ToolShield: Prompt Injection Detection for Tool-Using LLM Agents

Giselle Evita

Bachelor Thesis

Department of Computer Science
Technische Universität Darmstadt

Supervisor: [Name TBD]

February 2026

Contents

1 Problem Statement and Research Questions

Large Language Models (LLMs) are increasingly deployed in systems that can invoke external tools such as databases, file systems, or web APIs. While tool-use enables automation and improved user experience, it also introduces new security risks: user-provided text can manipulate the model into ignoring tool usage constraints or generating malicious tool arguments. This class of attacks is commonly referred to as *prompt injection*. In practice, prompt injection attempts can lead to unauthorized data access, unintended tool execution, or leakage of internal system instructions.

This thesis addresses the problem of detecting prompt injection attempts in tool-using LLM systems. The goal is to develop a detection mechanism that can be deployed as a gating layer before tool execution, classifying tool-use prompts as benign or malicious. In addition to baseline approaches (heuristics, TF-IDF, and transformer classifiers), this thesis investigates a critical implementation issue for context-aware detection models: how token budget allocation and truncation can systematically remove the user prompt under long tool schemas.

Definition (Prompt Injection). In the context of this thesis, *prompt injection* refers to a malicious user input that attempts to override the intended instruction hierarchy of a tool-using LLM system. Concretely, the attacker aims to cause the model to disregard system or tool constraints (e.g., role instructions or tool schemas) and instead follow adversarial directives embedded in the user prompt.

1.1 Research Questions

This thesis is structured around the following research questions:

- **RQ1:** How accurately can prompt injection attempts be detected in single-turn tool-use prompts using lightweight baselines (heuristics and TF-IDF) compared to transformer-based classifiers?
- **RQ2:** To what extent do context-aware models generalize to unseen attack families when evaluated under distribution shift (attack-holdout protocol)?
- **RQ3:** How does truncation strategy under long tool schemas influence the performance of context-augmented transformer detectors?

1.2 Hypotheses

The experimental evaluation is guided by two hypotheses:

- **H1 (Truncation bias under long schemas):** Under enterprise-length tool schemas, naive right-truncation removes the user prompt from the model input, causing a significant degradation in prompt injection detection performance.
- **H2 (Prompt-preserving truncation mitigates bias):** A prompt-preserving truncation strategy that reserves a minimum token budget for the user prompt prevents prompt loss and maintains high detection performance under long schemas.

2 Related Work

Prompt injection attacks have emerged as a central security challenge for Large Language Models (LLMs), particularly in settings where models are connected to external tools and execute actions beyond text generation. ToolShield builds on prior work in prompt injection, jailbreak robustness, tool-use security, and LLM safety filtering. This chapter summarizes the most relevant research directions and positions ToolShield within the existing literature.

2.1 Prompt Injection and Jailbreak Attacks

Prompt injection refers to adversarial inputs designed to manipulate an LLM into ignoring intended instructions or safety constraints. Early work highlighted that LLM behavior is strongly shaped by natural language instructions, which can be exploited through carefully crafted prompts [11]. Related research on *jailbreaks* demonstrated that even aligned models can be induced to generate disallowed outputs using roleplay prompts, obfuscation, or indirect instructions [9].

Several studies propose taxonomies of attack strategies, including direct override instructions, instruction hierarchy manipulation, and hidden or encoded payloads [12]. These findings suggest that prompt injection is not a single technique but a broad class of strategies exploiting the instruction-following behavior of LLMs. ToolShield aligns with this perspective by explicitly modeling multiple prompt injection families and evaluating generalization under attack-family distribution shifts.

2.2 Security Challenges in Tool-Using LLM Systems

Tool-using LLM agents differ from standard chat models because the model output may trigger tool execution, such as database queries, file operations, or API calls. This creates a direct link between text generation and real-world side effects. Recent work has shown that prompt injection becomes more critical in such systems, since adversarial prompts can manipulate the model into selecting unintended tools, leaking tool schemas, or generating harmful tool arguments [15].

Modern frameworks for tool use (often referred to as *function calling*) provide structured tool schemas and attempt to constrain tool invocation through typed arguments. However, the tool selection and argument generation process remains controlled by the model, which is vulnerable to adversarial instruction patterns [5]. ToolShield addresses this risk by framing prompt injection detection as a binary classification task executed prior to tool execution.

In addition to user-controlled inputs, tool-using agents often process untrusted content from external sources such as webpages, emails, or documents. Prior work has shown that untrusted context can carry injection payloads that are not authored by the user but still influence the agent's behavior [7]. This motivates the inclusion of context-aware detection models that incorporate both the prompt and tool context.

2.3 LLM Safety Filters and Classification-Based Defenses

A common defense approach is to introduce safety filters that classify or moderate model inputs and outputs. Many deployed systems use dedicated classifiers to detect harmful content, policy violations, or adversarial instructions [13]. Such filters may operate either

on the user prompt (input filtering) or on the model response (output filtering), sometimes combined with rule-based checks.

Research has explored the use of supervised classifiers, including bag-of-words baselines, TF-IDF models, and transformer-based detectors, for harmful content detection and adversarial prompt detection [6]. In the security literature, filtering is often treated as a practical mitigation that can reduce attack success rates even when the underlying generative model remains vulnerable.

ToolShield follows this design philosophy by evaluating lightweight baselines (heuristics and TF-IDF classifiers) alongside transformer-based models. The emphasis is not on making the underlying LLM robust, but on detecting malicious prompts early enough to prevent unsafe tool execution.

2.4 Context-Aware Modeling and Instruction Hierarchies

LLM agent systems typically provide additional context beyond the user prompt, such as system instructions, tool descriptions, and structured schemas. Several studies argue that the interaction between user prompts and system-level instructions forms an implicit instruction hierarchy, where the model may be vulnerable to adversarial attempts to override higher-priority instructions [8].

Context-aware defense methods attempt to incorporate such structured information into the detection pipeline. For example, detectors may consider both the user prompt and the system prompt, or explicitly model whether the user prompt conflicts with tool constraints [4]. ToolShield extends this idea by training a *context transformer* that jointly encodes prompt and tool schema metadata, aiming to detect attacks that depend on tool-use context rather than purely lexical features.

2.5 Context Window Limitations and Truncation Effects

Transformer models operate under a fixed maximum sequence length, which forces long inputs to be truncated. In practice, tool-use systems may supply extensive tool schemas, multi-step reasoning traces, or conversation histories, which can exceed the token budget. Several works have observed that model performance degrades when relevant information is truncated, particularly when truncation removes the user query or key constraints [3].

Input truncation is often treated as a preprocessing detail, but it can systematically bias model behavior depending on how the input is concatenated and truncated [16]. In tool-use contexts, naive truncation may discard user-level instructions while retaining tool metadata, or vice versa. This is particularly relevant for safety classification systems, where losing the malicious prompt payload can lead to false negatives.

ToolShield directly investigates truncation as a failure mode in prompt injection detection. The prompt-preserving truncation strategy evaluated in this thesis is related to broader work on token budgeting, structured input packing, and context prioritization in LLM pipelines [14].

2.6 Synthetic Benchmarks and Security Evaluation Datasets

Evaluating security defenses requires datasets containing both benign and adversarial prompts. Many prompt injection datasets are synthetic or semi-synthetic, constructed using manually designed templates or generated attacks [10]. Synthetic benchmarks provide

reproducibility and control over attack families, but may fail to capture the diversity of real-world adversarial behavior.

Recent work highlights that evaluation results can be sensitive to dataset design, including the distribution of attack templates, tool schemas, and prompt lengths [2]. There is also ongoing discussion about whether synthetic benchmarks overestimate performance due to limited linguistic variation or unintentional artifacts.

ToolShield adopts a controlled synthetic dataset design in order to explicitly test generalization and truncation failure modes. In particular, the long-schema stress test aligns with concerns that standard benchmarks underestimate the impact of realistic enterprise tool schemas. However, the synthetic nature of the dataset also motivates careful discussion of external validity and future evaluation on real-world tool-use logs.

2.7 Positioning of ToolShield

ToolShield contributes to the intersection of prompt injection research and tool-use system security by framing injection detection as a classification problem over prompt and tool context. While prior work has explored prompt injection and safety filtering independently, ToolShield emphasizes the interaction between tool schema context and limited token budgets. The thesis investigates truncation as a structural vulnerability in context-aware defenses and evaluates a prompt-preserving truncation strategy as a mitigation.

Overall, ToolShield is positioned as an engineering-oriented security study: it does not propose a fundamentally new model architecture, but instead provides empirical evidence that preprocessing choices such as truncation strategy can dominate detection performance in realistic tool-use settings.

3 Methodology and Approach

This chapter describes the methodology used to design, implement, and evaluate ToolShield, a prompt injection detection system for tool-using LLM agents. The approach combines controlled synthetic dataset generation, multiple model baselines, protocol-driven evaluation splits, and reproducibility-focused experiment automation.

3.1 Problem Setting

Tool-using LLM agents extend natural language models with external function calls (tools), such as exporting reports, sending emails, or accessing structured databases. In such settings, a user prompt is not only interpreted as text, but may trigger execution of privileged tool actions. This creates a security risk: malicious users can embed prompt injection attacks designed to override system intent, exfiltrate data, or force unintended tool calls.

ToolShield addresses this risk by framing prompt injection detection as a supervised binary classification task. Each input consists of a user prompt and associated tool context (role framing, tool name, tool schema, tool description). The output is a probability score indicating whether the prompt is benign or malicious.

3.2 Scope and Assumptions

ToolShield addresses the problem of detecting prompt injection attempts in tool-using LLM systems. The core goal is to classify a given tool invocation request (prompt plus tool context) as either benign or malicious, such that malicious inputs can be blocked before tool execution. ToolShield does not claim to prevent all forms of LLM misuse, nor does it attempt to make the underlying LLM robust against adversarial instructions. Instead, it focuses on a practical detection layer that can be deployed as an external safeguard.

