هم خطای مدل laf بیشتر است. ما فرایند آموزش را برای هر دو مدل تا دوره ۱۰۰ ادامه دادیم و تغییر خلافی مشاهده نشد. چنان‌چه در بخش ۴-۴ خواهیم دید پوشش کد مدل ۲ نیز در دوره‌های مختلف از مدل laf بالاتر است.

## ۴-۵ پوشش کد مدل‌های مولد

در آزمایش این بخش، مدل‌های مولد پیشنهادی خود را برای اولین مرحله تولید داده آزمون محک می‌زنیم. برای تولید داده‌های آزمون جدید از روش نمونه‌برداری و پارامتر تنوع  $D$  در همه آزمایش‌ها استفاده کردیم. همچنین هر بار با یک پیشوند تصادفی از مجموعه آزمون تولید داده از مدل را شروع می‌کنیم. می‌خواهیم ببینیم برای میزبان‌های مختلف کدام مدل مولد و نیز کدام تنوع تولید داده، به پوشش کد بالاتری دست می‌یابد. هدف مقایسه کارکرد مدل‌ها و انتخاب مناسب‌ترین مدل برای استفاده در آزمون فازی است. برای این منظور هریک از مدل‌های جدول ۱-۴ را به تعداد ۵۰ دوره آموزش دادیم. در پایان هر دوره یک نمونه از مدل آموزش دیده را ذخیره کردیم. سپس مدل با کمترین خطای از بین نمونه‌های ذخیره شده، برای تولید داده برگزیدیم.

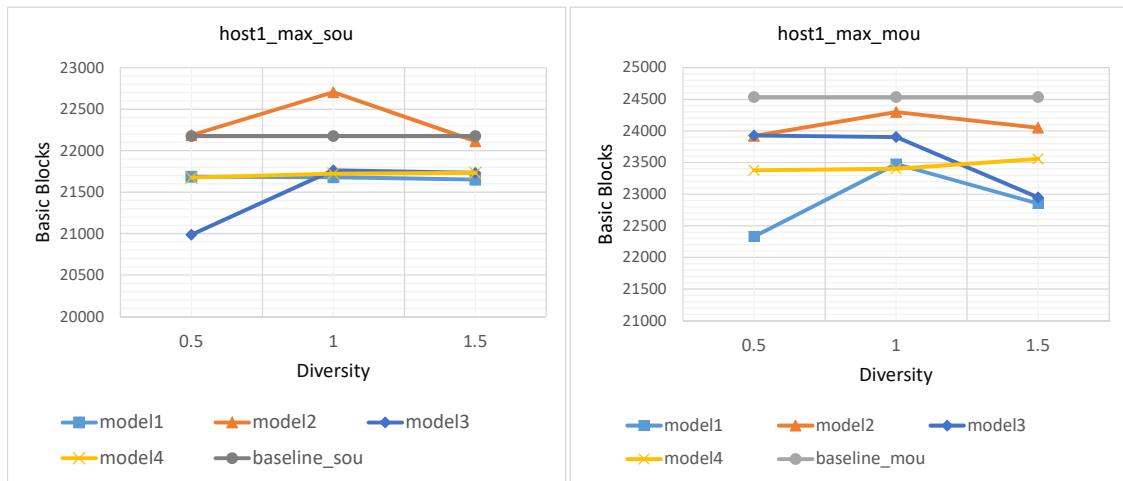
در مرحله بعد برای هر مدل در هنگام تولید داده با هریک از تنوع‌های تولید ۱، ۰/۵ و ۱/۵، تعداد ۱۰۰۰ فایل PDF با هر میزبان تولید کردایم. یعنی در مجموع با هر مدل و هر میزبان ۳۰۰۰ فایل را تولید می‌کنیم. چون ۴ مدل و ۳ میزبان داریم، در کل ۳۶۰۰۰ فایل تولید می‌شود و چون دو روش بروزرسانی شیء SOU و MOU هم نیاز است در نهایت ۷۲۰۰۰ فایل تولید می‌کنیم. دقت شود که در این مرحله هنوز عملیات آزمون فازی یعنی فاز ورودی، مشاهده رفتار برنامه در اشکال زدا و ذخیره خطاهای احتمالی، اعمال نشده است و تنها تأثیر مدل‌ها، تنوع تولید داده از آنها و میزبان‌ها، بر روی میزان پوشش کد مورد نظر است.

برای مقایسه با baseline\_sou تعداد ۱۰۰۰ شیء داده‌ای از هر مدل تولید کرده و برای مقایسه با baseline\_mou تعداد ۳۰۰۰ شیء داده‌ای با هر مدل تولید می‌کنیم. تولید ۱۰۰۰ فایل PDF جدید با استفاده از مدل‌ها برای حالت SOU (با اندازه بافر ۱۰۰<sup>۱</sup>) به طور میانگین ۶۰ دقیقه و برای حالت MOU به طور میانگین ۱۹۰ دقیقه به طول انجامید. در پایان پوشش کد نرمافزار MuPDF روی هریک از ۷۲ مجموعه ۱۰۰۰ فایلی اندازه‌گیری شد. ما همچنین ادغام پوشش‌های کد را برای هر تنوع در قالب host123 اندازه‌گرفتیم. اندازه‌گیری پوشش کد برای هر مجموعه نیز در حدود ۶۰ دقیقه زمان برد. کلیه نتایج به تفکیک میزبان‌ها در شکل‌های ۵-۱۳ تا ۵-۳ تذکر شده است.

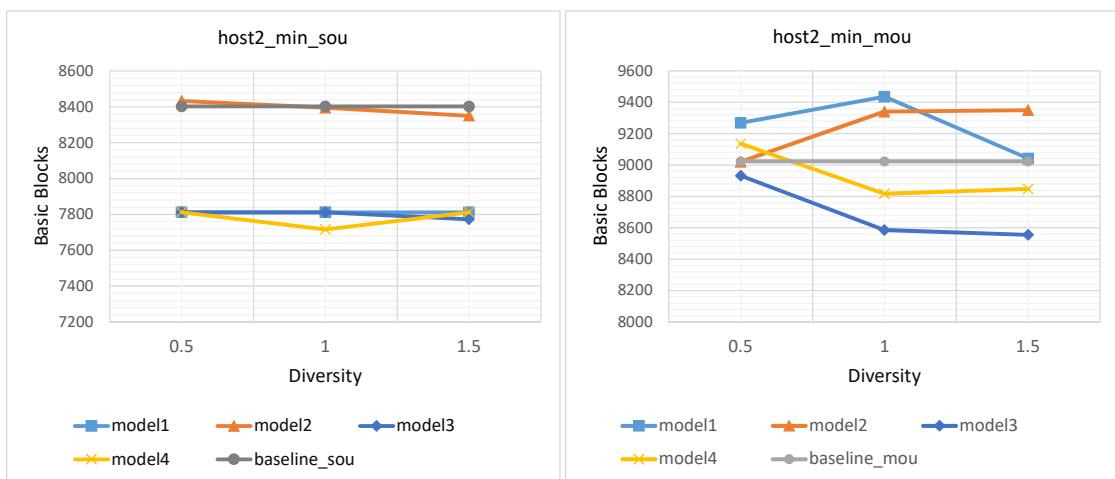
## مشاهدات

- پوشش کد داده‌های تولیدی از پوشش کد مینا در اکثر موارد کمتر است؛ زیرا، اشیای تولید شده به خوش‌شکلی اشیای واقعی نیستند. البته در برخی موارد شاهد افزایش پوشش کد هستیم. از جمله برای MOU در حالت host2\_min.
- تولید داده با تنوع ۱ در بیشتر مدل‌ها و روی اکثریت میزبان‌ها پوشش کد بهتری داشته است. یعنی پارامتر تنوع واقعاً سبب بدشکل شدن و بالا رفتن گوناگونی داده‌های تولید شده و در نتیجه پارامتری مؤثر بوده است.
- افزایش تنوع در مدل LSTM دوسویه در حالت کلی سبب افزایش پوشش کد شده است؛ اما در دیگر مدل‌ها خیر.
- تقریباً در همه نمودارها داده‌های تولید شده توسط مدل ۲، پوشش کد بیشینه را نتیجه داده است. یعنی مدل‌های ژرف ساده بهتر از مدل‌های ژرف پیچیده همچون LSTM دوسویه (مدل ۴)، عمل کرده‌اند. بهمین دلیل مدل ۲ با تنوع تولید داده ۱ پیروز نهایی این سری از آزمایش‌های ما هستند.

<sup>۱</sup> یعنی هر بار یک لیست از ۱۰۰ شیء تنوع توسط مدل مولد بازگردانیده می‌شود.

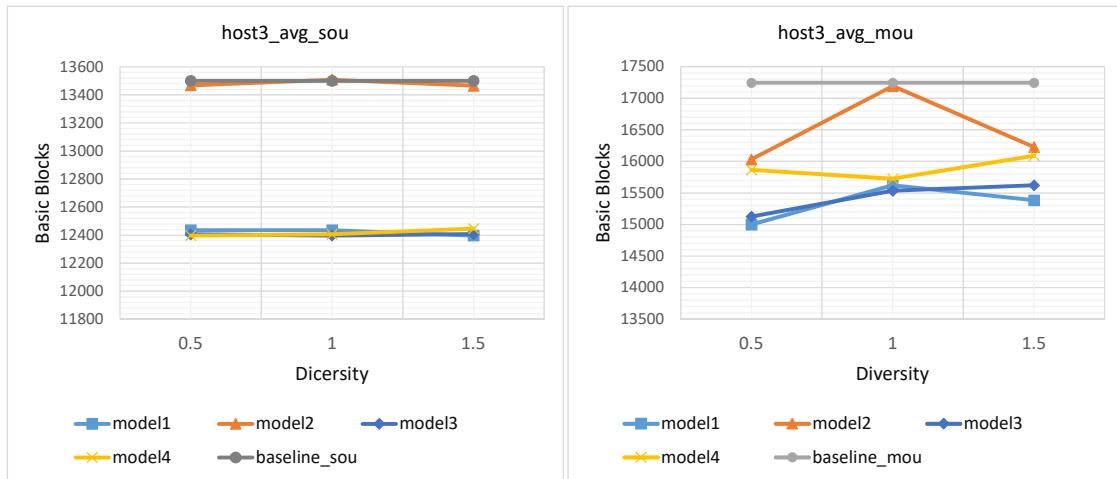


(آ) نمودار تغییرات پوشش کد در تنواع‌های نمونه‌برداری ۰/۵ تا ۱/۵ برای .host1\_max

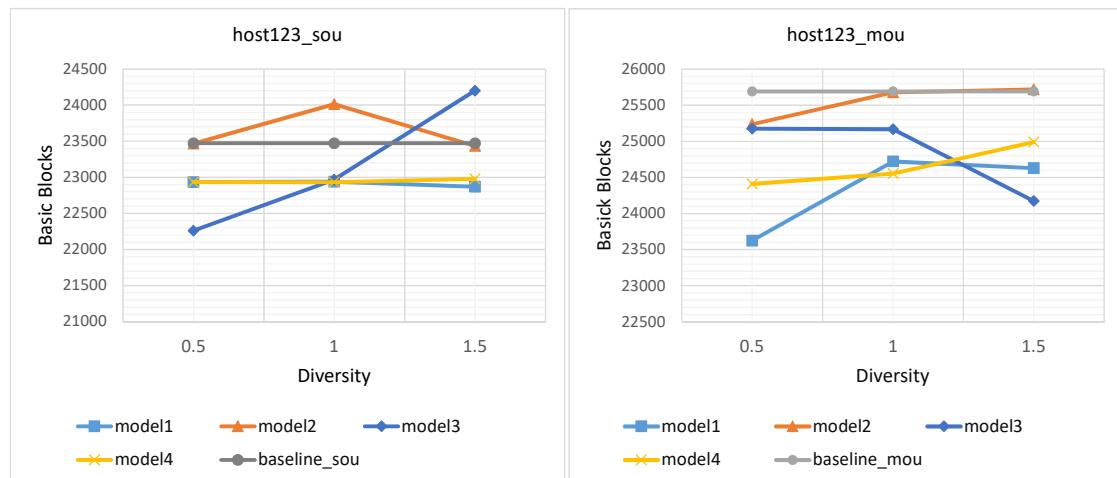


(ب) تغییرات پوشش کد در تنواع‌های نمونه‌برداری ۰/۵ تا ۱/۵ برای .host2\_min

شکل ۳-۵: نمودار تغییرات پوشش کد مدل‌های مختلف بر حسب تنواع.



(ب) تغییرات پوشش کد در تنواع‌های نمونه‌برداری ۰/۵ تا ۱/۵ برای .host3\_avg



(ت) تغییرات پوشش کد در تنواع‌های نمونه‌برداری ۰/۵ تا ۱/۵ برای .host123.

شکل ۵-۳: (ادامه). نمودار تغییرات پوشش کد مدل‌های مختلف بر حسب تنواع.

### ۳-۴-۵ مقایسه با مدل کدگذار-کدگشا

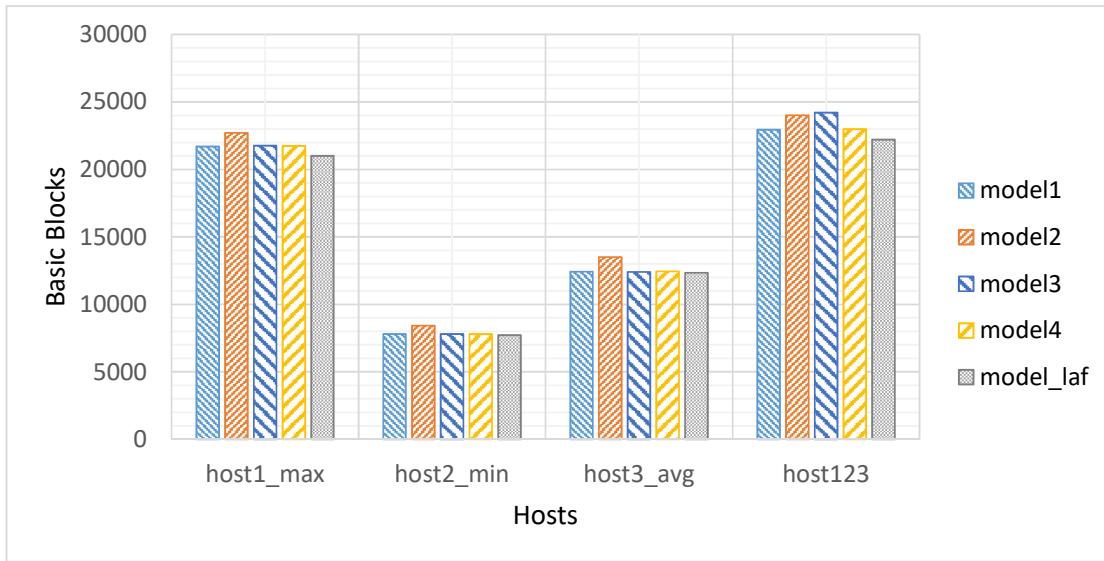
برای مقایسه با [۱۱] ابتدا مدل کدگذار-گشای توضیح داده شده را پیاده‌سازی و آن ۵۰ دوره آموزش دادیم. سپس ۱۰۰۰ فایل PDF را با این مدل تولید و پوشش کد مجموع آنها را اندازه‌گیری کردیم. چون راهبرد نمونه‌برداری برای مدل مذکور نیز بهترین روش گزارش شده است، ما نیز از نمونه‌برداری برای تولید داده‌ها با این مدل استفاده کردیم. برای مدل‌های چهارگانه خود نیز بالاترین پوشش کد کسب شده در میان همه تنوع‌های آزمایش شده در بخش ۲-۴-۵ را به عنوان نماینده پوشش کد برای هر مدل انتخاب کردیم. در نهایت دو حالت SOU و MOU را به صورت مجزا مورد ارزیابی و مقایسه قرار دادیم. نتایج در شکل ۴-۵ نشان داده شده است. شکل ۴-۵آ حالت SOU و شکل ۴-۵ب حالت MOU را نشان می‌دهد.

#### مشاهدات

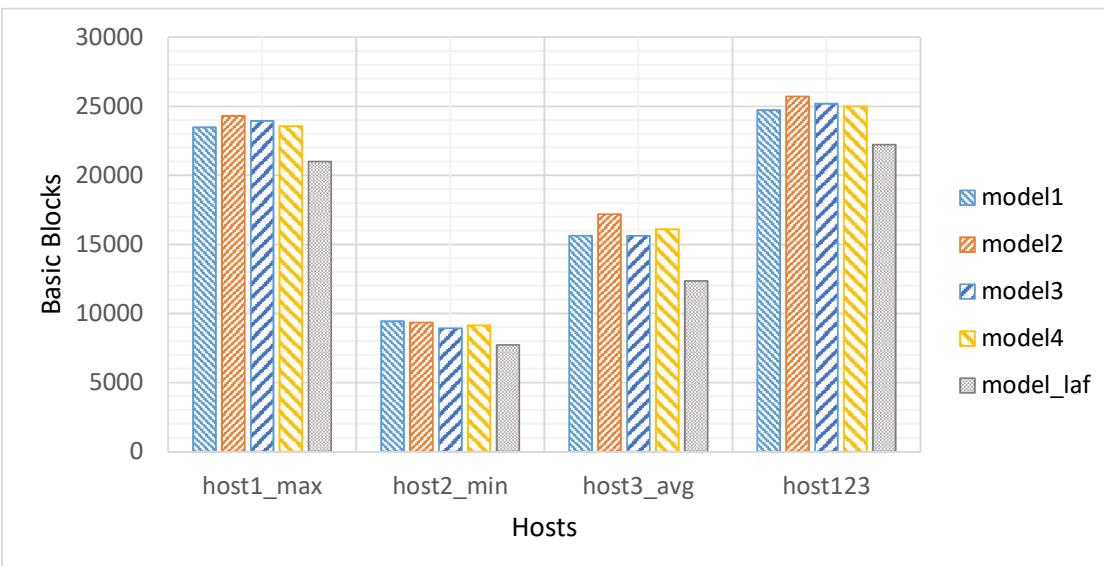
- در حالت SOU و برای host1\_max همه مدل‌های پیشنهادی پوشش کد بالاتری داشته‌اند. در همین حالت دو میزبان دیگر اختلاف پوشش کد ناچیز بوده است. اما در حالت اجتماع پوشش کدها یعنی host123 مدل‌های پیشنهادی بهتر ظاهر شده‌اند یعنی هر مدل برای هر میزبان مجموعه دستورات متفاوتی را اجرا کرده است.
- در حالت MOU مدل‌های پیشنهادی ما با اختلاف بهتر هستند. این نشان می‌دهد که تغییر بیشتر فایل‌های میزبان به افزایش پوشش کد، در تعداد داده آزمون مساوی، می‌انجامد. بنابراین حالت MOU را برای انجام آزمون فازی توصیه می‌کنیم.
- در هر دو حالت اختلاف بین پوشش کد مدل‌ها برای host2\_min و برای host1\_max بیشتر به‌چشم می‌خورد. این امر نشان می‌دهد که انتخاب فایل میزبان در مواردی که نیاز به آن هست، مانند فایل PDF که ساختار پیچیده‌ای دارد و یادگیری کامل آن به‌آسانی محقق نمی‌شود، بسیار حائز اهمیت است و تأثیر چشم‌گیری روی ارتقاء پوشش کد دارد. این انتخاب نبایست تصادفی انجام شود، چون دیدیم افزایش پوشش کد رابطه مستقیمی با پوشش کد فایل میزبان دارد. بنابراین فایل میزبان با پوشش کد بیشتر را برای انجام آزمون فازی توصیه می‌کنیم.

### ۴-۴-۵ مقایسه در دوره‌های مختلف

یک پارامتر قابل ارزیابی که در جدول ۱-۵ مطرح کردیم و تأثیر آن روی پوشش کد در [۱۱] نیز بررسی شده است تعداد دوره‌های آموزش است. به نظر می‌رسد که پوشش کد ارتباط مستقیمی با تعداد دوره‌های آموزش

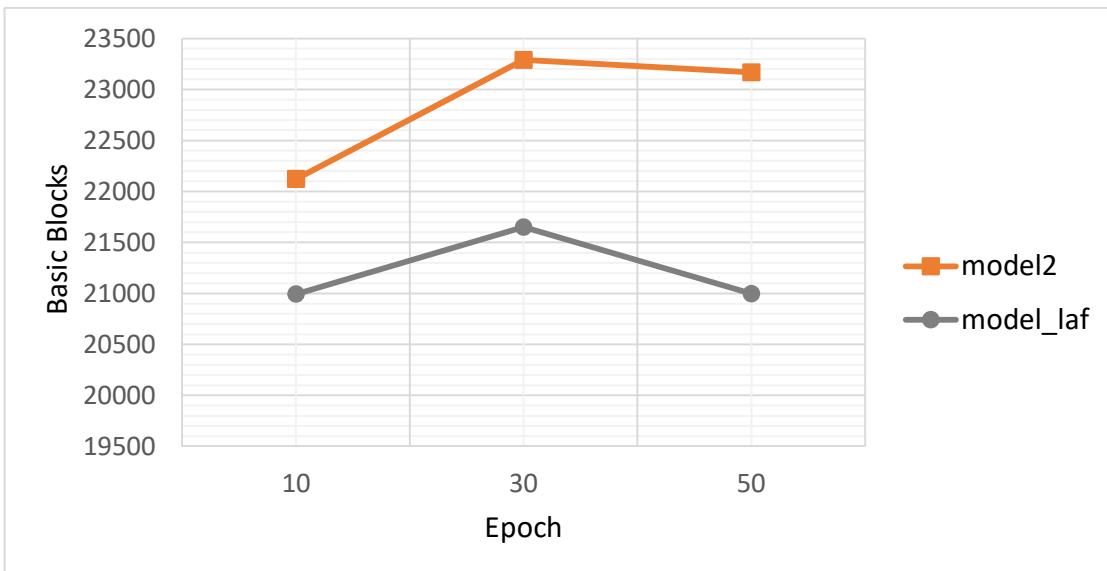


(ا) حالت SOU.



(ب) حالت MOU.

شکل ۴-۵: نمودار پوشش کد مدل‌های مختلف در مقایسه با مدل laf بر حسب فایل‌های میزبان.



**شکل ۵-۵:** نمودار تغییرات پوشش کد بر حسب دوره برای مدل‌های ۲ و laf.

مدل ژرف داشته باشد. برای این منظور پوشش کد ۱۰۰۰۰ فایل تولید شده با استفاده از مدل ۲ را در سه دوره ۱۰، ۳۰ و ۵۰ اندازه‌گیری کردیم. سپس همین کار را برای مدل laf نیز انجام دادیم. برای آن که نتایج قابل اطمینان باشند و اثر پارامترهای تصادفی از بین برود، هر آزمایش را سه مرتبه با سه مجموعه داده مجزا تکرار و میانگین پوشش کدها اندازه گرفتیم. نتایج در شکل ۵ نشان داده شده‌اند.

### مشاهدات

- برای هر دو مدل پوشش کد از دوره ۱۰ به ۳۰ افزایش و سپس در دوره ۵۰ کاهش یافته است. یعنی رابطه مستقیم معناداری بین پوشش کد و تعداد دوره‌های آموزش مدل وجود ندارد.
- با بررسی مقدار خطای مدل در هر دوره متوجه شدیم تا زمانی که نرخ کاهش خطای مدل در فرایند آموزش زیاد است، پوشش کد نیز افزایش می‌یابد؛ چراکه ساختار فایل بهتر یادگیری شده و در نتیجه مدل قادر به تولید داده‌های خوش‌شکل‌تری خواهد بود. هنگامی که نرخ کاهش خطای مدل حد آستانه کمتر می‌شود، افزایش پوشش کد نیز ثابت شده یا اندکی کاهش می‌یابد.
- در همه دوره‌های شکل ۵ پوشش کد مدل پیشنهادی از مدل laf بیشتر بوده است که نشان می‌دهد این مدل ساختار فایل را بهتر یادگرفته است.

## ۴-۵ آزمون فازی عصبی

در پنجمین و آخرین آزمایش نرم‌افزار MuPDF را با استفاده از فازر پیشنهادی در بخش ۴-۵، مورد آزمون فازی قرار دادیم. برای این منظور الگوریتم‌های فاز عصبی داده و فاز عصبی فراداده (الگوریتم‌های ۲-۴ و ۴-۳) را که در بخش ۴-۳ معرفی کرده بودیم، پیاده‌سازی و با استفاده از هریک از آنها تعداد ۱۰۰۰۰ فایل PDF را ایجاد کردیم. تنظیمات پارامترهای ورودی و ثوابت موجود در الگوریتم‌های پیشنهادی به قرار جدول ۵-۵ است. ستون آخر جدول ۵-۵، همچنین مقادیر مجاز برای هریک از ورودی‌ها و ثوابت داده شده را نشان می‌دهد. حداکثر طول یک شیء داده‌ای PDF را در بازه متغیر ۴۵۰ تا ۵۵۰ انتخاب کرده‌ایم؛ زیرا، به طور میانگین طول اشیای استخراج شده مجموعه‌های آموزش و آزمون، در این بازه قرار دارد. در آزمایش‌های آتی خود در نظر داریم تا آزمون فازی را با مقادیر متنوع انتخابی از مجموعه‌های داده شده انجام دهیم و بدین ترتیب قادر خواهیم بود تا اثر هریک از این پارامترها را به طور دقیق‌تری بررسی کنیم.

ما همچنین نسخه‌هایی از الگوریتم‌های ۱-۳ [۱۱] و ۴-۳ [۲۲] را پیاده‌سازی و آزمون فازی را با تولید داده آزمون از طریق این الگوریتم‌ها نیز انجام دادیم. برای حالت تصادفی (الگوریتم ۱-۳) از ابزار FileFuzz که یک فازر تصادفی تحت سیستم عامل ویندوز است و در بخش ۲-۲ معرفی شد، استفاده کردیم. متأسفانه در روش یادگیری و فاز [۱۱]، تقریباً اکثر ابرپارامترهای مورد نیاز در هنگام آزمایش‌ها نامشخص هستند و نویسنده‌گان اشاره‌ای به مقادیر آنها نکرده‌اند. برای این هریک از این ابرپارامترها نظیر  $d$ ، ما همان مقدار استفاده شده در الگوریتم‌های روش پیشنهادی را استفاده کردیم، تا بدین ترتیب شرایط یکسانی بر آزمایش‌ها حاصل باشد. در همه آزمایش‌ها، host1\_max به عنوان میزبان<sup>۱</sup> استفاده شد. نتایج حاصل از پوشش کد روش‌های مختلف در جدول ۶-۵ آمده است. همچنین جدول ۷-۵ میزان بهبود در پوشش کد روش پیشنهادی در مقایسه با کارهای مرتبط را نشان می‌دهد.

### مشاهدات

- الگوریتم فاز عصبی فراداده به پوشش کد کمتری دست پیدا کرده است، زیرا همان‌طور که انتظار می‌رود دستکاری بخش کوچکی در از قالب فایل ممکن است آن را کاملاً نامعتبر کند و در مرحله اولیه تجزیه توسط تجزیه‌گر رد شود. بنابراین الگوریتم در رسیدن به هدف خود یعنی فاز فراداده موفق بوده است.

- هر دو الگوریتم فاز عصبی داده و فاز عصبی فراداده پوشش کد بهتری نسبت به SampleFuzz داشته‌اند که نشان از عملکرد بهتر مدل‌های پیشنهادی در یادگیری ساختار فایل و عملکرد بهتر الگوریتم‌های

<sup>۱</sup> در تولید داده آزمون به روش تصادفی و روش مبتنی بر جایه‌جایی، به جای میزبان، به یک یا تعدادی دانه اولیه نیاز داریم که در اینجا از همان host1\_max استفاده گردید.

**جدول ۵-۵:** مقادیر مجاز و مقادیر داده شده به ثوابت و ورودی‌های الگوریتم‌های فازی عصبی داده و فاز عصبی فرا داده در هنگام آزمون فازی قالب فایل PDF

مقادیر مجاز	مقادیر استفاده شده	پارامتر ورودی / ثابت
$1, 2, 3, 4, laf$	۲	Learnt model M
ثابت رشته‌ای $(0, +\infty)$	انتخاب از مجموعه آزمون ۱	Sequence prefix P Diversity D
ثابت رشته‌ای $(0, 1]$	۰/۱۰	Fuzzing rate FR
ثابت رشته‌ای $(len(P), +\infty)$	endobj stream	End token ET Binary token BT
$(0, 1)$	$(450, 550)$	$(a, b)$
$(0, 1)$	۰/۵۰	(DataNeuralFuzz) $\alpha$
$(0, 1)$	۰/۹۰	(MetadataNeuralFuzz) $\beta$

**جدول ۵-۶:** نتایج پوشش کد حاصل از آزمون فازی نرم‌افزار MuPDF با الگوریتم‌های تولید داده آزمون مختلف.

الگوریتم تولید داده آزمون / معیار	پوشش بلوک پایه	درصد	پوشش خطکد	درصد
DataNeuralFuzz	۲۳۷۱۹	۱۹/۳۶	۱۸۶۷۳	۲۰/۸۱
MetadataNeuralFuzz	۲۲۵۸۳	۱۸/۴۳	۱۷۸۹۴	۱۹/۹۵
[۱۱] SampleFuzz	۲۰۹۵۷	۱۷/۱۰	۱۶۷۹۳	۱۸/۷۲
[۲۰] RandomFuzz (FileFuzz)	۷۵۶۳	۶/۱۷	۵۰۰۲	۵/۵۸

**جدول ۵-۷:** میزان بهبود پوشش کد ابزار MuPDF توسط الگوریتم‌های روش پیشنهادی در مقایسه با کارهای مرتبط. اعداد داخل جدول به صورت درصد هستند. مقادیر هر خانه برابر اختلاف پوشش کد حاصله از الگوریتم‌های ستون و سطر مربوط به آن مقدار است.