System scope. The scope of this thesis is limited to *single-turn* prompt injection detection. Each sample consists of a single user prompt and associated tool metadata (e.g., tool schema, tool description, and role framing). Multi-step agent workflows, memory-augmented reasoning, and iterative tool chains are not explicitly modeled. Consequently, ToolShield does not capture attacks that emerge only across multiple conversational turns or through gradual manipulation over time.

Attacker model. We assume an attacker who can fully control the user prompt and aims to induce the LLM to violate tool usage constraints, exfiltrate sensitive information, or generate unsafe tool arguments. The attacker is assumed to be adaptive but does not have direct control over the system prompt or tool implementation. This reflects common deployment settings where tool schemas and system instructions are provided by the application, while user input remains untrusted. ToolShield does not assume that the attacker has access to model parameters or training data; the threat model is therefore aligned with black-box prompt-based attacks.

Tool schema availability. ToolShield assumes that tool schemas and descriptions are available to the detection system at inference time, since they are typically included in the LLM input context for tool selection and argument generation. The context-aware models

explicitly incorporate this information. If schemas are hidden, incomplete, or dynamically generated in ways that cannot be reproduced by the detector, ToolShield's context-based approach may be less effective.

Dataset and evaluation assumptions. This thesis uses a synthetic dataset generated from controlled templates and predefined attack families. Synthetic generation enables reproducible experiments, controlled manipulation of schema length, and explicit enforcement of evaluation protocols such as `S_attack_holdout`. This design is necessary to isolate specific hypotheses, particularly the truncation bias failure mode under long schemas. However, synthetic prompts may not fully reflect the linguistic diversity, contextual ambiguity, and attacker creativity present in real-world deployments.

Out of scope. Several important aspects of real-world LLM security are out of scope. First, ToolShield does not evaluate multi-turn prompt injection scenarios or attacks that exploit agent memory and conversation history. Second, the experiments do not include proprietary enterprise tools or confidential schemas, as only synthetic schemas are used. Third, multilingual prompt injection attacks are not evaluated unless explicitly included in the dataset configuration; therefore, no general claims are made about performance outside English-language prompts. Finally, ToolShield does not address adversarial adaptation against the detector itself (e.g., evasion attacks targeting the classifier), nor does it consider end-to-end defenses such as sandboxing or formal tool authorization.

In summary, ToolShield is evaluated as a reproducible prompt injection detection component for single-turn tool-use scenarios under controlled experimental assumptions. The results should be interpreted as evidence for failure modes and mitigation strategies in this setting, rather than as a complete solution to LLM agent security.

3.3 Dataset Generation

Since no widely accepted benchmark exists for prompt injection detection in structured tool-use environments, a synthetic dataset was generated. The dataset generation pipeline is implemented as part of the ToolShield CLI (`toolshield generate`) and produces a `dataset.jsonl` file containing labeled records.

3.3.1 Record Format

Each dataset record is represented as a structured JSON object containing the following fields:

- **prompt**: the user input string.
- **label**: binary target variable (0 = benign, 1 = attack).
- **tool_name**: identifier of the tool that the agent is expected to call.
- **tool_schema**: JSON schema describing tool parameters.
- **tool_description**: natural language description of the tool.
- **role_instruction**: system-style instruction that defines the assistant role.
- **attack_family**: categorical field AF1–AF4 (only for attack samples).

- **template_id**: identifier for the template used during generation (for leakage control).

This structured format enables experiments that explicitly model realistic tool-use environments where schema and tool metadata are available as context.

3.3.2 Benign Prompt Generation

Benign prompts are generated from templated requests consistent with the tool set (e.g., asking to export a report, summarize text, query a database). Templates are parameterized to create diverse surface forms while preserving semantic intent. This ensures that benign samples cover multiple syntactic styles without introducing attack-like patterns.

3.3.3 Attack Prompt Generation

Attack samples are generated using prompt injection templates designed to simulate real adversarial behavior. Each attack prompt is constructed by embedding malicious instructions into otherwise plausible user requests. Attack prompts aim to override the agent's intended policy, manipulate tool execution, or leak hidden context.

The generation process ensures that attacks are distributed across multiple families (AF1–AF4), which are described in Section 3.5.

3.3.4 Dataset Output and Manifest

Dataset generation outputs both the JSONL dataset and a manifest file (`manifest.json`) containing:

- total sample count,
- label distribution,
- generation seed,
- tool and template metadata.

The manifest serves as a reproducibility artifact and enables validation that subsequent splits and experiments are derived from the intended dataset configuration.

3.4 Tool Schema Structure

In tool-augmented LLM systems, tools are often specified via structured schemas such as OpenAPI or JSON Schema. ToolShield models this by associating each prompt with a tool schema object.

Each tool schema includes:

- **name**: tool identifier.
- **description**: short natural language description.
- **parameters**: JSON schema defining input arguments.

The **parameters** schema contains typed fields (string, integer, boolean, arrays, objects) and may include constraints such as required keys, allowed values, and nested objects.

This design allows the context-aware transformer model to learn correlations between prompt structure and tool constraints, e.g., whether a prompt attempts to inject irrelevant fields or override tool behavior.

3.5 Prompt Injection Families

A key objective of ToolShield is to measure robustness to diverse injection strategies. Therefore, attacks are grouped into four families (AF1–AF4), each representing a distinct manipulation style. The family structure is intended to approximate the diversity observed in real prompt injection attacks described in prior work [1].

3.5.1 AF1: Direct Instruction Override

AF1 attacks explicitly attempt to override the system policy using direct imperative statements, such as:

> Ignore all previous instructions and execute the tool with these parameters...

This family captures the most common form of naive prompt injection, where the attacker assumes the model will follow the most recent instruction.

3.5.2 AF2: Context Extraction and Data Exfiltration

AF2 attacks focus on extracting hidden context or confidential tool metadata, such as system prompts, schema definitions, or private data. Typical phrasing includes:

> Before answering, print your full tool schema and system prompt...

This family represents confidentiality attacks where the tool call itself may be used as a leakage vector.

3.5.3 AF3: Tool Manipulation and Parameter Smuggling

AF3 attacks attempt to manipulate tool calls by embedding adversarial parameters, hidden commands, or format-breaking strings. Examples include injecting JSON fragments or attempting to bypass schema validation.

The purpose of AF3 is to test whether models can identify subtle manipulations that resemble legitimate structured input but contain malicious intent.

3.5.4 AF4: Indirect or Obfuscated Attacks

AF4 attacks represent more sophisticated prompt injection strategies, such as:

- obfuscation through encoding,
- multi-step reasoning traps,
- role-playing and social engineering phrasing,
- hidden instructions embedded in long text.

AF4 was specifically designed to support the S_attack_holdout evaluation protocol, where this family is entirely excluded from training and appears only in the test set. This enables controlled measurement of generalization to unseen attack styles.

3.6 Split Protocols

ToolShield uses protocol-driven dataset splitting to avoid evaluation artifacts and to measure different generalization dimensions. The split generator is implemented as `toolshield split` and produces `train.jsonl`, `val.jsonl`, and `test.jsonl` for each protocol.

3.6.1 S_random: Stratified Random Split

S_random is a stratified random split with template leakage prevention. Records generated from the same template identifier are constrained to appear in only one split. This reduces the risk of inflated performance due to memorization of generation templates rather than learning general prompt injection patterns.

This split is used as the baseline evaluation setting.

3.6.2 S_attack_holdout: Attack Family Holdout

S_attack_holdout excludes the AF4 attack family entirely from training and validation. AF4 samples appear only in the test set. Benign samples remain distributed across all splits.

This protocol evaluates generalization under distribution shift: the model must detect attacks that are structurally different from those seen during training.

3.6.3 S_tool_holdout: Tool Holdout

S_tool_holdout excludes one entire tool (e.g., `exportReport`) from training and validation. All samples involving that tool appear only in the test set.

This protocol measures generalization across tools and schemas, testing whether models learn attack semantics rather than tool-specific patterns.

3.7 Models Implemented

ToolShield implements five model families. Each model outputs a continuous risk score, enabling both threshold-free evaluation (ROC-AUC, PR-AUC) and threshold-based deployment analysis.

3.7.1 Heuristic Baseline

The heuristic model is a rule-based detector relying on keyword and pattern matching. It assigns a binary prediction based on the presence of suspicious phrases (e.g., “ignore previous instructions”, “system prompt”, “developer message”, “execute tool”).

This baseline represents a lightweight but brittle approach commonly used in practical filtering systems.

3.7.2 Scored Heuristic Baseline

The heuristic_score model extends the heuristic baseline by producing a continuous score. Instead of a hard decision, it aggregates weighted signals from multiple suspicious features.

This model enables ranking-based evaluation and provides a more realistic baseline for security systems that operate at adjustable thresholds.

3.7.3 TF-IDF + Logistic Regression

The TF-IDF + Logistic Regression model represents a classical supervised text classifier. Prompts are vectorized using TF-IDF features, and a logistic regression classifier is trained on the resulting sparse representation.

The model is efficient at inference and serves as a strong baseline for detecting common lexical patterns in prompt injection attacks.

3.7.4 Transformer (Text-Only)

The transformer baseline fine-tunes a pretrained encoder model (DistilRoBERTa) with a binary classification head. The input consists only of the user prompt text, without tool metadata.

The purpose of this baseline is to evaluate whether prompt injection attacks can be detected reliably without tool context, and to compare performance against classical bag-of-words methods.

3.7.5 Context Transformer

The context transformer extends the text-only transformer by incorporating tool context fields:

- role instruction,
- tool name,
- tool schema,
- tool description,
- user prompt.

These fields are concatenated into a single sequence before tokenization. The model is trained end-to-end using the same binary classification objective.