روش پیشنهادی		
MetaDataNeuralFuzz	DataNeuralFuzz	روش‌های موجود
+۱/۸۳	+۲/۲۶	[۱۱] SampleFuzz
+۶/۸۰	+۷/۷۳	[۲۰] AFL
+۶/۶۳	+۷/۵۶	[۲۰] AugmentedAFL
+۱۲/۲۶	+۱۳/۱۹	[۲۰] RandomFuzz (FileFuzz)

پیشنهادی در فاز کردن این فایل‌ها دارد. پوشش کد حاصل شده در این آزمایش‌ها همچنین از پوشش کد گزارش شده در [۲۰] که حاصل از آزمون فازی MuPDF با AFL و AFL افزوده بیشتر است. اعداد مربوط به آنها قبلاً در بخش ۱-۲-۳ ذکر شدند. جدول ۵-۷ در این بخش نیز اختلاف این مقادیر را با مقادیر حاصل از پوشش کد روش پیشنهادی نشان داده است.

- بهوضوح می‌توان برتری روش‌های هوشمند تولید داده آزمون را نسبت به روش‌های تصادفی مشاهده کرد. همانطور که در ابتدای این پایان‌نامه گفتیم روش‌های تصادفی در ساختارهای پیچیده پوشش کد بسیار کمی را نتیجه می‌دهند. پوشش کد الگوریتم فاز عصبی داده بیش از ۳ برابر الگوریتم فاز تصادفی است.
- باوجود استفاده از هوش مصنوعی و الگوریتم‌های هوشمند در تولید داده آزمون همچنان شاهد پوشش کد پایینی (کمتر از ۲۰ درصد) در پایان عملیات آزمون هستیم. بهنظر می‌رسد رسیدن به پوشش کد بالا در ساختارهای پیچیده، با آزمون فازی جعبه سیاه کار دشواری باشد. از طرفی فنون آزمون جعبه سفید مانند روش‌های اجرای نمادین نیز روی این ساختارها به علت پیچیدگی بالا وجود محدودیت‌های فراوان سخت، زمانبر و تا حد زیادی غیرممکن است و کماکان آزمون فازی مؤثرتر بوده است.

### خطاهای و آسیب‌پذیری‌های شناسایی شده

در بررسی گزارش‌های تولید شده توسط Application Verifier پس از هر آزمون، هیچ‌گونه خطای مشاهده نکردیم. با توجه به اینکه ما نسخه نهایی نرمافزار MuPDF را مورد آزمایش قرار دادیم، تصور می‌شود که بیشتر خطاهای آن در نسخه‌های آزمایشی برطرف شده باشد و در نتیجه پیدا کردن خطای جدید سخت خواهد بود. از طرفی MuPDF نرمافزاری تحت توسعه فعال و جامعه توسعه‌دهنده و کاربری بزرگی است که سبب می‌شود تا از کیفیت خوبی برخوردار باشد. با این حال الگوریتم فاز عصبی داده چندین مورد استفاده از توابع نامن را شناسایی کرد که Application Verifier آنها را در قالب هشدارهای امنیتی اعلام کرده است.

پیش از آزمایش‌های اصلی ما فازر و Application Verifier روی نرمافزارهای کوچکی که خطای آنها مشخص بود اجرا کردیم. بهنظر می‌رسد Application Verifier روی ویندوز ۱۰ نسخه ۶۴ بیتی قادر به تشخیص خطاهای حافظه برنامه‌های ۳۲ بیتی نیست؛ زیرا موفق به شناسایی این خطاهای نشدمیم. برای برنامه‌های ۶۴ بیتی اما این مشکل وجود نداشت. لذا ما هر دو نسخه ۳۲ و ۶۴ بیتی MuPDF را مورد آزمون قرار دادیم. فازر، برنامه PDF‌خوان را با داده آزمون ورودی باز و بعد از ۱۰ ثانیه آن را بسته و برای تزریق داده آزمون بعدی اقدام می‌کند. همزمان پوشش کد و گزارش وضعیت حافظه نیز ثبت و ذخیره می‌شوند. بنابراین زمان

آزمون برای هر مجموعه ۱۰۰۰۰۰ فایلی حدود ۲۸ ساعت به طول انجامید<sup>۱</sup>. برای همه نرم افزارهای دیگر آزمون به این شیوه قابل انجام خواهد بود. گفتیم که آزمون فازی نوعی آزمون فشار است. در نظر داریم تا آزمون فازی با این روش را با استفاده از ۱۰۰ هزار فایل تولید شده و حتی بیشتر انجام دهیم. در این صورت امکان سقوط برنامه MuPDF افزایش می یابد.

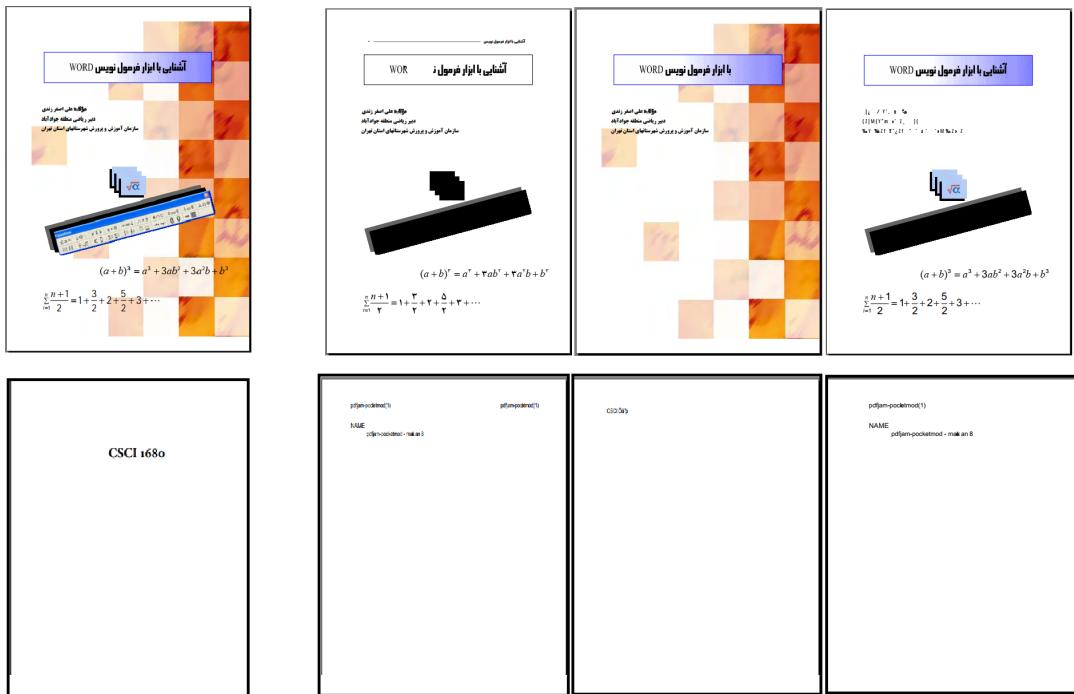
### تنوع داده های تولید شده

مدل های ما در روش پیشنهادی قادر به تولید داده های متنوع و به صورت کنترل شده بدشکل هستند. الگوریتم فاز عصبی داده به خوبی در تغییر محتویات فایل های PDF عمل کرده و می تواند فایل هایی با داده های جدید بدون تغییر در ساختار ایجاد کند. شکل ۵-۶ نمونه ای از داده های تولید شده توسط الگوریتم فاز عصبی داده را نشان می دهد. همان طور که در این شکل پیداست فایل های PDF در عین معتبر بودن حاوی داده های جدیدی هستند که با مقادیر مرزی در فرایند تولید داده آزمون جایگذاری شده اند. به دلیل معتبر بودن تعداد بیشتری از فایل های تولید شده توسط الگوریتم فاز عصبی داده پوشش کد این الگوریتم از پوشش کد الگوریتم فاز عصبی فرآداهه بیشتر است. اما هر دو الگوریتم مورد نیاز هستند؛ زیرا، هر کدام مرحله مجازی از مراحل پردازش فایل در کد برنامه را هدف آزمون قرار می دهنند.

## ۵-۵ خلاصه

در این فصل، ابتدا پارامترهای قابل بررسی و مؤثر در روش های تولید داده آزمون مبتنی بر یادگیری ژرف را معرفی و سپس روش پیشنهادی خود را که در فصل ۴ مطرح کرده بودیم، روی نرم افزار MuPDF و برای مهمترین این پارامترها مورد آزمایش و ارزیابی قرار دادیم. نتایج در حالت کلی حاکی از بهبود میزان پوشش کد در آزمون فازی MuPDF هنگام استفاده از مدل های و الگوریتم های تولید داده آزمون پیشنهاد شده است. به خصوص نسبت به تعدادی از روش های قبلی از جمله [۱۱] که مشابه ترین کار مرتبط بود، آمار و ارقام بهتری هم در دقت مدل ها و هم در پوشش کد مشاهده می شود. علاوه بر این نتیجه گیری کلی آزمایش های مختلف ما چندین واقعیت تجربی دیگر را مشخص کرد که مهمترین آنها عبارتند از:

<sup>۱</sup> Application Verifier و ابزار گذاری کد سرباره ای را به زمان هربار اجرای برنامه اضافه می کنند. همچنین فاصله زمانی کوتاهی بین تزریق داده های آزمون متوالی در نظر گرفته شد؛ زیرا، در مواردی سقوط برنامه ناشی از خطای دیگر بخش های سیستم است. با لحاظ این زمان ها از سقوط های این چنینی با اطمینان زیادی جلوگیری می نماییم.



**شکل ۵-۶:** نمونه‌ای از داده‌های آزمون متنوع تولید شده توسط الگوریتم فاز عصبی داده در فرایند آزمون فازی قالب فایل PDF. در هر ردیف، سمت چپ‌ترین فایل، فایل میزبان را نشان می‌دهد و ۳ فایل سمت راست فایل‌های حاصل از مدل مولد هستند. تغییر در محظوا به وضوح قابل مشاهده است.

LSTM دوسویه به عنوان مدل زبانی می‌تواند به دقت بیشتر و خطای کمتری روی مجموعه داده دست پیدا کند. با این حال مدل‌های ژرف ساده مانند LSTM یکسویه بدون لایه‌های Dropout توانستند مدل‌های پیچیده‌تری مثل LSTM دوسویه را در آزمون فازی شکست دهند. نتیجه‌گیری مشابهی در [۲۰] بیان شده است.

- فایل‌های PDF‌ای که پوشش کد بیشتری دارند، هنگام تغییر نحوه بروزرسانی افزایشی نیز، اختلاف پوشش کد بیشتری نسبت به دیگر فایل‌ها فراهم می‌کنند.
- تنوع تولید داده ۱ برای بیشتر مدل‌ها منجر به پوشش کد بهتری می‌شود.
- افزایش دوره‌های آموزش لزوماً به افزایش پوشش کدنمی انجامد اما تا زمانی که خطای مدل‌ها را به نحو خوبی کاهش دهد، پوشش کد را افزایش می‌دهد.
- روش‌های ترکیبی تولید داده آزمون مانند الگوریتم‌های فاز عصبی داده و فاز عصبی فراداده، که بخش‌های دودویی را نیز در فرایند آزمون فازی شرکت می‌دهند منجر به پوشش کد بیشتری می‌شوند.

● آزمون فازی با روش‌های تولید داده هوشمند، آزمون فازی تصادفی را همواره شکست می‌دهد ولی هنوز هم روی ساختارهای خیلی پیچیده به پوشش کد آنچنان بالایی منتهی نمی‌گردد، این امر ارزشمند بودن کار بر روی روش‌های تولید داده آزمون در آینده را نشان می‌دهد. البته دلیل پوشش کد پایین برای نرم‌افزار MuPDF بیشتر آن است که این نرم‌افزار قالب‌های فایل دیگری غیر از PDF را پشتیبانی می‌کند. لذا اجرای آن با فایل‌های PDF تنها کدهای مربوط به تجزیه و پرداخت PDF را سبب می‌شود. بنابراین باقیستی قالب‌های دیگر را نیز در فرایند آزمون شرکت داد که بدون شک منجر به افزایش پوشش کد می‌شود.

اثر تعدادی از پارامترهای دیگر مطرح شده در جدول ۱-۵ مانند راهبردهای تولید داده از مدل، در این آزمایش‌ها بررسی نشدند که جای دارد آزمایش‌هایی برای آنها نیز طراحی کرد. با این حال با توجه به محدودیت‌های که برای هریک از دیگر راهبردها مثل راهبرد حریصانه برشمرده شد، راهبرد نمونه‌برداری مناسب‌ترین راهبرد به نظر می‌رسد.

## فصل ۶

### نتیجه‌گیری و کارهای آتی

«ما ممکن است امیدوار باشیم که ماشین‌ها در نهایت در همه زمینه‌های هوشمند با انسان رقابت کنند، اما بهترین زمینه برای شروع کدام است؟!»

---

#### آلن تورینگ

### ۶-۱ نتیجه‌گیری

دکتر اندر وان جی<sup>۱</sup> هوش مصنوعی را یک الکترونیستیه جدید می‌نامد که می‌تواند تحول بزرگ بعدی را در صنعت رقم بزند. یادگیری ژرف و نگاه متفاوت آن به حل مسئله تا همین الان این تحول بزرگ را در وظیفه‌هایی مانند پردازش تصویر، پردازش صوت، پردازش متن و ترجمه ماشینی رقم زده است. سال ۲۰۱۷ آغاز استفاده از یادگیری ماشینی در آزمون فازی و تولید داده آزمون بود [۱۰، ۲۰]. پژوهشگران مایکروسافت برای نخستین بار از این فنون در آزمون فازی استفاده کردند. در این پایان‌نامه ما از مدل‌های یادگیری ژرف برای یادگیری ساختار فایل‌های پیچیده و سپس تولید داده آزمون جدید به منظور استفاده در فرایند آزمون فازی استفاده کردیم. به طور خاص هر فایل را می‌توان نمونه‌ای مشتق شده از زبان یا گرامر ساختار آن دانست. براین اساس ما با استفاده از شبکه‌های عصبی مکرر اقدام به ایجاد یک مدل زبانی عصبی برای هر ساختار فایلی می‌کنیم که یک توزیع احتمالی از چگونگی وقوع نشانه‌ها در یک فایل را با آموزش روی یک مجموعه داده تخمین می‌زنند. سپس این

---

<sup>۱</sup> Andrew Ng (<http://www.andrewng.org/>)

مدل زبانی را برای تولید فایل‌های جدید به کار می‌بندیم. یک ویژگی مفید امکان تشخیص و تمایز بین داده و فراداده در یک فایل با استفاده از چنین مدل‌هایی است در نتیجه می‌توان جابه‌جایی آزمون فازی را با استناد به تفکیک داده و فراداده به نحو هوشمندتری انجام داد. دو الگوریتم پیشنهادی در این راستا نشان دادند چنین روشی قادر به اجرای بخش‌های بیشتری از کد یک SUT است و پوشش کد بیشتری را رقم می‌زند که در نتیجه امکان شناسایی خطای نیز افزایش خواهد یافت.

در فصل ۱، مسائلی را در باب ساخت بودن تولید داده آزمون برای رسیدن به پوشش کد بالا در آزمون فازی قالب‌های فایل با ساختار پیچیده مثل PDF مطرح کردیم و دیدیم که چنان‌چه بتوان با استفاده از گرامر رودی داده‌های آزمون را تولید کرد، تعداد داده‌هایی که در مراحل اولیه توسط کدهای مدیریت استثنای تجزیه‌گر رد می‌شوند کاهش یافته و قادر به نفوذ به مسیرهای عمیق‌تر برنامه خواهیم بود. با این ایده و براساس مطالعات اولیه و کارهای قبلی که در فصل‌های ۲ و ۳ به آنها اشاره کردیم، در فصل ۴ یک روش یادگیری ساختار فایل را پیشنهاد دادیم که تولید داده آزمون مبتنی بر گرامر را خودکار می‌کند. در فصل ۵ هم ابتدا پارامترهایی را که در تولید خودکار داده آزمون از روی مدل‌های یادگیری ژرف، نقش دارند شناسایی و سپس آزمایش‌هایی را برای بررسی و مقایسه تأثیر هریک از این پارامترهای طرح‌ریزی، پیاده‌سازی و اجرا نمودیم.