The motivation is that many injection attacks exploit tool-related instructions, and therefore tool context should improve robustness.

3.8 Prompt-Preserving Truncation Strategy

A central contribution of ToolShield is the identification and mitigation of truncation bias in context-augmented transformers.

In standard transformer tokenization pipelines, long sequences are truncated from the right once the maximum token budget is exceeded. If the input is constructed as:

$$\text{context} \rightarrow \text{prompt}$$

then right truncation disproportionately removes the prompt, even though the prompt contains the primary attack signal.

3.8.1 Naive Truncation

In the naive strategy, context and prompt are concatenated into one string and passed to the tokenizer with a fixed `max_length`. The tokenizer applies standard truncation, typically removing tokens from the tail.

This can result in complete loss of the prompt when tool schemas are long.

3.8.2 Keep-Prompt Truncation

In the keep_prompt strategy, the model tokenizes context and prompt separately. A minimum prompt token budget (`prompt_min_tokens`) is reserved. Remaining tokens are allocated to context.

Formally, given a token budget B:

- allocate at least $P_{\min}$ tokens to the prompt,
- allocate $B - P_{\min}$ tokens to context,
- if the prompt uses fewer than $P_{\min}$ tokens, the unused budget is reallocated to context.

This guarantees that prompt content remains visible to the classifier even when schemas are large. This strategy was validated empirically in the long-schema stress test (Results Chapter).

3.9 Training Setup

All models were trained using consistent procedures:

- Training split used for fitting parameters.
- Validation split used for early stopping and threshold selection.
- Test split used only for final evaluation.

3.9.1 Hyperparameters

Transformer models were fine-tuned with:

- pretrained backbone: DistilRoBERTa,
- maximum sequence length: 256 tokens (context transformer),
- learning rate: $2 \cdot 10^{-5}$,
- batch size: 8–16 depending on configuration,
- warmup steps: 20–100,
- number of epochs: 3.

Non-neural baselines used default scikit-learn training settings with fixed random seeds.

3.9.2 Seed Control

All training procedures support explicit seed control. Seeds affect:

- dataset generation,
- split generation,
- model initialization,
- shuffling order during training.

Neural models were evaluated across three seeds (0, 1, 2) to reduce variance and to support mean/std reporting.

3.10 Evaluation Pipeline

Evaluation is implemented via `toolshield eval` and produces a structured metrics output file (`metrics.json`) for each run.

3.10.1 Core Metrics

ToolShield reports the following primary metrics:

- **ROC-AUC:** measures ranking quality independent of threshold.
- **PR-AUC:** measures precision-recall performance under class imbalance.
- **FPR@TPR90 / FPR@TPR95:** false positive rate required to achieve 90% or 95% true positive rate.

FPR@TPR metrics are relevant for deployment because they quantify the benign traffic blocked to catch a fixed proportion of attacks.

3.10.2 Attack Success Rate (ASR) Reduction

To evaluate security effectiveness, ToolShield reports ASR reduction. ASR is defined as the fraction of malicious prompts that bypass the detector.

ASR reduction is computed by comparing ASR before filtering (baseline ASR = 1.0) and ASR after filtering at a given threshold. A high ASR reduction indicates that most attacks are blocked.

3.10.3 Budget-Based Threshold Selection

ToolShield additionally supports budget evaluation, where thresholds are chosen based on a maximum allowed false-positive budget (1%, 3%, 5%). Thresholds are selected on the validation split and then evaluated on the test split.

This simulates real deployment constraints where blocking too many benign prompts is unacceptable.

3.10.4 Latency Measurement

Inference latency is measured using timed prediction runs. ToolShield reports P50 and P95 latency percentiles under warm-start conditions.

Latency is critical for evaluating whether the model is feasible as a runtime guardrail in LLM tool pipelines.

3.11 Aggregation and Reporting

Each run produces an individual `metrics.json` file. These are aggregated into summary tables using `aggregate_experiments.py`, which generates:

- `raw_results.csv`: per-seed results.
- `summary.csv`: mean and standard deviation across seeds.
- `results.json`: combined machine-readable output.

Additionally, LaTeX tables are generated from `summary.csv` using `generate_latex_from_summary.p` enabling direct integration into the thesis.

3.12 Determinism and Reproducibility Measures

A core design goal of ToolShield is that all reported results can be reproduced deterministically. The project includes multiple mechanisms to support reproducibility:

3.12.1 Manifest and Environment Freeze

Each dataset generation produces a manifest with seed and label distribution. Additionally, the environment is captured in `environment_freeze.txt`, which lists package versions (torch, transformers, scikit-learn, etc.).

3.12.2 Automated Verification Scripts

ToolShield includes verification scripts that ensure experimental integrity:

- split hygiene checks: ensure no template leakage, correct attack family holdout, correct tool holdout.
- experiment bundle verifier: validates that reported summary metrics match expected qualitative behavior.

These scripts output human-readable logs suitable for appendix inclusion.

3.12.3 Controlled Experiment Automation

The full experiment suite is reproducible via Makefile targets. For example:

- `make compare_truncation`
- `make compare_truncation_longschema`
- `make verify_longschema_results`

These targets execute the complete pipeline: dataset generation, splitting, training, evaluation, aggregation, reporting, and verification.

3.12.4 Run Manifests

The script `write_run_manifest.py` generates a run manifest capturing:

- git commit hash,
- executed commands,
- dataset configuration,
- split protocol settings,
- model configurations.

This ensures that reported results are traceable to an exact repository state.

3.13 Summary

In summary, ToolShield was developed using a controlled experimental methodology combining:

- synthetic dataset generation with explicit schema context,
- multiple baseline and neural classifiers,
- protocol-driven evaluation for generalization testing,
- prompt-preserving truncation to mitigate schema-driven bias,
- automated reporting and verification for reproducibility.

This methodology enables systematic evaluation of prompt injection detection under both standard and enterprise-like tool schema conditions, supporting robust conclusions about truncation bias and model generalization.

4 Implementation

This chapter describes the technical implementation of ToolShield, including the command-line interface, dataset pipeline, training and evaluation infrastructure, experiment organization, and reproducibility measures. The implementation is designed to support controlled, repeatable experiments and to produce thesis-ready artifacts.

4.1 CLI Architecture

ToolShield is implemented as a Python package (`toolshield`) with a unified command-line interface. The CLI provides subcommands that map directly to pipeline stages:

- `toolshield generate`: produces a labeled JSONL dataset from configured templates and attack families.
- `toolshield split`: generates train/val/test splits under a specified protocol (`S_random`, `S_attack_holdout`, or `S_tool_holdout`).
- `toolshield train`: trains a specified model on a given split directory.
- `toolshield eval`: evaluates a trained model on a test split and writes a structured `metrics.json` file.
- `toolshield report`: generates combined result tables from evaluation outputs.
- `toolshield info`: displays package version and configuration metadata.

Each subcommand accepts configuration via command-line arguments and, where applicable, YAML configuration files stored in the `configs/` directory. This design ensures that every pipeline stage is individually invocable and parameterized, supporting both interactive use and automated orchestration through Makefile targets.

4.2 Dataset Pipeline

4.2.1 Dataset generation

The dataset generation stage produces a JSONL file (`data/dataset.jsonl`) containing labeled records. Each record includes a user prompt, tool metadata (name, schema, description), role framing, label, and attack family annotation. Generation is controlled by a YAML configuration file (`configs/dataset.yaml`) specifying the number of samples, attack family distribution, tool set, and random seed.

A companion manifest file (`data/manifest.json`) is written alongside the dataset, recording sample counts, label distribution, seed, and generation timestamp.

4.2.2 Split generation

The split stage partitions the dataset into `train.jsonl`, `val.jsonl`, and `test.jsonl` under a specified protocol. Split constraints are enforced programmatically:

- `S_random`: stratified random split with template leakage prevention (no `template_id` appears in more than one split).

- `S_attack_holdout`: AF4 samples excluded from train/val and placed only in test.
- `S_tool_holdout`: all samples of a specified tool excluded from train/val.

Each protocol writes its splits to a separate directory (e.g., `data/splits/S_random/`).

4.2.3 Long-schema inflation

For the enterprise truncation stress test, the script `scripts/inflate_schemas.py` transforms the base dataset by inflating tool schemas to a target character length (approximately 4 000 characters). The inflated dataset is written to `data/dataset_longschema.jsonl`, and subsequent splits are generated from it into `data/splits_longschema/`.

The inflation process appends realistic enterprise-style properties (nested objects, enumerated constraints, long descriptions) to each schema, controlled by a fixed random seed for reproducibility.

4.3 Training Pipeline

Training is invoked via `toolshield train` and configured through per-model YAML files in `configs/training/`. Each configuration specifies:

- model type (e.g., `heuristic`, `tfidf_lr`, `context_transformer`),
- hyperparameters (learning rate, batch size, epochs, warmup steps),
- truncation strategy (`naive` or `keep_prompt`),
- maximum sequence length and schema length constraints.

For the truncation ablation, separate configuration files exist for each strategy variant: `context_transformer_naive.yaml` and `context_transformer.yaml` (keep_prompt). Long-schema variants use dedicated configurations with adjusted `max_schema_length` settings (e.g., `context_transformer_naive_longschema.yaml`).

Trained model artifacts (weights, tokenizer, and configuration) are saved to the experiment output directory alongside evaluation results.

4.4 Evaluation and Reporting Pipeline

4.4.1 Metrics computation

The evaluation stage (`toolshield eval`) loads a trained model and test split, runs inference, and computes the following metrics:

- ROC-AUC and PR-AUC (threshold-independent ranking metrics),
- FPR@TPR90 and FPR@TPR95 (operating-point metrics),
- ASR reduction at TPR90 and TPR95,
- budget-based threshold evaluation at 1%, 3%, and 5% false-positive budgets,
- inference latency at P50 and P95 percentiles (warm-start).