یک نتیجه‌گیری مهم که آزمایش‌های ما نشان می‌دهند این است که افزایش ظرفیت محاسباتی و پیچیدگی شبکه‌های عصبی ژرف که در نتیجه قدرت حل مسئله یک شبکه را بالا می‌برد لزوماً منجر به نتایج بهتری در آزمون فازی نخواهد شد و مشاهده کردیم که مدل‌های ساده مدل‌های پیچیده‌تر را در پوشش کد SUT شکست می‌دهند. این مدل‌ها طبیعتاً به زمان کمتری برای آموزش نیاز داشته و تولید داده با استفاده از آنها نیز سریع‌تر خواهد بود. بنابراین از همه نظر مقررین به صرفه هستند.

روش پیشنهادی در این پایان‌نامه، در عین مزایایی که برای آن بر شمرده شد محدودیت‌ها و نقاط ضعفی دارد. بزرگترین محدودیت را می‌توان عدم اطلاع مدل مولد از وضعیت SUT دانست. به عبارت بهتر مدل مولد در غیاب SUT و تنها روی مجموعه داده‌های یک قالب فایل آموزش دیده و سپس اقدام به تولید داده‌های آزمون می‌کند. فازر پیشنهادی نیز فاقد یک حلقه بازخورد برای دریافت اطلاعات زمان اجرای SUT است. این در حالی است که همواره بازخوردهای اجرای SUT می‌تواند حاوی اطلاعات خوبی برای پیش‌برد ادامه فرایند آزمون در اختیار فازر قرار دهد. این محدودیت می‌تواند به عنوان کارآتی مورد بررسی قرار گیرد. مشکل بعدی که البته مختص به روش پیشنهادی مانوده و هر روش یادگیری متشابه در این زمینه با آن مواجه است نیاز به تعدادی زیادی فایل به عنوان مجموعه داده است. برای قالب‌های مشهور فایل مانند PDF که مورد مطالعاتی این پایان‌نامه بود، این مشکل پُررنگ نیست اما اگر برای یک قالب خاص امکان تهیه مجموعه داده بزرگی نباشد، استفاده از این روش تقریباً غیر ممکن خواهد بود.

در هر حال تلاش در راستای موضوع این پایان‌نامه صرف نظر از نتایج تجربی آن به دلیل پیوند دو شاخه

به ظاهر کمتر مرتبط در علم کامپیوتر یعنی یادگیری ژرف و آزمون فازی ارزشمند به نظر می‌رسد. به ویژه که طراحی و آموزش شبکه‌های عصبی ژرف کاری مهیّج بوده و این مسیر پژوهشی نیز در ابتدای راه خود است. در این بین دو حوزه مذکور که این پایان‌نامه بر آنها بیان شده، نیز هر کدام در حالت کلی دارای مزایا و معایب هستند که ممکن است برخی از آنها را قبلًا هم ذکر کرده باشیم، با این حال در پایان این بخش نگاهی کوتاه به مزایا، معایب و آینده هریک از این دو حوزه خواهیم داشت؛ زیرا می‌توانند در تعیین روند پژوهش‌های آتی مؤثر واقع شوند.

## ۶-۱-۱ مزایا و معایب یادگیری ژرف

یادگیری ژرف و به تبع آن شبکه‌های عصبی ژرف در انجام وظایف ساده برای انسان، سخت برای ماشین بسیار موفق ظاهر شده‌اند. با توجه به افزایش قدرت محاسبات انجام حجم وسیعی از محاسبات در مسائل پیچیده، ارزان‌تر از نوشتن یک الگوریتم خاص می‌باشد به‌نحوی که در آینده شاهد افزایش ظرفیت‌های سخت‌افزاری برای توسعه چنین مدل‌هایی در مقیاس‌های بسیار بزرگ خواهیم بود. دو ویژگی بسیار مهم این شبکه‌ها عبارتند از تعمیم‌پذیر<sup>۱</sup> و تطبیق‌پذیر<sup>۲</sup> [۳۸]. تعمیم‌پذیری بدین معنی است که در صورت آموزش صحیح، شبکه برای ورودی‌های جدید نیز درست کار خواهد کرد. تطبیق‌پذیری یعنی در صورتی که داده‌ها تغییر کند، شبکه هم توانایی تغییر خواهد داشت. یعنی یک مدل می‌تواند برای حل خانواده‌ای از مسائل مشابه طراحی و ساخته شود و هر بار با داده‌های مختلفی آموزش ببیند.

شبکه‌های عصبی سراسر فایده و نوش‌داروی حوزه محاسبات جدید نیستند و در عین حال معایبی دارند. از جمله اینکه آموزش آنها سخت و بسیار مستعد خطا است. در واقع دقت نتایج بستگی زیادی به مجموعه آموزش دارد؛ به نحوی که یک مجموعه آموزش ضعیف (کوچک یا نادرست) عملاً شبکه غیرقابل استفاده‌ای را نتیجه می‌دهد. دیگر آنکه قوانین مشخصی برای طراحی یک شبکه جهت کاربردی خاص وجود ندارد. به عبارت بهتر تعیین ابرپارامترها مسئله تصمیم‌نایذیر است؛ یعنی یک نمی‌توان به صورت الگوریتمی بهترین مجموعه ابرپارامتر را برای یک شبکه در یک وظیفه خاص تعیین کرد. این کار معمولاً با سعی و خطا انجام می‌شود و بلآخره اینکه نمی‌توان به فیزیک یا قانون حاکم بر مسئله حل شده توسط شبکه پی برد و تنها با مشتی اعداد سروکار خواهیم داشت که روی یک مجموعه یک هدف خواسته شده را بهینه کرده‌اند.

<sup>1</sup>Generalization

<sup>2</sup>Adaptive

## ۲-۱-۶ مزایا و معایب آزمون فازی

آزمون فازی چندین سودمندی دارد. در درجه نخست سادگی و راحتی خودکارسازی فرایند شرح داده شده است. به همین دلیل، فازرهای بسیاری توسعه داده شده است. برای استفاده از فازرهای موجود تنها انتخاب SUT و فراهم ساختن تعدادی داده آزمون اولیه یا قالب ورودی لازم است. فازر می‌تواند به صورت یک پردازه پس زمینه<sup>۱</sup> و بدون دخالت اضافی کاربر، در یک حلقه بی‌نهایت، به مدت طولانی اجرا شود. برای ساخت یک فازر جدید نیز کافی است پیمانه‌های شکل ۲-۲، توسعه داده شده و در کنار هم قرار گیرند.

آزمون فازی همچنین کاستی‌های آزمون معمولی که به صورت دستی صورت می‌پذیرد را جبران می‌کند. آزمون‌های نوشته شده به صورت دستی تا حدودی تمایل به پیش‌دانسته‌های ذهنی فرد آزمون‌گر در مورد کد دارند. آزمون فازی از انحراف یاد شده مستثنی است و می‌تواند ورودی‌های بدشکل را به نحوی تولید کند که پیش از آن هیچگاه، به ذهن فرد یا افراد آزمون‌گر نرسیده است. با گذشت سه دهه از ابداع آزمون فازی این روش جایگاه ویژه‌ای در صنعت و نیز در پژوهش یافته است و به همین دلیل کار بر روی آن ارزشمند و اثر بخش است.

در حالی که آزمون فازی در پیدا کردن خطاهای فساد حافظه، بسیار خوب عمل می‌کند، خطاهای پیچیده‌تر مانند خطاهای منطقی به ندرت توسط این آزمون قابل شناسایی هستند؛ چرا که آشکارسازی آنها نیازمند به اتمام رسیدن اجرای برنامه بدون خطای حافظه و داشتن سروش<sup>۲</sup> آزمون است. افزون بر این تحریک برخی خطاهای حافظه برای فازرهای بسیار سخت خواهد بود. مسئله انفجار مسیر که ناشی از وجود حلقه‌های تکرار و شرط‌های تودرتو است مانع از اجرای نمادین برنامه‌های پیچیده می‌شود. در نتیجه به کارگیری اجرای نمادین در آزمون فازی جعبه سفید در بسیاری موارد ممکن نیست و این مسئله به پوشش کد پایین و خوب آزمون نشدن برنامه می‌انجامد.

آزمون فازی ممکن است برای اهداف سوء مورد استفاده قرار گیرد. یعنی شناسایی آسیب‌پذیری‌ها توسط فرد مهاجم و بهره‌برداری از آنها جهت حمله به یک سیستم نرم‌افزاری. در نتیجه روش‌هایی برای مقابله با امکان آزمون‌پذیری یک نرم‌افزار مطرح شده‌اند که می‌توان آنها را در طبقه روش‌های ضد مهندسی معکوس دانست. از جمله این روش‌ها می‌توان به حوزه‌ای جدید که اخیراً تحت عنوان ضد فازینگ<sup>۳</sup> مطرح شده است، اشاره کرد که در آن اقداماتی برای کندسازی یا به‌کلی مانع شدن کشف خطاهای توسط آزمون فازی انجام می‌گیرد. کاهش کارآمدی و پوشش خطاهای دو رویکرد از میان رویکردهای موجود در این زمینه هستند. دیگر فنون مقابله با مهندسی معکوس نظیر مبهم‌سازی کد را نیز می‌توان در ضد فازینگ به کار گرفت. چنان‌چه این رویکردها

<sup>1</sup>Background Process

<sup>2</sup>Oracle

<sup>3</sup>Anti-fuzzing

به بلوغ خوبی برسند و توسعه دهنده‌گان آنها را در برنامه‌های خود لحاظ نمایند، در آینده بایستی به دنبال فنون جایگزین برای روش آزمون فازی جعبه خاکستری و به طور کلی آزمون فازی باشیم. البته استفاده از این فن آزمون در بین توسعه دهنده‌گان و پیش از انتشار نسخه نهایی نرمافزار همچنان در صنعت ادامه خواهد یافت.

## ۶-۲ نوآوری‌ها

مجموعه نوآوری‌ها، دستاوردها و محصولات کار پژوهشی ما در قالب این پایان‌نامه عبارت است از:

۱. خودکارسازی فرایند یادگیری ساختار فایل ورودی و تولید داده‌های آزمون برای آزمون فازی قالب فایل با به کارگیری روش‌های جدید یادگیری ژرف.
  ۲. بهبود پوشش کد SUT با به کار بست روشنی ترکیبی یعنی تولید فراداده و داده‌های متنی، مبتنی بر گرامر و سپس تزریق داده‌های دودویی مبتنی بر جایه‌جایی تصادفی در مکان‌های تعیین شده.
  ۳. شناسایی و استخراج پارامترهای مؤثر در تولید داده آزمون به روش یادگیری ژرف و ارزیابی تأثیر آنها با ارایه مجموعه‌ای از آزمایش‌های کنترل شده.
  ۴. معرفی یک مجموعه دانه اولیه و یک مجموعه داده آزمون برای آزمون فازی قالب فایل PDF تحت یک پیکره از فایل‌های PDF به همراه پوشش کدهای آنها.
  ۵. طراحی و پیاده‌سازی یک فازر قالب فایل ساده با معماری کاملاً پیمانه‌ای و قابل حمل، مجهز به پیمانه تولید خودکار داده‌های آزمون با استفاده از مدل‌های مولد.
- به طور خلاصه در این پایان‌نامه یک مجموعه داده، چهار مدل مولد، دو الگوریتم فاز و یک فازر قالب فایل پیشنهاد و معرفی گردید که همه آنها در قالب یک بسته نرمافزاری تحت عنوان IUST Deep Fuzz منتشر شده‌اند. از این محصول می‌توان در عمل برای آزمون فازی و شناسایی خطاهای نرمافزارهایی با ورودی فایل استفاده کرد. تنظیمات کنونی بر روی قالب فایل PDF تعیین شده است اما به راحتی قابل تغییر است. نرمافزارهای با ورودی فایل علاوه بر PDF‌خوان‌ها، شامل خانواده وسیع کامپایلرها، مرورگرهای وب، محیط‌های توسعه مجتمع و غیره می‌شوند، که همه آنها در طبقه برنامه‌های کاربردی و بسیار مهم قرار می‌گیرند. تقریباً در همه موارد مذکور، داده کافی برای ایجاد مدل مولد وجود دارد. به عنوان مثال در آزمون مرورگرها، ایجاد یک مدل مولد برای تولید فایل‌های ترکیبی HTML، CSS و JavaScript داده آزمون بهتری نسبت به تولید تنها یکی از این فایل‌ها [۴۸] فراهم می‌کند و از طرفی برای هر سه قالب فایل نامبرده مجموعه داده به فراوانی یافت می‌شود.

## ۳-۶ ملاحظات اعتبارسنجی

در این پایان‌نامه بر بهبود معیار پوشش کد در آزمون فازی، به‌طور مکرر تأکید ورزیدیم. منظور از پوشش کد در اینجا پوشش دستور (یا در حالت کلی تر پوشش بلوک پایه است). در مواردی به پوشش مسیر نیز اشاره کردیم و به عنوان مثال در مورد پوشش مسیرهای اجرایی عمیق برنامه صحبت به میان آوردهیم. ذکر این نکته ضروری است که معیارهای پوشش دستور و پوشش مسیر متفاوت هستند و همچنان‌که در فصل ۲ دیدیم، پوشش مسیر معیار سخت‌گیرانه‌تری نسبت به پوشش دستور است؛ یعنی ممکن است تمامی دستورات برنامه در یک یا چند اجرا، حداقل یک‌بار اجرا شوند ولی لزوماً تمامی مسیرهای اجرایی پوشش داده نشوند.

یک برنامه با یک دستور  $\text{if}$  ساده را در نظر بگیرید. اگر برنامه در حالتی اجرا شود که حاصل ارزیابی عبارت شرطی درست شود، همه دستورات (همچنین همه بلوک‌های پایه) برنامه اجرا می‌شوند. در این حالت پوشش بلوک پایه معادل ۱۰۰ درصد خواهد بود. در حالی که این برنامه در بَدَوی‌ترین شکل خود، دو مسیر اجرایی دارد: یک مسیر که از بدنۀ دستور شرطی  $\text{if}$  عبور می‌کند و مسیر دیگر که وارد بدنۀ دستور  $\text{if}$  نمی‌شود. برای سناریوی اجرای ذکر شده، پوشش مسیر ۵۰ درصد است. بنابراین تفاوت معناداری میان پوشش دستور و پوشش مسیر در این مثال وجود دارد. ممکن است خواننده با این ابهام رو به رو شود که در چنین حالتی، نتایج گزارش شده موردنی بوده و قابل تعمیم نیست.