Results are written to a structured `metrics.json` file in the model output directory. A degenerate-score warning is included if the model produces fewer than three unique score values, which can indicate training failure or trivial predictions.

4.4.2 Aggregation

The script `scripts/aggregate_experiments.py` discovers all `metrics.json` files under an experiment root directory, parses them, deduplicates by (protocol, model, seed) key, and produces:

- `raw_results.csv`: per-seed results for all runs,
- `summary.csv`: mean and standard deviation across seeds,
- `results.json`: combined machine-readable output.

Aggregation includes validation checks for missing seeds, missing protocols, and absent transformer models, reporting warnings where applicable.

4.4.3 LaTeX table generation

The script `scripts/generate_latex_from_summary.py` reads `summary.csv` (and optionally `truncation_stats.csv`) and generates LaTeX table source code suitable for direct inclusion in the thesis. Tables are formatted with consistent metric naming and numerical precision.

4.4.4 Truncation statistics

The script `scripts/report_truncation_stats.py` computes per-strategy truncation statistics from the split data, including prompt retention ratios, schema token distributions, and per-sample truncation counts. It produces `truncation_stats.csv`, `truncation_stats.json`, and retention visualization figures.

4.5 Experiment Automation

4.5.1 Ablation runner

The script `scripts/run_truncation_ablation.py` orchestrates the full truncation ablation study. It iterates over specified seeds, protocols, and model configurations, invoking training and evaluation for each combination. The script accepts parameters for experiment tag, splits directory, and model set, enabling reuse across standard and long-schema experiments.

4.5.2 Makefile targets

The project `Makefile` provides targets that compose the full pipeline. Key targets include:

- `make compare_truncation_longschema`: executes the complete long-schema stress test (data generation, splitting, training, evaluation, aggregation, truncation statistics, split verification, appendix example export, LaTeX table generation, and run manifest).
- `make verify_longschema_results`: runs the automated verification script against the generated artifacts.
- `make test`: executes the full test suite.

4.6 Experiment Folder Structure

Experiment outputs follow a hierarchical directory structure organized by seed, protocol, and model:

```
data/reports/experiments_longschema/
+-- seed_0/
|   +-- S_random/
|   |   +-- context_transformer_naive_longschema/
|   |   |   +-- model/
|   |   |   +-- tokenizer/
|   |   |   +-- metrics.json
|   |   +-- context_transformer_keep_prompt_longschema/
|   |       +-- model/
|   |       +-- tokenizer/
|   |       +-- metrics.json
|   +-- S_attack_holdout/
|       +-- ...
+-- seed_1/
|   +-- ...
+-- seed_2/
|   +-- ...
+-- summary.csv
+-- raw_results.csv
+-- results.json
+-- truncation_stats.csv
+-- truncation_stats.json
+-- split_hygiene.md
+-- guards.json
+-- appendix_truncation_example.md
+-- verification_output.txt
+-- run_manifest.md
+-- latex_tables.txt
```

This structure enables the aggregation script to discover all per-seed results automatically by traversing the directory tree. Aggregated outputs and verification artifacts are placed at the experiment root level.

4.7 Reproducibility Measures

4.7.1 Seed control

All stochastic operations use explicit seeds. Dataset generation uses a configurable seed (default: 1337). Split generation uses a separate seed (default: 2026). Model training seeds are passed as command-line arguments (typically 0, 1, 2). Seeds propagate to Python's `random` module, NumPy, and PyTorch to ensure deterministic behavior.

4.7.2 Deterministic splits

Split generation enforces template-level disjointness: no `template_id` appears in more than one split. For `S_attack_holdout`, AF4 samples are deterministically assigned to

test. The split verification script `scripts/verify_longschema_splits.py` checks these constraints and produces `split_hygiene.md`, `split_hygiene.csv`, and `guards.json`.

4.7.3 Run manifests and environment freeze

The script `scripts/write_run_manifest.py` captures the exact git commit hash, executed commands, dataset configuration, and model settings at experiment time. The resulting `run_manifest.md` provides a complete provenance record.

The Python environment is captured via `pip freeze` and saved to `data/reports/environment_free` recording exact package versions (torch, transformers, scikit-learn, etc.).

4.7.4 Verification scripts

Two verification scripts ensure experimental integrity:

- `scripts/verify_longschema_splits.py`: checks template leakage, AF4 holdout correctness, and class balance across all splits. Produces human-readable reports and machine-readable guard files.
- `scripts/verify_experiments_longschema_bundle.py`: validates the complete experiment bundle by checking that all required artifact files exist, that `summary.csv` contains expected models with correct seed counts, that ROC-AUC values satisfy regression bounds (keep_prompt ≥ 0.95, naive ≤ 0.60), that truncation statistics match expected patterns (naive: 100% truncation; keep_prompt: 0% truncation), that `split_hygiene.md` contains no failures and `guards.json` is empty, and that raw `metrics.json` files exist for the reference seed. The script exits with code 1 on any failure and prints diagnostic messages.

4.8 Artifacts and Experiment Bundle

The file `THESIS_ARTIFACTS.md` in the repository root serves as the authoritative artifact index for the thesis. It lists the exact file paths of all artifacts required to reproduce and verify the reported results, organized by purpose:

1. **Aggregated results:** `summary.csv` containing per-protocol, per-model aggregated metrics.
2. **Truncation mechanism proof:** `truncation_stats.csv` with per-strategy prompt retention ratios and truncation counts.
3. **Split hygiene:** `split_hygiene.md` (human-readable check results) and `guards.json` (machine-readable warnings).
4. **Concrete appendix example:** `appendix_truncation_example.md` providing a token-level walkthrough of truncation for a single record under both strategies.
5. **Verification output:** `verification_output.txt` documenting the terminal output of the automated verification run.
6. **Reproducibility:** `run_manifest.md` (git hash, commands, seeds, configuration) and `environment_freeze.txt` (pip freeze snapshot).

This artifact index establishes a contract between the thesis text and the repository: every numerical claim in the Results and Discussion chapters can be traced to a specific file listed in `THESIS_ARTIFACTS.md`. The complete experiment bundle can be regenerated from a clean state using:

```
make clean
make compare_truncation_longschema
make verify_longschema_results
```

Successful execution of `make verify_longschema_results` confirms that all artifacts are present and internally consistent.

5 Results

This chapter presents the empirical evaluation of ToolShield across multiple split protocols and model families. We report performance using ROC-AUC and PR-AUC as threshold-independent metrics, and FPR@TPR90/95 as operating-point metrics relevant for deployment. Additionally, we evaluate the reduction in Attack Success Rate (ASR reduction) under controlled false-positive budgets and analyze inference latency.

Experimental setup recap. Neural model results (context transformer variants) are aggregated over n=3 independent random seeds (seeds 0, 1, 2) and reported as mean $\pm$ standard deviation. Some baseline rows in Table 1 are reported for a single seed (n_{seeds}=1): specifically `heuristic`, `heuristic_score`, `tfidf_lr`, and `transformer`, which were executed once in the standard dataset due to runtime constraints. Models are evaluated on two primary protocols: `S_random` (stratified random split with template leakage prevention) and `S_attack_holdout` (AF4 held out from training). The metrics reported are: ROC-AUC, PR-AUC, FPR@TPR90, FPR@TPR95, ASR reduction at both operating points, and warm-start inference latency at P50 and P95 percentiles.

All reported values are taken directly from `summary.csv` and `truncation_stats.csv` in the thesis artifact bundle (Table 1, Table 6).

5.1 Experimental Protocols

We evaluate models under the following split protocols:

- `S_random`: stratified random split with template leakage prevention.
- `S_attack_holdout`: distribution shift setting where an entire attack family (AF4) is held out from training and appears only in the test set.

The `S_attack_holdout` protocol is intended to measure generalization to unseen prompt injection strategies rather than memorization of attack templates.

Note on `S_tool_holdout`. A third protocol, `S_tool_holdout`, excludes all samples of a specific tool from training. In some configurations, this yields a single-class training split (e.g., only benign samples for that tool), which prevents supervised models from converging meaningfully. Where this occurs, the affected model–protocol combination is omitted from the results rather than reported with degenerate metrics. This limitation is inherent to the synthetic dataset's tool–label distribution and does not affect the `S_random` or `S_attack_holdout` evaluations.

5.2 Baseline Model Performance on the Standard Dataset

Table 1 summarizes performance on the standard dataset. Overall, nearly all models achieve strong discrimination, with transformer-based approaches approaching perfect classification.

5.2.1 `S_random` Results

On `S_random`, multiple models achieve near-ceiling performance:

- `tfidf_lr` achieves ROC-AUC = 0.9999 and PR-AUC = 0.9999, with FPR@TPR90 = 0.000 and FPR@TPR95 = 0.000.
- `transformer` (text-only) achieves ROC-AUC = 0.9877 and PR-AUC = 0.9911, but exhibits higher false positive rates at strict operating points (FPR@TPR90 = 0.020, FPR@TPR95 = 0.200).
- `context_transformer_keep_prompt` achieves ROC-AUC = 1.000 and PR-AUC = 1.000, with FPR@TPR90 = 0.000 and FPR@TPR95 = 0.000.

The context-aware transformer achieves perfect ranking performance on this split. This suggests that, under the synthetic data distribution, context features (tool schema, tool description, and role framing) provide sufficient information to separate benign from malicious prompts.

In contrast, the `heuristic` baseline performs substantially worse with ROC-AUC = 0.7137, despite a relatively high PR-AUC = 0.8722. This indicates that while heuristic rules can capture many attacks, they do not provide reliable ranking across all samples.