چنان‌چه یک دستور (بلوک پایه) جدید اجرا شود، می‌توان گفت یک مسیر جدید اجرا شده است. بنابراین از این منظر، تلاش برای افزایش پوشش بلوک پایه منجر به افزایش پوشش مسیر نیز می‌گردد. اما دو مجموعه برابر از پوشش‌های دستور، لزوماً پوشش مسیر برابری به دست نمی‌دهند. اندازه‌گیری پوشش مسیر دشوارتر بوده و سربار بیشتری به آزمون تحمیل می‌کند. از لحاظ تئوری نیز، برخی از مسیرهای ایستای برنامه، غیرقابل دسترسی هستند و با وجود حلقه‌های تکرار در برنامه، تعداد مسیرهای اجرایی در مواردی بی‌نهایت می‌شود. به سبب این‌گونه مسائل، ابزارهای معرفی شده در فصل ۲، هیچ‌کدام پوشش مسیر اجرایی را اندازه‌گیری نمی‌کنند. در بیشتر کارهای مربوط به آزمون فازی منظور از پوشش کد، همان پوشش در سطح دستورات برنامه است. بنابراین آمار و ارقام ارایه شده در این پایان‌نامه نیز مشابه کارهای پیشین بر حسب همان میزان پوشش دستورات و در حالت کلی تر پوشش بلوک پایه بنا شده است.

در آزمایش‌هایی که پوشش دستور دو مجموعه داده آزمون، دقیقاً برابر باشد، برای مقایسه لازم است تا پوشش مسیر اندازه‌گیری گردد. در آزمایش‌های انجام شده توسط ما، همواره پوشش بلوک پایه مجموعه‌های آزمون استفاده شده پس از گرفتن اجتماع، متفاوت بوده است که نشان از متفاوت بودن پوشش مسیرها نیز دارد. لذا نتایج گزارش شده، صحت داشته و دلالت بر بهبود میزان پوشش کد SUT در حالت کلی دارد. در هر صورت، ما در نظر داریم تا آزمایش‌های خود را در مقیاس بسیار بزرگ‌تری و روی قالب‌های فایل مختلف،

تکرار کرده و اندازه‌گیری درست پوشش مسیر را نیز برای آزمایش‌های جدید انجام دهیم.

## ۶-۴ کارهای آتی

پایان‌نامه پیش‌رو، مباحث تولید داده آزمون، آزمون فازی و یادگیری ژرف (و به طور خاص‌تر مدل‌های زبانی عصبی) را به یکدیگر پیوند زده است. هر سه موضوع یاد شده از موضوعات مهم، کاربردی و داغ در پژوهش‌های علوم و مهندسی کامپیوتر هستند. پیشنهادهای زیادی برای کارهای آتی مرتبط با موضوع این پایان‌نامه مطرح است که در ذیل به چندین مورد از آنها اشاره می‌کنیم.

۱. استفاده از دیگر مدل‌ها و معماهای شبکه‌های عصبی ژرف در تولید داده آزمون. به عنوان مثال می‌توان از مدل‌های GAN<sup>۱</sup> [۵۳] برای تولید داده آزمون استفاده کرد. به طور خلاصه GAN از دو شبکه عصبی تشکیل شده است. یک شبکه مولد که داده‌های جدید را تولید می‌کند و یک شبکه که آنها ارزیابی می‌کند. هدف شبکه مولد این است که داده‌هایی تولید که از دید شبکه ارزیاب خطای کمتری داشته باشند. از GAN در تولید محیط‌های جدید در بازی‌های رایانه‌ای استفاده شده است اما کاربرد آنها محدود به این مورد نیست.

۲. استفاده از یادگیری ژرف در دیگر فازرها. به عنوان نمونه یادگیری ساختار پروتکل‌های شبکه که می‌تواند برای تولید داده در فازرها شبکه و پروتکل‌ها استفاده شود؛ این مسئله بمویژه در آزمون پروتکل‌هایی با ساختار ناشناخته مثل باتنت‌ها قابل توجه است. در این مورد یادگیری ممکن است به اهداف مهندسی معکوس کمک نماید.

۳. تولید داده آزمون براساس اهداف مختلف. هدف آزمون فازی و فازر نبایست لزوماً افزایش پوشش کد یا به عبارتی Input Gain تعیین شود. داده‌های آزمونی که توابع نامن را فراخوانی می‌کنند برای مثال حائز اهمیت هستند [۴۹]. ایجاد مدل‌های یادگیری که امتیاز را به فراخوانی مجموعه‌ای از توابع نامن نسبت دهند، به این امید که آزمون برنامه با ورودی‌هایی که این مسیرها را اجرا می‌کند، منجر به وقوع خطای حافظه می‌شود، می‌تواند جالب و قابل توجه باشد.

۴. افزودن حلقه بازخورد به فازر پیشنهادی در این پایان‌نامه. قبل نیز اشاره شد که یک محدودیت روش پیشنهادی عدم استفاده از حلقه بازخورد است. می‌توان این حلقه را در قالب یک فازر جدید اضافه

<sup>1</sup>Generative Adversarial Network

کرد یا برای مثال از روش پیشنهادی برای تولید دانه‌های اولیه برای فازرهایی مثل AFL استفاده کرد که در این صورت بایستی یک مرحله کمینه‌سازی دانه<sup>۱</sup> نیز برای افزایش کارایی AFL ارایه شود.

۵. افزودن تعامل کاربر به آزمون فازی جعبه سیاه. تعامل کاربر با برنامه منجر به اجرای کدهای مربوط به کارهای کاربر می‌شود. ما در برخی آزمایش‌های خود مشاهده کردیم که این امر تأثیر زیادی بر میزان پوشش کد دارد. البته این مورد با آزمون فازی واسط کاربر متفاوت است؛ زیرا، لزوماً همه تعامل‌ها مربوط به واسط کاربر نیستند و بعضی از آنها کدهای مربوط به مرحله پرداخت فایل را اجرا می‌کنند. تعامل کاربر نیازمند دخالت مستقیم کاربر و انجام عملیاتی از طریق صفحه‌کلید یا دیگر ابزارهای ورودی است که در نتیجه خودکار نیست. بنابراین می‌توان فازی طراحی کرد که برروی یک داده آزمون ورودی تولید شده یک دنباله از عملیات کاربر را به‌طور خودکار و به‌صورت ترتیبی یا تصادفی انجام دهد. ما این مورد را در کار بعدی خود لحاظ می‌کنیم.

۶. استفاده از یادگیری ژرف در دیگر مراحل آزمون نرم‌افزار. آزمون نرم‌افزار تنها مختص به مرحله تولید داده آزمون و مکان‌یابی خطای نیست. می‌توان در دیگر مراحل نیز از فنون یادگیری ژرف استفاده کرد. یک زمینه پژوهشی نو، ترمیم خودکار برنامه‌ها پس از مشخص شدن مکان خطای است. برای این منظور استفاده از مدل CAN<sup>2</sup> [۵۴] ایده جالبی به‌نظر می‌رسد. CAN نوع خاصی از GAN است که تلاش می‌کند عوامل خلاقیت و تولید داده جدید را به آن اضافه کند. در ترمیم خودکار برنامه نیز نیازمند سازوکاری برای تنوع‌بخشی به قسمت‌های خطدار باهدف ترمیم آن قسمت‌ها، هستیم.



<sup>1</sup>Seed Minimization

<sup>2</sup>Creative Adversarial Networks

## مراجع

- [1] A. Takanen, J. DeMott, and C. Miller. *Fuzzing for software security testing and quality assurance*. Norwood, MA, USA: Artech House, Inc., 1 ed. , 2008.
- [2] M. Sutton, A. Greene, and P. Amini. *Fuzzing: brute force vulnerability discovery*. Addison-Wesley Professional, 2007.
- [3] B. P. Miller, L. Fredriksen, and B. So, “An empirical study of the reliability of Unix utilities,” *Commun. ACM*, vol.33, pp.32–44, Dec. 1990.
- [4] B. P. Miller, D. Koski, C. Pheow, L. V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, “Fuzz revisited: a re-examination of the reliability of Unix utilities and services,” tech. rep., University of Wisconsin-Madison, 1995.
- [5] J. E. Forrester and B. P. Miller, “An empirical study of the robustness of Windows NT applications using random testing,” in *Proceedings of the 4th Conference on USENIX Windows Systems Symposium - Volume 4*, WSS’00, (Berkeley, CA, USA), pp.6–6, USENIX Association, 2000.
- [6] B. P. Miller, G. Cooksey, and F. Moore, “An empirical study of the robustness of MacOS applications using random testing,” in *Proceedings of the 1st International Workshop on Random Testing*, RT ’06, (New York, NY, USA), pp.46–54, ACM, 2006.
- [7] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, “Vuzzer: application-aware evolutionary fuzzing,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, Feb. 2017.
- [8] U. Kargén and N. Shahmehri, “Turning programs against each other: high coverage fuzz-testing using binary-code mutation and dynamic slicing,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (New York, NY, USA), pp.782–792, ACM, 2015.

- 
- [9] S. Veggalam, S. Rawat, I. Haller, and H. Bos, “Ifuzzer: an evolutionary interpreter fuzzer using genetic programming,” in *Computer Security – ESORICS 2016* (I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, eds. ), (Cham), pp.581–601, Springer International Publishing, 2016.
  - [10] P. Godefroid, M. Y. Levin, and D. Molnar, “Sage: whitebox fuzzing for security testing,” *Queue*, vol.10, Jan. 2012.
  - [11] P. Godefroid, H. Peleg, and R. Singh, “Learn&fuzz: machine learning for input fuzzing,” in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, (Piscataway, NJ, USA), pp.50–59, IEEE Press, 2017.
  - [12] A. Kettunen, *Test harness for web browser fuzz testing*. M.Sc. Thesis, University of Oulu, 2014.
  - [13] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2016.
  - [14] N. Rathaus and G. Evron. *Open source fuzzing tools*. Syngress Publishing, 2007.
  - [15] O. Bastani, R. Sharma, A. Aiken, and P. Liang, “Synthesizing program input grammars,” *SIGPLAN Not.*, vol.52, pp.95–110, June 2017.
  - [16] M. Höschele and A. Zeller, “Mining input grammars from dynamic taints,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, (New York, NY, USA), pp.720–725, ACM, 2016.
  - [17] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds. ), pp.3104–3112, Curran Associates, Inc., 2014.
  - [18] K. Cho, B. van Merriënboer, Ç. Gülcühre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *CoRR*, vol.abs/1406.1078, 2014.
  - [19] M. Zalewsky, “American fuzzy lop,” [Online]. Available: <http://lcamtuf.coredump.cx/afl/>, [Accessed: 2017-10-11].
  - [20] M. Rajpal, W. Blum, and R. Singh, “Not all bytes are equal: neural byte sieve for fuzzing,” *CoRR*, vol.abs/1711.04596, 2017.

- 
- [21] E. Dubrova. *Fault-tolerant design*. Springer Publishing Company, Incorporated, 2013.
  - [22] Symantec, “ISTR internet security threat report,” tech. rep., 2018. <https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf>.
  - [23] Microsoft Corporation, “Simplified implementation of the SDL,” tech. rep., 2010.
  - [24] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, “A systematic review of fuzzing techniques,” *Computers and Security*, vol.75, pp.118–137, 2018.
  - [25] Artifex Software, Inc., “MuPDF,” [Online]. Available: <https://mupdf.com/>, [Accessed: 2018-07-25].
  - [۲۶] م. ذاکری نصرابادی، بررسی روش‌های تولید خودکار داده‌های آزمون برای استفاده در فازرهای مبتنی بر قالب فایل. سینیار کارشناسی ارشد، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، آبان ۱۳۹۶.
  - [27] Paul C. Jorgensen. *Software testing a craftsman's approach*, vol.47. CRC Press Taylor & Francis Group, fourth edi ed. , 2014.
  - [28] E. Nikravan and S. Parsa, “A reasoning-based approach to dynamic domain reduction in test data generation,” *International Journal on Software Tools for Technology Transfer*, May 2018.
  - [29] R. McNally, K. Yiu, and D. Grove, “Fuzzing: the state of the art,” *DSTO Defence Science and Technology Organisation*, p.55, 2012.
  - [30] M. E. Khan and F. Khan, “A comparative study of white box, black box and grey box testing techniques,” *International Journal of Advanced Computer Science and Applications*, vol.3, no.6, pp.12–15, 2012.
  - [31] D. L. Bruening, *Efficient, transparent, and comprehensive runtime code manipulation*. Ph.D. thesis, Cambridge, MA, USA, 2004. AAI0807735.
  - [32] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, “Pin: building customized program analysis tools with dynamic instrumentation,” *SIGPLAN Not.*, vol.40, pp.190–200, June 2005.
  - [33] QEMU Team, “QEMU,” [Online]. Available: <https://www.qemu.org/>, [Accessed: 2018-07-27].

- 
- [34] C. Miller and Z. Peterson, “Analysis of mutation and generation-based fuzzing,” *White Paper, Independent Security Evaluators*, pp.1–7, 2007.
  - [35] C. Holler, K. Herzig, and A. Zeller, “Fuzzing with code fragments,” in *Proceedings of the 21st USENIX Conference on Security Symposium*, Security’12, (Berkeley, CA, USA), pp.38–38, USENIX Association, 2012.
  - [36] Microsoft, “Application verifier (appverif.exe),” [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/application-verifier>, [Accessed: 2018-07-18].
  - [37] Network Working Group, “Internet security glossary,” <https://tools.ietf.org/html/rfc2828>, [Accessed: 2017-10-11].
  - [38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
  - [39] D. Jurafsky and J. H. Martin. *Speech and language processing (3rd ed. draft)*. 2017. <https://web.stanford.edu/~jurafsky/slp3/>.
  - [40] A. Karpathy, *Connecting images and natural language*. Ph.D. Thesis, Stanford University, 2016.
  - [41] B. Sautermeister, *Deep learning approaches to predict future frames in videos*. M.Sc. Thesis, Technical University of Munich, 2016.
  - [42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol.15, pp.1929–1958, 2014.
  - [43] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: a search space odyssey,” *CoRR*, vol.abs/1503.04069, 2015.
  - [44] Z. C. Lipton, “A critical review of recurrent neural networks for sequence learning,” *CoRR*, vol.abs/1506.00019, 2015.
  - [45] M. T. Luong, *Neural machine translation*. Ph.D Thesis, Stanford university, 2016.
  - [46] T. Mikolov, *Statistical language models based on neural networks*. Ph.D. Thesis, Brno University of Technology, 2012.

- [47] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: large-scale machine learning on heterogeneous distributed systems,” *CoRR*, vol.abs/1603.04467, 2016.
- [۴۸] س.م. یعقوبی، طراحی و پیاده‌سازی فائزر با هدف تعیین آسیب‌پذیری‌های مروگر وب. پایان‌نامه کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، آبان ۱۳۹۲.
- [۴۹] س. امینی، طراحی و پیاده‌سازی یک روش تولید داده آزمون به منظور کشف آسیب‌پذیری‌های نرم‌افزار. پایان‌نامه کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، آبان ۱۳۹۵.
- [50] F. Chollet *et al.*, “Keras,” <https://keras.io>, 2015.
- [51] D. P. Kingma and J. Ba, “Adam: a method for stochastic optimization,” *CoRR*, vol.abs/1412.6980, 2014.
- [52] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *SIGKDD Explor. Newsl.*, vol.11, pp.10–18, Nov. 2009.
- [53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds. ), pp.2672–2680, Curran Associates, Inc., 2014.
- [54] A. M. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, “CAN: creative adversarial networks, generating ”art” by learning about styles and deviating from style norms,” *CoRR*, vol.abs/1706.07068, 2017.

## پیوست آ

# ساختار فایل PDF

در این پیوست به طور خلاصه ساختار کلی یک فایل PDF را به عنوان یک ساختار فایل پیچیده بیان می‌کنیم. در فصل ۱ اشاره کردیم که ویژگی‌های کامل قالب PDF بسیار زیاد است. بیشتر این ویژگی‌ها، در حدود ۷۰ درصد، در ارتباط با توضیح اشیای داده<sup>۱</sup> و ارتباط آنها بین بخش‌های مختلف یک فایل PDF هست. تمرکز اصلی در این پیوست نیز بر روی ساختار اشیای داده خواهد بود. ما ساختار کلی PDF را در دو بخش فیزیکی و منطقی بررسی خواهیم کرد.

فایل‌های PDF در یک قالب متنی کدگذاری می‌شوند که ممکن است شامل جریان دودویی<sup>۲</sup> مانند تصویر و غیره باشند. یک فایل PDF ترکیبی از دست‌کم یک بدن<sup>۳</sup> است. یک بدن از سه بخش تشکیل شده است: شیء<sup>۴</sup> (obj)، جدول ارجاع متقابل<sup>۵</sup> (xref) و Trailer. در ابتدای هر فایل یک سرآیند قرار می‌گیرد که با عبارت %PDF آغاز شده و در ادامه آن یک عدد که نگارش قالب PDF را مشخص می‌کند، آمده است. شکل آ-۱ یک فایل PDF شامل متن Hello World را که در یک ویراستار متنی باز شده است، نشان می‌دهد. در ادامه این ساختار را بررسی می‌کنیم.

## آ-۱ ساختار فیزیکی

جزئیات ساختار فیزیکی بدن فایل PDF به شرح زیر است:

- اشیاء. داده و فراداده در پروندهای PDF در یک واحد اولیه که شیء نامیده می‌شود سازماندهی می‌شوند. اشیا همگی قالب مشابهی دارند که در شکل آ-۱ مشخص است و همچنین یک ساختار بیرونی مشترک هم دارند. اولین خط یک شیء شناسه آن است که برای ارجاع‌های غیر مستقیم استفاده

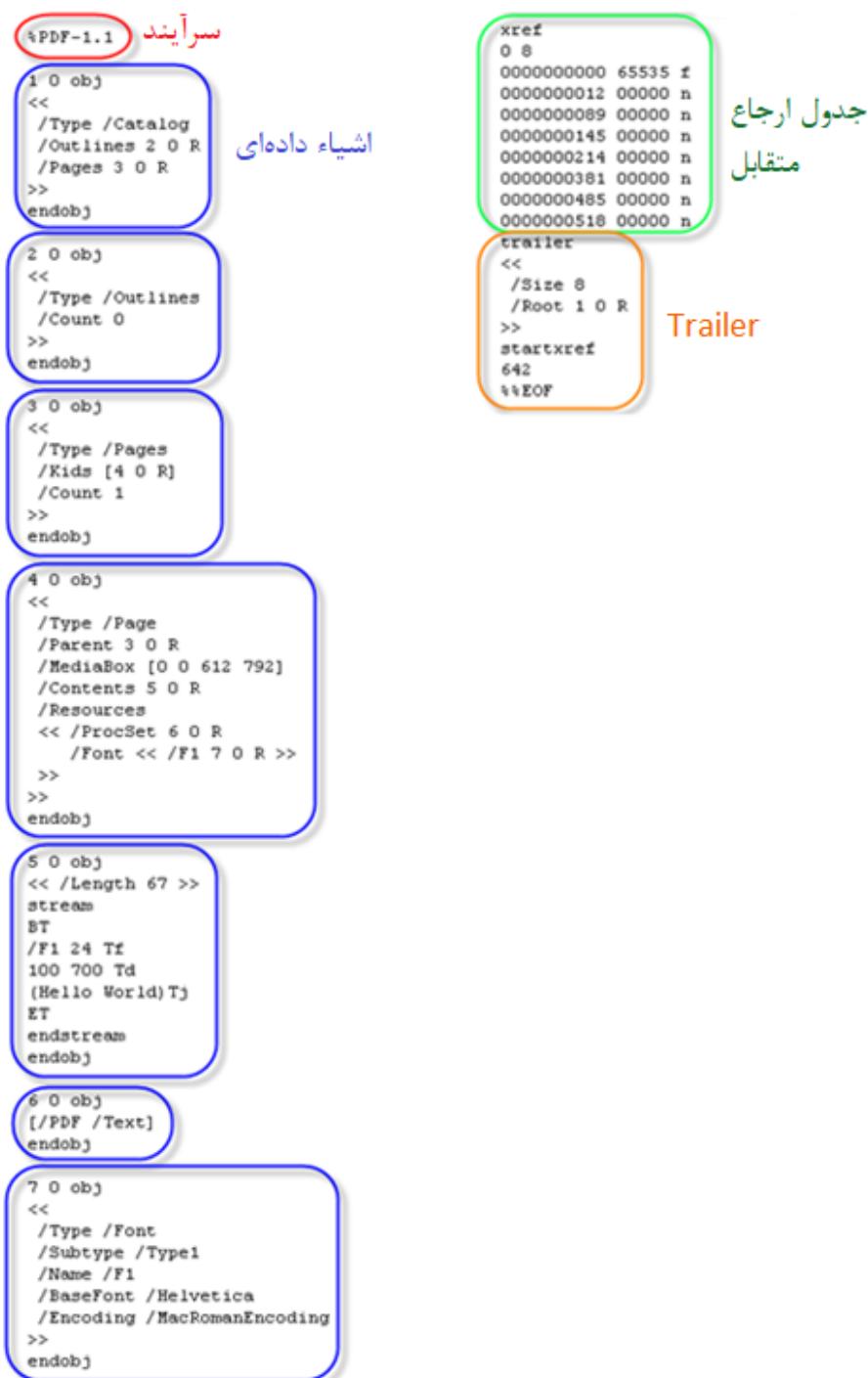
<sup>1</sup>Data Object

<sup>2</sup>Binary Stream

<sup>3</sup>Body

<sup>4</sup>Object

<sup>5</sup>Cross-reference Table



**شکل آ-۱:** یک فایل ساده PDF باز شده در یک ویراستار متقد شامل عبارت Hello Wrold است. چپ: سرازیند فایل و اشیاء داده‌ای؛ اشیاء با مستطیل آبی (شماره‌های ۱ تا ۷) مشخص شده‌اند. راست: ادامه محتويات همان فایل، شامل جدول ارجاع متقابل و Trailer.

می شود. در ادامه عدد تولیدی آن است که اگر شی با یک نسخه جدیدتر بازنویسی شود، افزایش می یابد. سپس رشته "obj" که شروع یک شیء را مشخص می کند آمده و پس از آن محتويات شیء قرار داده می شود. رشته "endobj" نیز پایان یافتن محدوده یک شیء را مشخص می کند.

- جدول ارجاع متقابل. این جدول در بدنه PDF شامل آدرس نسبی اشیای مورد ارجاع قرار گرفته داخل یک فایل به صورت بایت می باشد. در شکل آنچه ۱ این جدول شامل ۷ شیء با شناسه یک تا ۷ و یک مکان نگهدارنده برای شناسه صفر است که به همچ شیئی اشاره نمی کند.

• **Trailer** بندگی لغت ۱ از اطلاعات بدن، مثل تعداد کل اشیای دارای **size** 8 (/size)، شناسه و شماره ترتیبی شیء ریشه (root 0 R) و غیره است. فرهنگ لغت بین نمادهای <> و <> قرار می‌گیرد. ادامه Trailer شامل startxref است که آدرس نسبی شروع جدول ارجاع متقابل در فایل را مشخص می‌کند. این فن اجزه می‌دهد تا بدن از پایان با خواندن startxref پویش شود، سپس به جدول ارجاع متقابل بازگشته و آن را نیز پویش می‌کند. بدین ترتیب تنها اشیای پویش می‌شوند که نیاز هستند. در شکل ۱-۲ آدرس شروع جدول ارجاع متقابل که پس از startxref در Trailer آمده است برابر با ۶۴۲ است. در صورتی که آدرس‌های بایت‌های این فایل بررسی شود، مشاهده خواهد شد که بایت ۶۴۲ مام شروع جدول ارجاع متقابل (کاراکتر 'x') است. در پایان Trailer نیز عبارت EOF % قرار می‌گیرد که پایان فایل را مشخص می‌کند.

ساختار جدول ارجاع متقابل و Trailer به نسبت ساده و در فایل‌های گوناگون مشابه است، اما اشیای PDF انواع مختلفی دارند. برای مثال شیء شماره ۱ در شکل آ-۱، حاوی یک ساختار فرهنگ لغت است لذا بین نمادهای <> و <> واقع شده و حاوی کلیدهایی می‌شود که با کاراکتر / شروع شده و در ادامه مقادیر آنها آمده است. مثلاً مقدار R ۰۲ یک ارجاع به شیئی در همین فایل با شناسه ۲ و شماره ترتیبی ۰ است. از آنجایی که یک فایل ممکن است خیلی بزرگ باشد؛ آدرس نسبی شیئی که به آن ارجاع داده شده است از طریق جدول ارجاع متقابل، که یک جدول دسترسی تصادفی است، قایل دسترسی خواهد بود.

اشیای PDF تنها محدود به فرهنگ لغت نمی‌شوند گرچه به نظر می‌آید که بیشتر آنها دست کم شامل یک فرهنگ لغت هستند. شیء شماره ۵ در شکل آ-۱ برای نمونه، حاوی یک جریان دودویی است که بین واژه‌های کلیدی stream و endstream قرار گرفته است. تصاویر داخل PDF نمونه‌ای از جریان‌های دودویی هستند که به این صورت می‌توانند ظاهر شوند. شکل آ-۲، یک شیء حاوی تصویر را نشان می‌دهد که در یک ویراستار متن باز شده است. شیء شماره ۶ در شکل آ-۱ یک آرایه از کلیدها را شامل می‌شود. مقادیر آرایه می‌توانند انواع مختلفی داشته باشند که در هر صورت بین نمادهای [ ] و ] قرار می‌گیرند و با کاراکتر فاصله از یکدیگر تمیز داده می‌شوند. بدلیل تنوع ذکر شده در ساختار اشیای PDF، قواعد تعریف و ترکیب این اشیاء بیشترین پخش از توضیحات مشخصه‌های قالب فایل PDF را تشکیل می‌دهند.

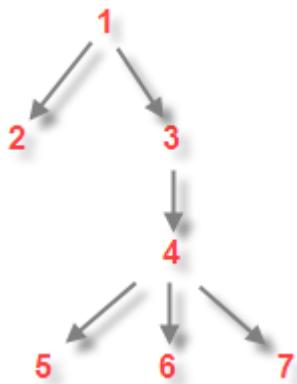
1 Dictionary

```

1397 0 obj
<</Filter/FlateDecode/Length 160>>
stream
xœ32U0P0RÐ5T02T06WH1ä*T0 `EOT
SUB (CAN [ (CANéEMSTXå€Dr.-"!-¾S-BIQi*~8PDC1-¾BELP
-¾S€³, !-¾K`!-A,-¾>¾³³ ESCESC`c£í
ÔÍ¥íæÙSš>WEEÔbgçåé¢ ¶ðô\ûKÿÿéÿ;tÊI (Ks#,,,
$, ýM>p²*z}fŽÓáNULFACKSOH.WO...@.NULÖH1£
endstream
endobj

```

**شکل آ-۲:** یک شی PDF حاوی یک تصویر. محتوای دودویی تصویر بین stream و endstream قرار گرفته است.



**شکل آ-۳:** ساختار منطقی فایل PDF نشان داده شده در شکل آ-۱. گره‌های درخت شناسه اشیاء در فایل PDF و یال‌های آن معرف نحوه فراخوانی هر شیء توسط شیء دیگر هستند.

## آ-۲ ساختار منطقی

آنچه در بخش آ-۱ صحبت شد مربوط ساختار فیزیکی یک فایل PDF و نحوه قرار گرفتن اجزای فایل در کنار یکدیگر بود (مرتبط با گام پویش در هنگام اجرا). ساختار منطقی فایل PDF، یعنی قواعد حاکم بر نحوه تفسیر و پردازش آنچه در یک فایل قرار دارد و ارتباط بین اجزا، یک ساختار سلسله مراتبی است (مرتبط با گام پرداخت در هنگام اجرا). شناسه شیء ریشه در Trailer مشخص می‌شود. بهمین ترتیب شیء یا اشیای بعدی مورد نیاز در شیء ریشه معلوم می‌گردد. در فایل PDF شکل آ-۱ شیء شماره ۱ ریشه است. اشیای شماره ۲ و ۳ داخل بدنه شیء ۱ مورد دسترسی قرار می‌گیرند و در نهایت ترتیب نحوه دسترسی‌ها به صورت یک درخت قابل نمایش است که ساختار منطقی فایل را نشان می‌دهد. شکل آ-۳ ساختار منطقی فایل نشان داده شده در شکل آ-۱ را نشان می‌دهد.

ساختار فیزیکی یک فایل PDF را می‌توان بدون تغییر ساختار منطقی آن، به شکل دیگری تبدیل کرد. برای مثال در شکل آ-۱ اشیاء به ترتیب صعودی شناسه خود در فایل ظاهر شده بودند. می‌شود ترتیب قرارگرفتن

```

xref
0 8
000000000000 65535 f
0000000565 00000 n
0000000509 00000 n
0000000440 00000 n
0000000273 00000 n
0000000169 00000 n
0000000136 00000 n
0000000012 00000 n

```

**شکل آ\_۴:** جدول ارجاع متقابل بروزرسانی شده در حالتی که اشیاء به ترتیب نزولی شناسه خود در فایل PDF ظاهر شده‌اند.

اشیاء را به صورت نزولی در آورد. در این صورت جدول ارجاع متقابل باستی بروزرسانی شود؛ زیرا، هر شی در در مکان متفاوتی نسبت به قبل قرار گرفته است. اما این پدیده تأثیری بر ساختار منطقی ندارد. ساختار منطقی کماکان همان ساختار شکل آ\_۳ و نتیجه اجرا نیز با فایل قبلی یکسان خواهد بود. جدول ارجاع متقابل برای حالتی که اشیاء به صورت نزولی در فایل ظاهر شده باشند، در شکل آ\_۴ نشان داده شده است.

به طور کلی اشیاء PDF می‌توانند در مکان‌های تصادفی داخل یک فایل PDF ظاهر شوند بدون اینکه تأثیر بر پرداخت فایل داشته باشند. برای فایل ساده شکل آ\_۱ تنها با تعویض ترتیب ظاهر شدن اشیاء می‌توان تعداد ۵۰۴۰ (برابر با ۷!) فایل با ساختار فیزیکی متفاوت داشت. این درحالی است که تغییر ترتیب قرارگیری اشیا تنها یک روش برای جابه‌جایی ساختار فیزیکی فایل PDF است و روش‌های دیگری مثل تغییر محل قرارگیری جدول ارجاع متقابل نیز وجود دارد. بنابراین رابطه بین ساختار منطقی و فیزیکی فایل PDF به صورت یک به چند است.