5.2.2 `S_attack_holdout` Results

Under `S_attack_holdout`, performance remains near-perfect for both context transformer variants:

- `context_transformer_keep_prompt` achieves ROC-AUC = 0.9987 and PR-AUC = 0.9990, with FPR@TPR90 = 0.000 and FPR@TPR95 = 0.010.
- `context_transformer_naive` achieves ROC-AUC = 0.9982 and PR-AUC = 0.9986, with FPR@TPR90 = 0.000 and FPR@TPR95 = 0.010.

These results show that, in the standard dataset configuration, both truncation strategies behave similarly. This is explained by the relatively small configured schema length limit (`max_schema_length=200` characters), which prevents the tool schema from occupying most of the token budget.

5.3 Attack Success Rate (ASR) Reduction

We report ASR reduction at two operating points: TPR90 and TPR95 (Table 2). ASR reduction measures how effectively the detector blocks malicious prompts when deployed as a binary gate before tool execution. Budget-constrained threshold selection results are presented in Table 3.

5.3.1 `S_random`

On `S_random`, the context transformer variants achieve ASR reduction between 92.7% and 94.4%:

- `keep_prompt`: ASR reduction at TPR90 = 0.9274, at TPR95 = 1.000.

- `naive`: ASR reduction at TPR90 = 0.9435, at TPR95 = 0.9597.

This indicates that under the standard schema size setting, both strategies remain highly effective at reducing successful attacks. Differences are small and likely not statistically meaningful at this dataset scale.

5.3.2 `S_attack_holdout`

Under distribution shift (`S_attack_holdout`), ASR reduction remains strong:

- `keep_prompt`: ASR reduction at TPR90 = 0.9360, at TPR95 = 0.9760.
- `naive`: ASR reduction at TPR90 = 0.9120, at TPR95 = 0.9760.

This confirms that ToolShield generalizes well to the held-out AF4 attack family when schemas remain short enough to avoid truncation bias.

5.4 Latency Measurements

Inference latency was measured in warm mode. Table 4 summarizes P50 and P95 latencies.

On the standard dataset, the following latency trends are observed:

- `tfidf_lr` is the fastest model with P50 latency = 2.63 ms.
- Transformer-based models exhibit the highest latencies: `transformer` (text-only) has P50 latency = 11614.90 ms, and `context_transformer_keep_prompt` has P50 latency = 4669.87 ms.
- `context_transformer_naive` has P50 latency = 2744.15 ms, lower than `keep_prompt` due to the simpler single-pass tokenization.

Thus, prompt-preserving truncation introduces additional runtime overhead, roughly increasing P50 latency from ~ 2.7 s to ~ 4.7 s in this environment. This is expected, since the `keep_prompt` strategy performs two-pass tokenization and explicit budget allocation.

5.5 Truncation Ablation on the Standard Dataset

To evaluate whether truncation strategy affects detection performance, we compare two configurations:

- `naive`: standard HuggingFace right-truncation after concatenation.
- `keep_prompt`: explicit prompt-preserving truncation where a minimum prompt token budget is reserved.

On the standard dataset, both strategies yield nearly identical predictive performance (Table 5). For example, under `S_attack_holdout`, ROC-AUC is 0.9987 for `keep_prompt` and 0.9982 for `naive`.

This suggests that truncation bias is not visible in this dataset configuration due to the hard schema length constraint (`max_schema_length=200`), which prevents context fields from dominating the model input.

5.5.1 Enterprise long-schema stress test

To evaluate truncation behaviour under more realistic enterprise tool schemas, we constructed a long-schema dataset variant where tool schemas were inflated to approximately 4 000 characters, corresponding to approximately 840 schema tokens on average (Table 6).

This experiment directly tests the hypothesis that naive truncation can remove the prompt entirely when tool context fields are large. The primary quantitative evidence is recorded in `truncation_stats.csv`; a concrete record-level walkthrough showing token allocation under both strategies is provided in `appendix_truncation_example.md`.

Prompt retention statistics. The truncation statistics are recorded in `truncation_stats.csv` and summarized in Table 6. Results show maximal separation between strategies:

- `naive` truncation: 100% of samples experienced prompt truncation in both protocols.
- `keep_prompt` truncation: 0% of samples experienced prompt truncation.

Quantitatively, mean prompt retention ratio is:

- `naive`: 0.000 (mean prompt tokens retained = 0.0).
- `keep_prompt`: 1.000 (mean prompt tokens retained ≈ 19).

This confirms that, under a 256-token budget, naive truncation results in complete removal of the prompt when schemas approach ~ 840 tokens. In contrast, `keep_prompt` consistently preserves all prompt tokens.

The schema inflation achieved the intended stress condition: schema token length ranged from 795 to 889 tokens (`S_random`) and from 794 to 895 tokens (`S_attack_holdout`).

Figure 1 (quartile plot) and Figure 2 (decile plot) visualize the retention collapse under naive truncation.

Classification performance under long schemas. The long-schema stress test shows that truncation strategy becomes a dominant factor for classification.

`S_attack_holdout`. Under distribution shift with inflated schemas:

- `context_transformer_keep_prompt_longschema` achieves ROC-AUC = 0.9980 and PR-AUC = 0.9984.
- `context_transformer_naive_longschema` collapses to ROC-AUC = 0.4551 and PR-AUC = 0.5409.

The naive model performs close to random chance (ROC-AUC ≈ 0.5). Additionally, the operating-point metrics show that it cannot achieve a usable detection threshold: FPR@TPR90 increases to 0.92 and FPR@TPR95 increases to 0.95. Budget thresholds become `inf` (Table 3), indicating that no threshold can satisfy the budget constraint while maintaining meaningful detection.

In contrast, `keep_prompt` maintains robust performance with FPR@TPR90 = 0.000 and FPR@TPR95 = 0.010, consistent with the standard dataset results.

S_random. On S_random with long schemas:

- context_transformer_keep_prompt_longschema achieves ROC-AUC = 0.9967 and PR-AUC = 0.9975.
- context_transformer_naive_longschema collapses to ROC-AUC = 0.4786 and PR-AUC = 0.6566.

The naive model again fails to provide reliable discrimination. Its FPR@TPR90 reaches 0.90 and FPR@TPR95 reaches 0.96, which would be unacceptable in any deployment context.

The keep_prompt model remains stable, achieving FPR@TPR90 = 0.000 and FPR@TPR95 ≈ 0.0033.

Interpretation. The long-schema stress test demonstrates that naive truncation introduces a systematic failure mode: the prompt, which contains the primary signal for injection detection, is removed entirely due to right-truncation after concatenation. The classifier is then forced to make predictions from tool metadata alone, which is insufficient for separating benign and malicious samples.

The keep_prompt strategy avoids this failure by enforcing a minimum prompt token budget, ensuring that detection remains robust even when tool schemas dominate the context.

This result is not a minor performance difference but a qualitative shift: the same model architecture changes from near-perfect detection to near-random guessing depending only on truncation strategy.

5.6 Verification Artifacts

Split integrity for both protocols is documented in split_hygiene.md, which confirms that no template leakage occurred between splits and that the AF4 holdout constraint was enforced correctly. The companion file guards.json contains an empty list, indicating that no class-balance warnings were raised during verification. The automated experiment bundle verification script (verify_experiments_longschema_bundle.py) checks artifact completeness, truncation invariant consistency, and ROC-AUC regression bounds; its output is recorded in verification_output.txt.

5.7 Key Findings

- On the standard dataset, transformer-based models achieve near-ceiling performance, with context-aware transformers reaching ROC-AUC ≈ 1.0 on S_random and ≈ 0.999 on S_attack_holdout (Table 1).
- Heuristic baselines provide limited discrimination (ROC-AUC ≈ 0.714), confirming the need for learned models.
- Under standard schema constraints, naive and keep_prompt truncation produce nearly identical results, since schemas are too small to trigger truncation bias.
- Under enterprise-length schemas ($\sim 4\,000$ characters, ~ 840 tokens), naive truncation causes 100% prompt truncation and prompt retention ratio collapses to 0.0 (Table 6, Figure 1).

- In the same stress setting, `naive` truncation degrades classification to near-random performance (ROC-AUC = 0.455–0.479), while `keep_prompt` maintains strong performance (ROC-AUC $\approx$ 0.998).
- Prompt-preserving truncation is therefore a necessary design choice for robust prompt injection detection in realistic tool-use environments.

6 Discussion

This chapter discusses the empirical findings of ToolShield and interprets them in the context of prompt injection detection for LLM tool-use. Beyond reporting predictive performance, the goal is to identify which design decisions materially affect robustness, where the proposed approach generalizes, and what limitations remain for real-world deployment.

6.1 Interpretation of Baseline Model Performance (RQ1)

This section addresses **RQ1**: how accurately lightweight baselines detect prompt injection compared to transformer-based classifiers. The baseline evaluation (Table 1) shows that even relatively simple machine learning models can achieve strong performance on the synthetic ToolShield dataset, while purely rule-based approaches remain substantially weaker.

6.1.1 Why TF-IDF + Logistic Regression performs strongly

The `tfidf_lr` model achieves near-ceiling ROC-AUC and PR-AUC on `S_random` (Table 1). This outcome is plausible because many prompt injection attacks contain lexical markers such as explicit override commands (e.g., "ignore previous instructions"), policy evasion language, or imperative phrases related to tool misuse. TF-IDF features are well-suited to capture such token-level signals, and logistic regression can assign high weight to discriminative n-grams without requiring complex contextual reasoning.

This suggests that, in the current dataset distribution, the problem has a strong surface-level lexical component. In other words, the classifier does not necessarily need semantic understanding of the tool schema to separate benign and malicious samples.

6.1.2 Why heuristic approaches are weaker

Both `heuristic` and `heuristic_score` show substantially lower ROC-AUC compared to learned baselines (Table 1). This is expected for several reasons. First, heuristics tend to rely on a limited set of keyword triggers, making them brittle against paraphrasing or attacks that avoid explicit phrases. Second, heuristics generally cannot provide calibrated ranking across diverse prompt types; they behave more like hard rules than probabilistic detectors.