## آ\_۳ بروزرسانی فایل PDF

دیدیم که چگونه ساختار منطقی فایل PDF مستقل از ساختار فیزیکی آن شکل می‌گیرد. بروزرسانی فایل PDF بر همین مبنای است. فایل‌های PDF می‌توانند به صورت افزایشی<sup>۱</sup> بروزرسانی شوند. بدین ترتیب که اگر فایل PDF بخواهد اطلاعات داخل شیء شماره ۷ را بروزرسانی کند، یک بدنی جدید آغاز می‌کند. سپس محتوای شیء جدید را داخل آن نوشه، عدد تولیدی (عدد بعد از شناسه شیء) را یک واحد نسبت به عدد تولیدی شیء قدیم افزایش داده و برای شیء جدید می‌نویسد. در انتها نیز یک جدول ارجاع متقابل را که به شیء جدید اشاره می‌نماید، تولید کرده و بدنی جدید را به سند قبلی الصاق می‌کند.

<sup>۱</sup> Incremental

## پیوست ب

### IUST-DeepFuzz فازر

در این پیوست جزئیات پیاده‌سازی و استفاده از روش پیشنهادی خود را در قالب محصول نرم‌افزاری-IUST DeepFuzz شرح می‌دهیم. کلیه کدها و مستندات این محصول از طریق صفحه پروژه در وب‌سایت GitHub<sup>۱</sup> قابل مشاهده و دریافت است. برنامه به صورت مجموعه‌ای از اسکریپت‌های زبان پایتون نوشته شده است؛ لذا، روی هر سیستم عاملی پس از نصب کتابخانه‌ها و چارچوب‌های لازم قابل اجرا خواهد بود. فایل‌های پروژه، به‌تفکیک وظایف، در تعدادی دایرکتوری سازماندهی شده است. همچنین بخشی از فازر که عملیات آزمون فازی، پایش خطأ و ابزارگذاری کد SUT را انجام می‌دهد، به صورت یک فایل Batch در سیستم عامل ویندوز نوشته شده است که در صورت تغییر سیستم عامل یا هریک از ابزارهای پایش و ابزارگذاری SUT، بایستی تغییر داده شود. پیمانه تولیدکننده خودکار داده‌های آزمون اما نیاز به تغییر ندارد.

#### ب-۱ پیش‌نیازهای نرم‌افزاری و سخت‌افزاری

پیش‌نیازهای نرم‌افزاری پروژه در جدول ب-۱ آمده است. پس از نصب تمامی بسته‌ها اسکریپت‌های اصلی پروژه قابل اجرا خواهد بود. برای تسریع آموزش مدل‌ها استفاده از سیستمی با پردازش‌گر گرافیکی قوی (تعداد هسته‌های بالا) توصیه می‌شود ولی الزامی نیست. همچنین پردازش‌گر مرکزی قوی و حافظه اصلی بالا (۸ گیگابایت و بیشتر) برای اجرای قابل قبول پروژه مورد نیاز خواهد بود.

#### ب-۲ ساختار سطح بالای پروژه

در دایرکتوری ریشه پروژه تعدادی فایل و تعدادی زیردایرکتوری قرار دارند که بخش‌های مختلف برنامه را تشکیل می‌دهند. در انتهای نام هر فایل یک عدد ترتیبی قرار دارد که نسخه آن را مشخص می‌کند. در میان فایل‌های دارای یک نام بدین ترتیب فایل با نسخه بالاتر اجرایی است و نیازی به فایل‌های قبلی نیست.

<sup>۱</sup>[https://github.com/m-zakeri/iust\\_deep\\_fuzz](https://github.com/m-zakeri/iust_deep_fuzz)

### جدول ب-۱: بسته‌های نرم‌افزاری مورد نیاز جهت اجرای صحیح برنامه IUST DeepFuzz

ردیف	بسته نرم‌افزاری	نسخه مورد نیاز
۱	Python	نسخه ۳/۵ و بالاتر
۲	TensorFlow	نسخه ۱/۵ و بالاتر
۳	Keras	نسخه ۲/۲/۰

دایرکتوری‌ها عبارتند از:

- **دایرکتوری ریشه.** در این دایرکتوری ریشه فایل‌های اصلی پیکربندی<sup>۱</sup>، فایل تعريف مدل‌های ژرف، فایل آموزش مدل و تولید داده از مدل و نیز فایل راهنمای وجود دارد.
- **دایرکتوری batch\_jobs.** این دایرکتوری کدهای مربوط به پیمانه‌های تزریق و پایش (خطا و پوشش کد) فازر را شامل می‌شود.
- **دایرکتوری binary\_to\_base64.** این دایرکتوری برای کارهای آتی رزرو شده و در نسخه فعلی محصول به کار نمی‌رود.
- **دایرکتوری dataset.** این دایرکتوری همان‌طور که از نام آن پیداست. محل قرار گیری مجموعه داده مورد آموزش و آزمون مدل‌ها است. برای قالب‌های فایل جدید کافی است مجموعه داده را زیر دایرکتوری مختص به آن قرار دهیم. در حال حاضر مجموعه داده خوبی برای فایل‌های PDF و مجموعه داده کوچکی نیز برای فایل‌های XML داخل آن قرار دارند.
- **دایرکتوری generated\_results.** در این دایرکتوری داده‌های آزمون جدید تولید شده توسطه مدل‌های ژرف ذخیره می‌شوند.
- **دایرکتوری incremental\_update.** این دایرکتوری شامل کدهای مربوط به بروزرسانی افزایشی و ساخت فایل‌های PDF جدید است. بنابراین مختص آزمون فازی قالب فایل PDF است و در قالب‌های فایل دیگر کاربردی ندارد.
- **دایرکتوری logs\_csv.** در این دایرکتوری فایل‌های گزارش ضبط شده در فرایند آموزش مدل‌ها ذخیره می‌شود. این فایل‌ها شامل میزان خطای دقت و سرگشتشگی مدل در هر دوره آموزش هستند.
- **دایرکتوری logs\_tensorboard.** فایل‌های گزارش مربوط به ابزار مصورسازی Tensorboard در این دایرکتوری ذخیره می‌شود. این فایل‌ها سپس توسط همین ابزار خوانده و گزارش‌های دقت، خطای داده مدل و غیره را به صورت تصویری در اختیار آزمون‌گر قرار می‌دهد.

<sup>۱</sup>Configuration

- دایرکتوری **model\_checkpoint**. در پایان هر دوره آموزش مدل، یک نمونه از مدل در این دایرکتوری ذخیره می‌شود. هر نمونه مستقل از سایر نمونه‌های بوده است. آزمون‌گر بدین ترتیب پس از اتمام فرایند آموزش می‌تواند بهترین مدل را برای تولید داده آزمون انتخاب کند.
- دایرکتوری **modelpic**. این دایرکتوری شامل گراف محاسباتی مربوط به مدل‌های ژرف است که توسط کتابخانه Keras تولید شده است.
- دایرکتوری **seed**. این دایرکتوری پوشش کد تمامی فایل‌های دانه اولیه آزمون را ذخیره می‌کند.

## ب-۳ پیکربندی پروژه

در حال حاضر این محصول مبتنی بر GUI نیست و کلیه تنظیمات از طریق فایل config.py در دایرکتوری ریشه مشخص می‌شود و مهمترین فایلی است که در پروژه در اختیار آزمون‌گر قرار دارد. این فایل شامل تعدادی فرهنگ لغت است. هر فرهنگ لغت تعدادی کلید به صورت کلید-مقدار دارد که کلید نام تنظیم ورودی بود که توسط برنامه مشخص شده است و مقدار آن توسط آزمون‌گر قبل از انجام آزمون تعیین می‌شود. فایل config.py کاملاً مستندگذاری شده و مقادیر قابل استفاده برای هر کلید در مقابل آن درج شده است. برای تغییر آن کافی است یکی از مقادیر معتبر را به جای مقدار پیش‌فرض قرار دهید. مهمترین تنظیم‌های قابل اعمال در این فایل عبارتند از:

• **file\_format**. این کلید قالب فایل مورد آزمون را مشخص می‌کند. برای مثال PDF یا XML.

• **training\_set\_path**. این کلید آدرس مجموعه آموزش را مشخص می‌کند.

• **validation\_set\_path**. این کلید آدرس مجموعه ارزیابی را مشخص می‌کند.

• **testing\_set\_path**. این کلید آدرس مجموعه آزمون را مشخص می‌کند.

• **new\_objects\_path**. این کلید آدرس محل ذخیره داده‌های آزمون تولید شده را مشخص می‌کند.

افزون بر تنظیمات بالا، تعدادی از تنظیمات مختص آزمون فازی قالب PDF هستند. این تنظیمات در فرهنگ لغت iu\_config تعریف شده‌اند؛ از جمله:

• **single\_object\_update**. این کلید مقادیر True و False را می‌گیرد و همان‌طور که از نام آن پیداست تعیین می‌کند که هنگام تولید فایل PDF جدید تنها یک شیء بروزرسانی شود و یا چند شیء.

• **portion\_of\_rewrite\_objects**. این کلید نسبت تعداد اشیای مورد بروزرسانی را در حالت MOU مشخص می‌کند. بنابراین تنها وقتی که تنظیم single\_object\_update برابر False است به کار می‌رود.

- این کلید نوع انتخاب اشیاء برای بروزرسانی را مشخص می‌کند که می‌تواند یکی از موارد random برای انتخاب تصادفی، bottom\_up برای انتخاب برحسب شماره نزولی اشیاء و top\_down برای انتخاب برحسب شماره صعودی اشیاء باشد.
- این کلید آدرس فایل‌های میزبان را مشخص می‌کند.

## ب-۴ پیاده‌سازی‌ها

هر یک از اسکریپت‌های پایتون بخشی از پروژه را پیاده‌سازی می‌کند. فایل deep\_models.py هریک از مدل‌های ژرف را در قالب یکتابع پایتون تعریف کرده است. فایل lstm\_text\_generation\_pdf\_objs\_9datafuzz.py الگوریتم فاز عصبی داده و فایل lstm\_text\_generation\_pdf\_objs\_9formatfuzz.py الگوریتم فاز عصبی فراداده را پیاده‌سازی کرده است. همچنین فایل iu\_6.py بروزرسانی افزایشی PDF را پیاده‌سازی می‌کند.

### ب-۴-۱ تعریف مدل‌ها

در دو صورت می‌توان مدل‌های یادگیری را تعریف کرد. یک روش با استفاده از پشتکردن لایه‌ها مختلف که برای مدل‌های ترتیبی به کار می‌رود و یک روش با استفاده از کلاس Model و API‌های آن، که برای ساخت مدل‌هایی با چندین ورودی و خروجی و به طور کلی معماری‌های پیچیده‌تر استفاده می‌شود. تعریف مدل‌های ما با روش اول انجام شده است. در برنامه ب-۱ تعریف مدل‌ها را آورده‌ایم.

```

1 from keras.models import Sequential, load_model
2 from keras.layers import Dense, Activation, Dropout
3 from keras.layers import LSTM, Bidirectional
4
5 # Unidirectional LSTM (Many to One)
6 def model_1(input_dim, output_dim):
7     print('Build model... ')
8     model = Sequential()
9     model.add(LSTM(128, input_shape=input_dim))
10    model.add(Dense(output_dim))
11    model.add(Activation('softmax'))
12    return model, 'model_1'
13
14 # Unidirectional LSTM (Many to One)
15 def model_2(input_dim, output_dim):
16    model = Sequential()
```

```

17     model.add(LSTM(128, input_shape=input_dim,
18                   return_sequences=True))
19     model.add(LSTM(128, input_shape=input_dim,
20                   return_sequences=False))
21     model.add(Dense(output_dim))
22     model.add(Activation('softmax'))
23     return model, 'model_2'
24
25 # Unidirectional LSTM (Many to One)
26 def model_3(input_dim, output_dim):
27     model = Sequential()
28     model.add(LSTM(256, input_shape=input_dim,
29                   return_sequences=True, recurrent_dropout=0.1))
30     model.add(Dropout(0.3))
31     model.add(LSTM(256, input_shape=input_dim,
32                   return_sequences=False, recurrent_dropout=0.1))
33     model.add(Dropout(0.3))
34     model.add(Dense(output_dim))
35     model.add(Activation('softmax'))
36     return model, 'model_3'
37
38 # Bidirectional LSTM (Many to One)
39 def model_4_1(input_dim, output_dim):
40     model = Sequential()
41     model.add(Bidirectional(LSTM(256, return_sequences=
42                             False), input_shape=input_dim, merge_mode='sum'))
43     model.add(Dense(output_dim))
44     model.add(Activation('softmax'))
45     return model, 'model_4_1'
46
47 # Bidirectional Deep LSTM (Many to One)
48 def model_4_2(input_dim, output_dim):
49     model = Sequential()
50     model.add(Bidirectional(LSTM(128, return_sequences=True
51 ), input_shape=input_dim, merge_mode='sum'))
52     model.add(Bidirectional(LSTM(128, return_sequences=

```

```

    False), merge_mode='sum'))
47   model.add(Dense(output_dim))
48   model.add(Activation('softmax'))
49   return model, 'model_4_2'
50 # End of model definition

```

برنامه ب-۱: پیاده‌سازی مدل‌های یادگیری در Keras

## ب-۴-۲ آموزش و تولید داده

کدهای مربوط به فرایند آموزش مدل و تولید داده از مدل در یک فایل قرار دارند به ترتیب اجرا می‌شوند. در اینجا از ذکر کدها امتناع می‌کنیم؛ زیرا طولانی بوده و فایل کامل در مخزن پروژه وجود دارد. اجرای نهایی برنامه پس از انجام تنظیمات گفته شده از طریق فراخوانی هر فایل در خط فرمان سیستم عامل در مسیر پروژه انجام می‌شود. برای آموزش مدل می‌توان تعداد دوره محدودی در نظر گرفت و سپس با بررسی میزان خطای آموزش را متوقف ساخت.

## ب-۴-۳ پیاده‌سازی آزمون فازی

آزمون فازی همان‌طور که گفتیم توسط یه برنامه Batch روی سیستم عامل ویندوز نوشته شده است. برنامه ب-۲ یک پیاده‌سازی اولیه از آن را نشان می‌دهد.

```

1 REM IUST DeepFuzz version 0.2
2 REM @ECHO off
3 SET PATH=%PATH%; "C:\Program Files (x86)\Microsoft Visual
   Studio 14.0\Team Tools\Performance Tools"
4 CD D:\SUT_DIRECTORY
5 CLS
6
7 START VSPerfMon /coverage /output:fuzzing_code_coverage.
   coverage
8 timeout /t 5
9
10 FOR %%i IN (.\\TEST_DATA\\*.pdf) DO (
11   mupdf %%i
12   timeout /t 10
13   taskkill /IM mupdf.exe

```

```

14    timeout /t 2
15    )
16
17 timeout /t 5
18 VSPerfCmd /shutdown

```

---

برنامه ب-۲: پیاده‌سازی آزمون فازی

## ب-۵ ملاحظات عملی

در پیاده‌سازی همواره با برخی از پیچیدگی‌ها روبرو می‌شویم که در تئوری مسئله وجود ندارد. در این پروژه تبدیل تمامی توالی‌های آموزشی به بردارهای عددی نیازمند حافظه بالای ۲۰ گیگابایت بود. به عبارت دیگر امکان بارگذاری یکباره همه مجموعه آموزش در حافظه اصلی نبود. برای حل این مشکل، هر بار یک دسته از توالی‌ها را انتخاب، آنها را از حافظه جانبی وارد حافظه اصلی کرده و پس از شرکت دادن در فرایند آموزش، این دسته را خارج و دسته بعدی را می‌خوانیم. این امکان با نوشتن از یک تابع از نوع Generator به کد برنامه اضافه گردید. مصرف حافظه به این ترتیب با تعیین اندازه دسته، در اختیار برنامه‌نویس و آزمون‌گر خواهد بود.