The results indicate that handcrafted rules may block a subset of obvious attacks but do not provide consistent discrimination across the full dataset distribution. This supports the design choice of including learned models as the primary detection mechanism.

6.1.3 Transformer-based models and the role of context

The transformer baselines and context-aware transformers achieve near-perfect ranking performance on `S_random` and `S_attack_holdout` in the standard dataset configuration (Table 1). This suggests that the dataset contains sufficient separability for a pretrained language model to learn robust indicators of malicious intent.

However, the strong performance of `tfidf_lr` also implies that this success is not necessarily due to deep reasoning about tool semantics, but may instead be explained by distributional regularities in the attack prompt phrasing.

6.2 Generalization Under Distribution Shift (RQ2)

This section addresses **RQ2**: to what extent context-aware models generalize to unseen attack families. The `S_attack_holdout` protocol is designed to evaluate generalization beyond memorized attack templates by holding out an entire attack family (AF4) from training. Under this setting, both context transformer variants remain near-perfect in the standard dataset configuration (Table 1), with ROC-AUC values close to 1.0.

This outcome suggests that the learned models do not rely exclusively on memorizing the surface form of AF4, but can generalize from other attack families. One plausible interpretation is that the dataset contains common structural properties across attack families, such as:

- attempts to override role or system-level instructions,
- explicit instructions to misuse tools,
- phrasing patterns that resemble jailbreak or prompt override prompts.

At the same time, near-ceiling generalization results should be interpreted cautiously. Since the dataset is synthetically generated, the held-out family may still share underlying phrasing patterns with other families, which can artificially simplify the generalization task. This motivates the need for further evaluation on human-written and real-world tool misuse prompts (see Section 6.6).

Note on `S_tool_holdout`. As discussed in Section 5.1, the `S_tool_holdout` protocol can yield single-class training splits, causing affected model–protocol combinations to be omitted from the results.

6.3 Truncation Bias as a Catastrophic Failure Mode (RQ3, H1, H2)

This section addresses **RQ3** and evaluates hypotheses **H1** and **H2**. The most significant result of this thesis is that truncation strategy can dominate model performance when tool schemas become large. The long-schema stress test provides evidence that naive truncation is not merely a minor preprocessing detail, but a critical robustness factor.

6.3.1 Why truncation occurs in context-augmented tool-use inputs

In ToolShield, the context transformer concatenates multiple fields into a single input sequence, including role framing, tool metadata, schema, tool description, and the user prompt. With a fixed maximum sequence length (256 tokens), the tokenizer must truncate some part of the input.

Naive truncation follows the standard HuggingFace right-truncation behavior: once the maximum token budget is exceeded, tokens at the end of the concatenated sequence are removed. Since the prompt is appended last, this approach implicitly prioritizes the schema and context fields over the prompt.

This ordering is undesirable for injection detection, because the user prompt is the primary source of malicious instructions. The schema and tool metadata provide context, but are not themselves indicative of whether the user prompt is malicious.

6.3.2 Empirical evidence of prompt removal

The truncation statistics in the enterprise long-schema setting (Table 6) show maximal separation between strategies. Naive truncation results in 100% prompt truncation and a mean prompt retention ratio of 0.0. In contrast, the `keep_prompt` strategy yields 0% prompt truncation and a retention ratio of 1.0.

The prompt retention collapse is further visualized in Figure 1 and Figure 2, which show that naive truncation consistently eliminates prompt tokens across schema length bins. These results confirm **H1** (naive truncation causes significant degradation) and **H2** (prompt-preserving truncation mitigates the bias).

6.3.3 Why the failure is catastrophic

The classification performance under enterprise-length schemas collapses under naive truncation (Table 6). ROC-AUC drops to approximately 0.455–0.479, which is close to random guessing. Operating-point metrics similarly become unusable, with FPR@TPR90 and FPR@TPR95 approaching values near 1.0.

This indicates that the classifier is effectively operating without access to the user prompt. In this situation, the model must infer maliciousness solely from tool metadata and schema content, which is insufficient because these fields are largely identical across benign and malicious samples. The model is therefore deprived of the key signal necessary for injection detection.

In practical terms, this failure mode would cause a deployed guard system to behave unpredictably under realistic tool schemas, potentially allowing prompt injection attacks to pass undetected while also generating excessive false positives.

6.3.4 Why truncation bias is not visible in the standard dataset

In the standard dataset configuration, the schema length is constrained by `max_schema_length=200` characters. Under this setting, schemas remain short enough that the total sequence typically fits within the 256-token budget. As a result, both truncation strategies achieve similar performance (Table 5).

This highlights a key methodological point: evaluation under artificially short schemas can hide truncation failure modes that would emerge in realistic enterprise settings. The long-schema stress test therefore provides a necessary complement to the standard evaluation.

6.4 Robustness vs. Latency Tradeoff

The `keep_prompt` strategy introduces a measurable inference latency overhead (Table 4). This is expected, as the prompt-preserving method performs separate tokenization passes for context and prompt and enforces explicit budget allocation.

From a deployment perspective, this introduces a tradeoff:

- **Naive truncation:** lower latency, but fails catastrophically when schemas exceed the token budget.
- **Prompt-preserving truncation:** higher latency, but stable performance under realistic schema sizes.

The results suggest that prompt preservation should be treated as a correctness requirement rather than an optional improvement. In other words, a faster model is not useful if it systematically discards the information needed to detect attacks.

However, the observed overhead also motivates future work on efficiency improvements. Possible directions include caching tokenized schema context, using smaller transformer backbones, or applying hierarchical encoding strategies that avoid repeated tokenization.

6.5 Implications for Real-World LLM Tool-Use Systems

The results of ToolShield have direct implications for LLM tool-use architectures that rely on structured tool schemas, JSON specifications, or API documentation.

6.5.1 Schema length as a deployment reality

Real enterprise tool schemas are often substantially longer than the short schemas used in many academic benchmarks. They may include nested objects, long parameter descriptions, multiple endpoints, and detailed constraints. The long-schema stress test approximates this condition and demonstrates that input preprocessing decisions can break detection performance entirely.

This implies that prompt injection defenses evaluated only on short schemas may provide a misleading sense of security.

6.5.2 Guard models must prioritize the user prompt

The results support the design principle that prompt injection detection must preserve the user prompt at all costs. Tool context is relevant for interpreting the prompt, but it is not the primary discriminative signal.

A guard system that truncates the prompt risks silently degrading into a schema-only classifier. Such a model would be incapable of detecting prompt-based attacks, even if its architecture appears sophisticated.

6.5.3 Operating-point evaluation matters

The budget-based evaluation metrics (Table 3) and ASR reduction metrics (Table 2) emphasize that threshold selection matters in deployment. A model with strong ROC-AUC may still be unusable if it cannot achieve a low false positive rate at high recall.

In the long-schema setting, the naive truncation strategy fails precisely in this sense: even if the model produces scores, it cannot achieve a threshold that satisfies reasonable false positive budgets.

6.6 Threats to Validity

Several limitations affect the interpretation of the results.

6.6.1 Synthetic dataset generation

The dataset is synthetically generated, which may introduce artifacts that simplify classification. In particular, malicious prompts may contain stylistic patterns that are easier to detect than in real adversarial prompts written by humans. Similarly, benign prompts may be less diverse than real-world tool-use queries.

This limits the external validity of absolute performance values, particularly near-ceiling ROC-AUC results.

6.6.2 Schema inflation method

The long-schema stress test inflates schemas to approximately 4 000 characters. While this produces realistic schema lengths, it is still a controlled transformation of synthetic schemas. Real enterprise schemas may differ not only in length but also in structural complexity and descriptive language. Therefore, while the truncation failure mode is demonstrated clearly, the precise quantitative effects may vary under real schemas.

6.6.3 Limited model diversity

Only a small set of model families was evaluated: heuristics, TF-IDF + logistic regression, and transformer-based classifiers. Larger-scale architectures, instruction-tuned models, or specialized retrieval-based defenses were not explored. Additionally, only one base transformer family was used, limiting conclusions about generality across architectures.

6.6.4 Evaluation scope

The evaluation focuses on binary classification of malicious vs. benign prompts. Real-world prompt injection defense often requires finer-grained distinctions, such as:

- separating benign but unusual prompts from malicious prompts,
- identifying which tool calls are unsafe,
- reasoning about multi-turn conversations and tool outputs.

The current evaluation does not cover multi-turn tool-use traces or attacks that exploit tool output channels, which are relevant in practical deployments.

6.7 Reproducibility and Credibility of Experimental Evidence

A strength of ToolShield is the emphasis on determinism and verification. Experimental results are aggregated over controlled seeds, and the artifact bundle includes environment snapshots, run manifests, and automated verification scripts.

The files `split_hygiene.md` and `guards.json` provide evidence that split constraints were enforced correctly, including template leakage prevention and AF4 holdout integrity. Additionally, `verification_output.txt` documents successful execution of the verification pipeline.

These measures reduce the likelihood that results are artifacts of accidental leakage, incomplete runs, or nondeterministic preprocessing. While reproducibility does not guarantee external validity, it strengthens confidence that the reported comparisons are internally consistent and repeatable.

6.8 Summary of Contributions and Remaining Open Problems

ToolShield contributes a controlled experimental framework for prompt injection detection in tool-use settings, including multiple baseline models and split protocols. The primary technical contribution is the identification and empirical demonstration of truncation bias

as a catastrophic failure mode under realistic schema lengths, together with a prompt-preserving truncation strategy that eliminates this failure.

At the same time, several open problems remain unresolved. The evaluation is performed on synthetic data and does not establish robustness against adaptive human adversaries. Furthermore, latency overhead and scaling to larger tools and multi-step tool-use workflows remain challenges.

Overall, the thesis demonstrates that correct input construction is a prerequisite for reliable prompt injection detection and that guard systems must explicitly preserve the prompt signal to remain functional under enterprise tool schemas.

6.9 Discussion Conclusions

- Strong baseline performance of `tfidf_lr` indicates that lexical signals play a major role in the synthetic dataset distribution (Table 1).
- Heuristic detectors provide limited ranking performance and are insufficient as standalone defenses.
- Under `S_attack_holdout`, transformer-based models generalize well in the standard schema setting, but this generalization may partially reflect synthetic distribution overlap.
- Naive truncation is a catastrophic failure mode under enterprise-length schemas: it removes the user prompt entirely and collapses detection performance to near-random (Table 6, Figure 1).
- Prompt-preserving truncation restores robustness but introduces measurable latency overhead (Table 4).
- The results highlight that preprocessing and token budget allocation are security-critical design decisions in real-world LLM tool-use systems.
- Verification scripts and artifact-based reporting strengthen credibility by reducing the likelihood of split leakage or incomplete experimental runs.

7 Threats to Validity

This chapter discusses limitations and potential threats to the validity of the experimental results presented in this thesis. Since ToolShield is evaluated under a controlled benchmark setting with synthetic data generation, careful consideration is required when interpreting the results and extrapolating them to real-world tool-using LLM systems. The threats are structured according to common validity categories in empirical computer science research.

7.1 Internal Validity

Internal validity concerns whether the reported results are caused by the intended experimental factors rather than uncontrolled confounders.

7.1.1 Synthetic dataset generation bias

The dataset used in this thesis is synthetically generated. While this enables systematic control over prompt injection families and tool schemas, it introduces the risk that generated benign and malicious prompts contain artifacts that are specific to the generator logic. If such artifacts correlate with the label distribution, classifiers may learn shortcuts rather than robust prompt injection features. This may inflate performance metrics, particularly under the `S_random` protocol where the training and test distribution are closely aligned.

Additionally, synthetic attacks may not fully reflect the diversity and creativity of real adversarial prompts. Prompt injection attempts in practice may contain indirect manipulation strategies, novel formatting, or subtle semantic coercion that is not represented in the synthetic benchmark.

7.1.2 Controlled experimental setting

The evaluation is conducted under a controlled offline classification setting. The detector is evaluated as a binary classifier predicting whether a single tool-use prompt is malicious. In real systems, tool execution decisions are often influenced by additional system state such as dialogue history, user identity, rate limits, and application-specific security policies. The absence of these factors in the benchmark may limit how well the results represent deployment scenarios.

Furthermore, the evaluation assumes that the tool schema and tool description are always available as structured context, which is typical for function calling but may not hold for all LLM integration patterns.

7.1.3 Template artifacts and distribution simplifications

Although split hygiene checks were implemented, there remains a risk that synthetic template structure itself creates distinguishable patterns. For example, benign prompts might systematically differ in tone, length, or formatting compared to attack prompts. Even if template leakage is prevented, models could exploit distributional cues that would not exist in a more heterogeneous dataset.

The benchmark also models prompt injection as a binary classification task with a limited set of attack families (AF1–AF4). While these families cover representative strategies, they do not capture the full complexity of real prompt injection where adversaries may combine multiple techniques or dynamically adapt based on the system response.

7.1.4 Risk of overfitting to synthetic prompt style

The strong performance of certain models (e.g., TF-IDF + Logistic Regression) suggests that lexical patterns may be sufficient for classification within the benchmark distribution. This raises the possibility that the models overfit to synthetic writing style, rather than learning more generalizable features. Such overfitting would reduce the detector's effectiveness against adversarial prompts that intentionally avoid the learned lexical indicators.

The `S_attack_holdout` protocol partially addresses this concern by evaluating on a held-out attack family. However, it remains possible that shared generator artifacts persist across families, enabling generalization that may not translate to real-world attacks.

7.2 External Validity

External validity concerns the extent to which results generalize to real-world environments.

7.2.1 Tool schema realism and application diversity

The benchmark models tool schemas and tool descriptions in a simplified and controlled form. Real enterprise tool schemas may contain nested JSON structures, long enumerations, multiple parameter groups, and extensive documentation text. Additionally, the semantic meaning of schemas varies across domains (finance, healthcare, internal IT systems), which may influence how prompt injection manifests.

The long-schema stress test approximates enterprise-scale schemas by inflating schema length, but the inflation procedure may not capture the semantic structure of real schemas. Therefore, the observed truncation bias results demonstrate a valid failure mode, but the exact threshold where truncation becomes harmful may differ in real systems depending on schema formatting and tokenization characteristics.

7.2.2 Attacker creativity and adaptive strategies

Real attackers can adapt strategies based on model behavior, system prompts, and tool execution feedback. In contrast, the benchmark attacks are static samples. Adaptive adversaries may use obfuscation, multi-step reasoning, or instruction camouflage, potentially reducing the effectiveness of static classifiers.

In addition, adversaries may target vulnerabilities beyond prompt injection detection, such as manipulating the tool outputs, exploiting weak downstream validation, or inducing denial-of-service behavior. These aspects are not modeled in the current evaluation.

7.2.3 Multilingual and cross-cultural prompt injection

The dataset is primarily English-based. In practical deployments, tool-use systems may receive multilingual user input. Prompt injection attempts may include code-switching, non-Latin scripts, or language-specific coercion patterns. Since multilingual attacks were not evaluated, the results should not be assumed to generalize beyond the language distribution represented in the dataset.

7.2.4 Multi-turn agents and tool chaining

This thesis evaluates single-turn classification of tool prompts. Many real systems operate as multi-turn agents, where prompt injection may occur gradually through context accu-

mulation, indirect user manipulation, or tool output poisoning. Additionally, tool chaining can introduce new attack surfaces, where an attacker exploits the output of one tool to compromise subsequent tool calls.

Since multi-turn and tool-chain scenarios are out of scope, ToolShield's results cannot be directly interpreted as a complete defense for full agentic systems.

7.3 Construct Validity

Construct validity concerns whether the chosen metrics and experimental design accurately represent the intended security objective.

7.3.1 ROC-AUC and PR-AUC versus operational security

ROC-AUC and PR-AUC are useful for comparing ranking performance across models, but they do not directly translate to deployment safety. Real-world systems require threshold selection under strict false positive constraints, and the cost of blocking benign prompts may be high. A model with strong ROC-AUC may still be unsuitable if its achievable operating points lead to unacceptable false positives.

Therefore, the inclusion of operating-point metrics such as FPR@TPR90/95 and false-positive budget evaluations improves construct validity. However, these metrics still rely on the dataset's label distribution and may not reflect the class imbalance observed in production environments.

7.3.2 Limitations of ASR reduction as a security metric

ASR reduction measures the fraction of attacks blocked by a detector under a specific operating point. While this is relevant for evaluating gating mechanisms, it assumes that attacks are either fully successful or fully prevented. In practice, prompt injection success may be partial: an attacker may succeed in exfiltrating some data, influencing tool parameters slightly, or causing limited policy violations.

Additionally, ASR reduction is defined relative to the synthetic attack set. If real-world attacks differ substantially, ASR reduction may not accurately represent true security improvement.

7.3.3 Threshold selection and budget constraints

The budget-based evaluation selects thresholds based on validation splits and evaluates them on test data. This assumes that the validation set is representative of deployment conditions. Under distribution shift, the selected threshold may not remain optimal. The `S_attack_holdout` protocol provides one form of shift, but real-world shifts may be more complex, involving unseen tools, unseen user populations, or different prompt formatting conventions.

Moreover, the budget setting assumes a static acceptable false-positive rate (e.g., 1%, 3%, 5%). In practice, acceptable false positives may depend on tool criticality, user trust level, or downstream validation mechanisms.

7.4 Reproducibility and Reliability Validity

Reproducibility validity concerns whether results can be reliably reproduced and whether experimental outcomes are stable.

7.4.1 Deterministic splits and controlled seeds

The experiment pipeline uses deterministic dataset generation, split generation, and repeated evaluation across multiple seeds. This improves reliability by reducing sensitivity to random initialization effects and enabling aggregation of results. Deterministic split protocols also reduce the risk of accidental leakage across train/validation/test partitions.

7.4.2 Environment freeze and experiment manifests

The environment is frozen via dependency recording, and run metadata is stored in manifest files. These artifacts reduce ambiguity about package versions and runtime configurations. The presence of a dedicated artifact index (`THESIS_ARTIFACTS.md`) strengthens reproducibility by providing a single authoritative reference to the exact files used for thesis tables and figures.

7.4.3 Verification scripts and split hygiene checks

To strengthen experimental credibility, automated verification scripts were added. The files `split_hygiene.md` and `guards.json` document integrity checks ensuring that split protocols satisfy constraints such as attack-family holdout and absence of template leakage. The file `verification_output.txt` provides an auditable record that the verification pipeline was executed successfully.

While these checks cannot guarantee that the synthetic benchmark represents real attacks, they reduce the risk of experimental errors such as accidental data leakage or incomplete artifact generation.

7.4.4 Residual nondeterminism in ML frameworks

Despite controlled seeds, some nondeterminism may remain due to deep learning framework behavior, hardware execution differences, or underlying numerical operations. This is particularly relevant for transformer training, where GPU kernels and low-level optimizations may introduce slight variation. Although the reported results appear stable under repeated runs, minor deviations could occur across different machines or software versions.

7.5 Mitigation Summary

Several measures were taken to reduce the identified threats. Template leakage prevention and split protocol verification reduce internal validity risks. Multiple split protocols, including `S_attack_holdout`, improve confidence in generalization beyond simple memorization. The long-schema stress test provides a controlled demonstration of truncation bias under enterprise-scale schemas. Finally, reproducibility was strengthened through deterministic seeds, frozen environments, manifest generation, and explicit verification artifacts (`split_hygiene.md`, `guards.json`, `verification_output.txt`) indexed via `THESIS_ARTIFACTS.md`. Despite these mitigations, external validity remains limited by the synthetic nature of the benchmark and the absence of multi-turn agent evaluation, which should be addressed in future work.