# واژه‌نامه فارسی به انگلیسی

Representation .....	بازنمایی .....
Supervised .....	بانظارت .....
Bias .....	بایاس .....
Magic Bytes .....	بایت جادویی .....
Malformed .....	بدشکل .....
Body .....	بلدنه .....
Label .....	برچسب .....
Recognizer Program .....	برنامه شناسنده .....
Generator Program .....	برنامه مولد .....
Basic Block .....	بلوک پایه .....
Exploit .....	بهربرداری .....
Overfitting .....	بیشبرازش .....
Softmax .....	بیشینه هموار .....

پ

Filter .....	پاکساز .....
Monitor .....	پایش .....
Render .....	پرداخت .....
Background Process .....	پردازه پس زمینه .....
Backpropagation .....	پس انتشار .....
Basic Block Coverage .....	پوشش بلوک پایه .....
Line Coverage .....	پوشش خط .....
Partial Line Coverage .....	پوشش خط جزئی .....
Domain Coverage .....	پوشش دامنه .....
Statement Coverage .....	پوشش دستور .....
Syntax-Based Coverage .....	پوشش ساختار نحوی .....

باج افزار .....

Test .....	آزمون .....
Security Testing .....	آزمون امنیت .....
Fuzz Testing .....	آزمون فازی .....
Stress Testing .....	آزمون فشار .....
Robustness Testing .....	آزمون قدرتمندی .....
Penetration Testing .....	آزمون نفوذ .....
Vulnerability .....	آسیب‌پذیری .....
Hyper-parameter .....	ابر پارامتر .....
Instrumenting .....	ابزارگذاری .....
Dependability .....	اتکاپذیری .....
Noise .....	اختلال .....
Extrinsic Evaluation .....	ارزیابی بیرونی .....
Intrinsic Evaluation .....	ارزیابی درونی .....
Error .....	اشکال .....
Validation .....	اعتبارسنجی .....
Input Space Partitioning .....	افزایشی فضای ورودی .....
Incremental .....	افزوده .....
Augmented .....	افزوده .....
Vulnerable Pattern .....	الگوی آسیب‌پذیر .....
End to End .....	انتهای‌به‌انتهای .....
Explosion .....	انفجار .....
Internet of Things .....	اینترنت چیزها .....

ب

Black Box .....	جعبه سیاه .....	Branch Coverage .....	پوشش شاخه .....
		Code Coverage .....	پوشش کد .....
		Graph Coverage .....	پوشش گراف .....
	ح	Path Coverage .....	پوشش مسیر .....
Greedy .....	حریصانه .....	Logic Coverage .....	پوشش منطق .....
		Prefix .....	پیشوند .....
		Configuration .....	پیکربندی .....
	خ	Corpus .....	پیکره .....
		Module .....	پیمانه .....
Failure .....	خرابی .....		
Fault .....	خطا .....		
Cross Entropy Error .....	خطای آنتروپی مقاطع .....		
Mean Absolute Error .....	خطای مطلق میانگین .....		
Mean Squared Error .....	خطای میانگین مربعات .....		
Well-formed .....	خوش‌شکل .....		
	ت	Merge Function .....	تابع ادغام .....
		Activation Function .....	تابع انگیزش .....
		Error Function .....	تابع خطای .....
		Cost Function .....	تابع هزینه .....
	د	History .....	تاریخچه .....
Test Data .....	داده آزمون .....	Parse .....	جزئیه .....
Initial Seed .....	دانه اولیه .....	Parser .....	جزئیه‌گر .....
Verification .....	درستی‌یابی .....	Undecidable .....	تصمیم‌ناپذیر .....
Shuffle .....	درهم‌آمیزی .....	Adaptive .....	تطبیق‌پذیر .....
Epoch .....	دوره .....	Generalization .....	تعیین‌پذیر .....
Bidirectional .....	دوسیویه .....	Diversity .....	تنوع .....
		Sequence .....	توالی .....
	ر	Multinomial Distribution .....	توزیع چندجمله‌ای .....
Forward .....	روبه‌جلو .....	Token .....	токن .....
Backward .....	روبه‌عقب .....	Binary Token .....	توكن دودویی .....
	ج		
Context .....	زمینه .....	Cross-reference Table .....	جدول ارجاع متقابل .....
	ز	Binary Stream .....	جريان دودویی .....
		Gray Box .....	جعبه خاکستری .....
		White Box .....	جعبه سفید .....

Metadata .....	فراداده .....	س
Dictionary .....	فرهنگ لغت .....	
Memory Corruption .....	فساد حافظه .....	
	Consistent .....	سازگار .....
	Perplexity .....	سرگشتنگی .....
	Oracle .....	سروش .....
	ق	
Portability .....	قابلیت حمل .....	ش
	Neural Network .....	شبکه عصبی .....
	Feed-forward Neural Network .....	شبکه عصبی روبه‌جلو .....
Source Code .....	کد منبع .....	شبکه عصبی ژرف .....
Encoder-Decoder .....	کدگذار-کدگشا .....	Object .....
Seed Minimization .....	کمینه‌سازی دانه .....	شیء داده .....
Exception Handling .....	کنترل استثنای .....	Data Object .....
	گ	ض
Step .....	گام .....	ضد فازینگ .....
Time Step .....	گام زمانی .....	ضریب انشعاب .....
Forward Pass .....	گذر جلو .....	Branching Factor .....
	ل	ط
Hidden Layer .....	لایه پنهان .....	طبقه‌بندی .....
	م	طراحی .....
Generation Based .....	مبتنی بر تولید .....	Classification .....
Mutation Based .....	مبتنی بر جابه‌جایی .....	Design .....
Test Set .....	مجموعه آزمون .....	ع
Training Set .....	مجموعه آموزش .....	عصب .....
Validation Set .....	مجموعه ارزیابی .....	Neuron .....
Generative Model .....	مدل مولد .....	ف
		فازر .....
		Fuzzer .....
		فازر قالب فایل .....
		File Format Fuzzer .....

Learn&Fuzz .....	یادگیری و فاز	Specification .....	مشخصه
Unidirectional .....	یک‌سویه	Valid .....	معتبر
		Coverage Criteria .....	معیارهای پوشش
		Concept .....	مفهوم
		Regularization .....	منظم‌سازی
		Component .....	مؤلفه
		Attacker .....	مهاجم
		Reverse Engineering .....	مهندسی معکوس
		Buffer .....	میانگیر
		Host .....	میزبان

## ن

Vanishing .....	نایپید شدن
Learning Rate .....	نرخ یادگیری
Normalization .....	نرمال‌سازی
Markers .....	نشانگر
Requirement .....	نیازمندی

## و

Unit .....	واحد
Vocabulary .....	واژگان
Task .....	وظیفه

## ه

Smoothing .....	هموارسازی
-----------------	-----------

## ی

Supervised Learning .....	یادگیری با ناظارت
Deep Learning .....	یادگیری ژرف

# واژه‌نامه انگلیسی به فارسی

Code Coverage .....	پوشش کد .....	A
Component .....	مولفه .....	تابع انگیزش .....
Concept .....	مفهوم .....	تطبیق‌پذیر .....
Configuration .....	پیکربندی .....	ضد فازینگ .....
Consistent .....	سازگار .....	مهاجم .....
Context .....	زمینه .....	افزوده .....
Corpus .....	پیکره .....	
Cost Function .....	تابع هزینه .....	
Coverage Criteria .....	معیارهای پوشش .....	B
Cross Entropy Error .....	خطای آتروپی متقاطع .....	پردازه پس‌زمینه .....
Cross-reference Table .....	جدول ارجاع متقابل .....	پساننتشار .....
		Backward .....
		روبه‌عقب .....
		بلوک پایه .....
		Basic Block .....
		Basic Block Coverage .....
		باپاس .....
		دوسویه .....
		جریان دودویی .....
		توكن دودویی .....
		Binary Stream .....
		Binary Token .....
		جعبه سیاه .....
		Body .....
		پوشش شاخه .....
		Branch Coverage .....
		ضریب انشعاب .....
		Buffer .....
		E
Encoder-Decoder .....	کدگذار-کدگشا .....	C
End to End .....	انتهابه‌انتها .....	طبقه‌بندی .....
		Classification .....

## H

Hidden Layer .....	لایه پنهان
History .....	تاریخچه
Host .....	میزبان
Hyper-parameter .....	ابر پارامتر

Epoch .....	دوره
Error .....	اشکال
Error Function .....	تابع خطأ
Exception Handling .....	کنترل استثنا
Exploit .....	بهره‌برداری
Explosion .....	انفجار
Extrinsic Evaluation .....	ارزیابی بیرونی

## I

Incremental .....	افزایشی
Initial Seed .....	دانه اولیه
Input Space Partitioning .....	افراز فضای ورودی
Instrumenting .....	ابزارگذاری
Internet of Things .....	اینترنت چیزها
Intrinsic Evaluation .....	ارزیابی درونی

## F

Failure .....	خرابی
Fault .....	خطا
Feed-forward Neural Network .....	شبکه عصبی روبه‌جلو
File Format Fuzzer .....	فازر قالب فایل
Filter .....	پاک‌ساز
Forward .....	روبه‌جلو
Forward Pass .....	گذر جلو
Fuzz Testing .....	آزمون فازی
Fuzzer .....	فازر

## L

Label .....	برچسب
Learn&Fuzz .....	یادگیری و فاز
Learning Rate .....	نرخ یادگیری
Line Coverage .....	پوشش خط
Logic Coverage .....	پوشش منطق

## G

Generalization .....	تعیین‌پذیر
Generation Based .....	مبتنی بر تولید
Generative Model .....	مدل مولد
Generator Program .....	برنامه مولد
Graph Coverage .....	پوشش گراف
Gray Box .....	جعبه خاکستری
Greedy .....	حریصانه

## M

Magic Bytes .....	بایت جادویی
Malformed .....	بدشکل
Markers .....	نشانگر
Mean Absolute Error .....	خطای مطلق میانگین
Mean Squared Error .....	خطای میانگین مربعات
Memory Corruption .....	فساد حافظه
Merge Function .....	تابع ادغام
Metadata .....	فرداده

Regularization .....	منظم سازی .....	Module .....	پیمانه .....
Render .....	پرداخت .....	Monitor .....	پایش .....
Representation .....	بازنمایی .....	Multinomial Distribution .....	توزیع چندجمله‌ای .....
Requirement .....	نیازمندی .....	Mutation Based .....	مبتنی بر جابه‌جایی .....
Reverse Engineering .....	مهندسی معکوس .....		
Robustness Testing .....	آزمون قدرتمندی .....		

## N

Security Testing .....	آزمون امنیت .....	Neural Network .....	شبکه عصبی .....
Seed Minimization .....	کمینه‌سازی دانه .....	Noise .....	اختلال .....
Sequence .....	توالی .....	Normalization .....	نرمال‌سازی .....
Shuffle .....	درهم آمیزی .....		
Smoothing .....	هموارسازی .....		
Softmax .....	بیشینه هموار .....	Object .....	شیء .....
Source Code .....	کد منبع .....	Oracle .....	سرچش .....
Specification .....	مشخصه .....	Overfitting .....	بیش‌برازش .....
Statement Coverage .....	پوشش دستور .....		
Step .....	گام .....		
Stress Testing .....	آزمون فشار .....		

## O

Supervised .....	بانظارت .....	Parse .....	تجزیه .....
Supervised Learning .....	یادگیری بانظارت .....	Parser .....	تجزیه‌گر .....
Syntax-Based Coverage .....	پوشش ساختار نحوی .....	Partial Line Coverage .....	پوشش خط جزئی .....
		Path Coverage .....	پوشش مسیر .....
		Penetration Testing .....	آزمون نفوذ .....
		Perplexity .....	سرگشتگی .....
		Portability .....	قابلیت حمل .....
		Prefix .....	پیشوند .....

## P

Task .....	وظیفه .....	Ransomware .....	باچ افزار .....
Test .....	آزمون .....	Recognizer Program .....	برنامه شناسنده .....
Test Data .....	داده آزمون .....		
Test Set .....	مجموعه آزمون .....		
Time Step .....	گام زمانی .....		
Token .....	توكن .....		
Training Set .....	مجموعه آموزش .....		

## R

131
-----

## U

Undecidable .....	تصمیم‌ناپذیر .....
Unidirectional .....	یک‌سویه .....
Unit .....	واحد .....

## V

Valid .....	معتبر .....
Validation .....	اعتبارسنجی .....
Validation Set .....	مجموعه ارزیابی .....
Vanishing .....	ناپدید شدن .....
Verification .....	درستی‌یابی .....
Vocabulary .....	واژگان .....
Vulnerability .....	آسیب‌پذیری .....
Vulnerable Pattern .....	الگوی آسیب‌پذیر .....

## W

Well-formed .....	خوش‌شکل .....
White Box .....	جعبه سفید .....

**Abstract:**

**F**UZZ testing (Fuzzing) is a dynamic software testing technique. In this technique with repeated generation and injection of malformed test data to the software under test (SUT), we are looking for the possible faults and vulnerabilities. To this goal, fuzz testing requires varieties of test data. The most critical challenge is to handle the complexity of the file structures as program input. Surveys have revealed that many of the generated test data in these cases follow restricted numbers and superficial paths, because of being rejected by the parser of SUT in the initial stages of parsing. Using the grammatical structure of input files to generate test data lead to increase code coverage. However, often, the grammar extraction is performed manually, which is a time consuming, costly and error-prone task. In this thesis, we proposed an automated method for hybrid test data generation. To this aim, we apply neural language models (NLMs) that are constructed by recurrent neural networks (RNNs). The proposed models by using deep learning techniques can learn the statistical structure of complex files and then generate new textual test data, based on the grammar, and binary data, based on mutations. Fuzzing the generated data is done by two newly introduced algorithms, called neural fuzz algorithms that use these models. We use our proposed method to generate test data, and then fuzz testing of MuPDF complicated software which takes portable document format (PDF) files as input. To train our generative models, we gathered a large corpus of PDF files. Our experiments demonstrate that the data generated by this method leads to an increase in the code coverage, more than 7%, compared to state of the art file format fuzzers such as American fuzzy lop (AFL). Experiments also indicate a better learning accuracy of simpler NLMS in comparison with the more complicated encoder-decoder model and confirm that our proposed models can outperform the encoder-decoder model in code coverage when fuzzing the SUT.

**Keywords:** Fuzz Testing, Test Data, Code Coverage, Deep Learning, Recurrent Neural Network.



**Iran University of Science and Technology  
School of Computer Engineering**

# **Automatic Test Data Generation in File Format Fuzzers**

**A Thesis Submitted in Partial Fulfillment of the Requirement for the Degree  
of Master of Science in Computer Engineering**

**By:**

**Morteza Zakeri Nasrabadi**

**Supervisor:**

**Dr. Saeed Parsa**

**September 2018**