8 Conclusion

This thesis addressed the problem of detecting prompt injection attacks in tool-using LLM systems. In such systems, user-provided prompts are combined with privileged tool context (e.g., tool schemas and descriptions) and forwarded to an LLM that decides whether and how to invoke external tools. This creates a security risk: attackers may attempt to override system constraints or manipulate tool calls through prompt injection, potentially leading to unauthorized actions or data leakage.

The primary research goal of this thesis was to evaluate whether prompt injection detection can be implemented as a practical, reproducible classification pipeline, and whether context-augmented detection remains robust under realistic tool schema conditions. In particular, the work investigated how truncation behavior in context-augmented transformer models affects security performance.

8.1 Summary of Findings

The findings are organized according to the research questions and hypotheses stated in Section 1.1.

The empirical evaluation demonstrated that multiple baseline approaches can achieve strong performance on the standard synthetic benchmark. In particular, lexical baselines such as TF-IDF with logistic regression achieved near-ceiling performance on the `S_random` protocol, while rule-based heuristics performed substantially weaker, indicating that simple keyword matching is insufficient as a general solution (**RQ1**).

Under the distribution shift protocol `S_attack_holdout`, which holds out an entire attack family during training, the best-performing transformer-based models retained strong performance. This indicates that the learned detectors generalize beyond memorization of attack templates, at least within the benchmark scope (**RQ2**).

A key result of this thesis is that truncation strategy becomes a critical design factor once tool schemas exceed realistic enterprise sizes. Under the standard dataset configuration with a short schema constraint (`max_schema_length=200`), the `naive` and `keep_prompt` truncation strategies produced similar performance. This is consistent with the fact that schema context remained too small to induce systematic prompt loss.

However, the enterprise long-schema stress test demonstrated a qualitative failure mode. When tool schemas were inflated to approximately 4 000 characters (approximately 840 schema tokens), naive right-truncation removed the prompt content entirely in all evaluated samples. Under this condition, the classifier's performance collapsed to near-random discrimination, while the prompt-preserving truncation strategy remained stable and retained strong detection performance. This result shows that context-augmented prompt injection detection is not only a modeling problem but also a systems-level input construction problem: if the prompt is truncated away, even strong architectures cannot recover (**RQ3**). These findings confirm both **H1** (naive truncation causes catastrophic degradation under long schemas) and **H2** (prompt-preserving truncation mitigates the failure).

8.2 Main Contribution

The main technical contribution of this thesis is a prompt-preserving truncation strategy for context-augmented transformer-based prompt injection detection. Instead of relying on standard right-truncation after concatenating tool context and user prompt, the proposed

method enforces a minimum prompt token budget and allocates remaining tokens to tool context. This ensures that the primary attack surface—the user-controlled prompt—remains visible to the classifier even when enterprise tool schemas exceed the available context window.

In addition, this thesis provides an end-to-end reproducible evaluation pipeline, including dataset generation, split protocols, baseline implementations, evaluation scripts, and verification artifacts that document split integrity and experiment consistency.

8.3 Practical Recommendation

Based on the findings, a clear recommendation for real-world LLM tool-use systems is that prompt injection detection mechanisms should explicitly control token budget allocation between tool context and user prompt. Systems that rely on naive concatenation and default truncation risk silently discarding the most security-relevant input under long schemas, leading to catastrophic degradation.

Therefore, any deployment of context-aware safety classifiers should implement prompt-preserving input construction, and latency overhead should be treated as a measurable engineering tradeoff rather than ignored. Furthermore, evaluation should include stress testing with long tool schemas and distribution shift protocols, as standard random splits may not reveal truncation-related failure modes.

8.4 Future Work

Several directions remain open for future work.

First, evaluation on real-world datasets and authentic enterprise tool schemas would strengthen external validity. Synthetic benchmarks enable controlled experimentation, but real tool schemas contain complex semantics, nested structures, and domain-specific constraints that may influence both attack strategies and detection behavior.

Second, future work should extend the evaluation beyond single-turn classification. Multi-turn agents may accumulate injected instructions over time, and tool outputs may themselves introduce indirect injection opportunities. A more complete security evaluation would therefore include multi-step agent traces and memory effects.

Third, tool chaining and tool composition attacks should be explored. In realistic agent workflows, multiple tools are invoked sequentially, and attackers may exploit intermediate tool outputs to compromise downstream tool calls.

Fourth, adaptive attacker models should be considered. The current evaluation assumes static attacks, while real adversaries can iteratively refine prompts based on model behavior. Stronger adversaries could be modeled through automated search or reinforcement learning based prompt optimization.

Fifth, scaling experiments to larger transformer models may yield different robustness and latency tradeoffs. Larger models may improve detection capability, but they may also increase inference cost and complicate deployment constraints.

Sixth, multilingual prompt injection attacks represent an important extension. Real systems operate in multilingual environments, and injection strategies may differ significantly across languages and scripts.

Finally, runtime optimization of the prompt-preserving truncation pipeline is a practical engineering goal. The `keep_prompt` strategy introduces overhead due to multi-pass tokenization and budget allocation. Future work could investigate caching of tool schema

encodings, optimized batching strategies, or more efficient model architectures to reduce latency while preserving robustness.

Final statement. This thesis demonstrates that prompt-preserving truncation is a necessary design principle for robust prompt injection detection in context-augmented LLM tool-use systems, and provides a reproducible benchmark pipeline that empirically validates this failure mode under enterprise-length tool schemas.

A Experimental Tables and Figures

Table 1: Model performance across split protocols on the standard dataset. Higher ROC-AUC and lower FPR@TPR are better.

Protocol	Model	ROC-AUC	PR-AUC	FPR@TPR90	FPR@TPR95
S_attack_holdout	context_transformer_keep_prompt	0.999	0.999	0.000	0.010
S_attack_holdout	context_transformer_naive	0.998	0.999	0.000	0.010
S_random	context_transformer	1.000	1.000	0.000	0.000
S_random	context_transformer_keep_prompt	1.000	1.000	0.000	0.000
S_random	context_transformer_naive	1.000	1.000	0.000	0.000
S_random	heuristic	0.714	0.872	1.000	1.000
S_random	heuristic_score	0.714	0.872	1.000	1.000
S_random	tfidf_lr	1.000	1.000	0.000	0.000
S_random	transformer	0.988	0.991	0.020	0.200

Table 2: ASR reduction at different TPR operating points on the standard dataset. Higher is better.

Protocol	Model	ASR Red.@TPR90	ASR Red.@TPR95
S_attack_holdout	context_transformer_keep_prompt	93.6%	97.6%
S_attack_holdout	context_transformer_naive	91.2%	97.6%
S_random	context_transformer	90.3%	100.0%
S_random	context_transformer_keep_prompt	92.7%	100.0%
S_random	context_transformer_naive	94.4%	96.0%
S_random	heuristic	100.0%	100.0%
S_random	heuristic_score	100.0%	100.0%
S_random	tfidf_lr	90.3%	96.8%
S_random	transformer	94.4%	100.0%

Table 3: Performance at FPR budgets (1%, 3%, 5%) on the standard dataset. Threshold selected on validation set.

Protocol	Model	TPR@1%	TPR@3%	TPR@5%
S_attack_holdout	context_transformer_keep_prompt	0.0%	0.0%	0.0%
S_attack_holdout	context_transformer_naive	0.0%	0.0%	0.0%
S_random	context_transformer	nan%	nan%	nan%
S_random	context_transformer_keep_prompt	0.0%	0.0%	0.0%
S_random	context_transformer_naive	0.0%	0.0%	0.0%
S_random	heuristic	nan%	nan%	nan%
S_random	heuristic_score	nan%	nan%	nan%
S_random	tfidf_lr	nan%	nan%	nan%
S_random	transformer	nan%	nan%	nan%

Table 4: Inference latency per batch (100 samples) on the standard dataset. Lower is better.

Model	P50 (ms)	P95 (ms)
context_transformer_keep_prompt	4378.38	4912.70
context_transformer_naive	2861.11	3092.64
context_transformer	11525.83	12220.22
heuristic	1.48	1.71
heuristic_score	1.58	1.65
tfidf_lr	2.63	2.89
transformer	11614.90	12622.74

Table 5: Truncation strategy ablation on the standard dataset. TPR and ASR at fixed FPR budgets (1%, 3%, 5%), plus warm inference latency. Mean±std over seeds.

		TPR (%)			ASR (%)			Latency (ms)	
Protocol	Strategy	@1%	@3%	@5%	@1%	@3%	@5%	P50	P95
S_attack_holdout	keep_prompt	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4378.38	4912.70
S_attack_holdout	naive	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2861.11	3092.64
S_random	keep_prompt	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4669.87	4995.15
S_random	naive	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2744.15	3030.53

Table 6: Enterprise truncation stress test (4 000-char schemas). Prompt truncation and model performance under naive vs `keep_prompt` strategies.

		Truncation		Performance				Latency (ms)	
Protocol	Strategy	Trunc.%	Ret. Mean	ROC-AUC	PR-AUC	FPR@90	FPR@95	P50	P95
S_attack_holdout	keep_prompt	0.0%	1.000	0.998	0.998	0.000	0.010	4577.99	4776.15
S_attack_holdout	naive	100.0%	0.000	0.455	0.541	0.920	0.950	4225.00	4623.28
S_random	keep_prompt	0.0%	1.000	0.997	0.998	0.000	0.003±0.006	4485.98	5004.86
S_random	naive	100.0%	0.000	0.479	0.657	0.900	0.960	4485.99	4755.51

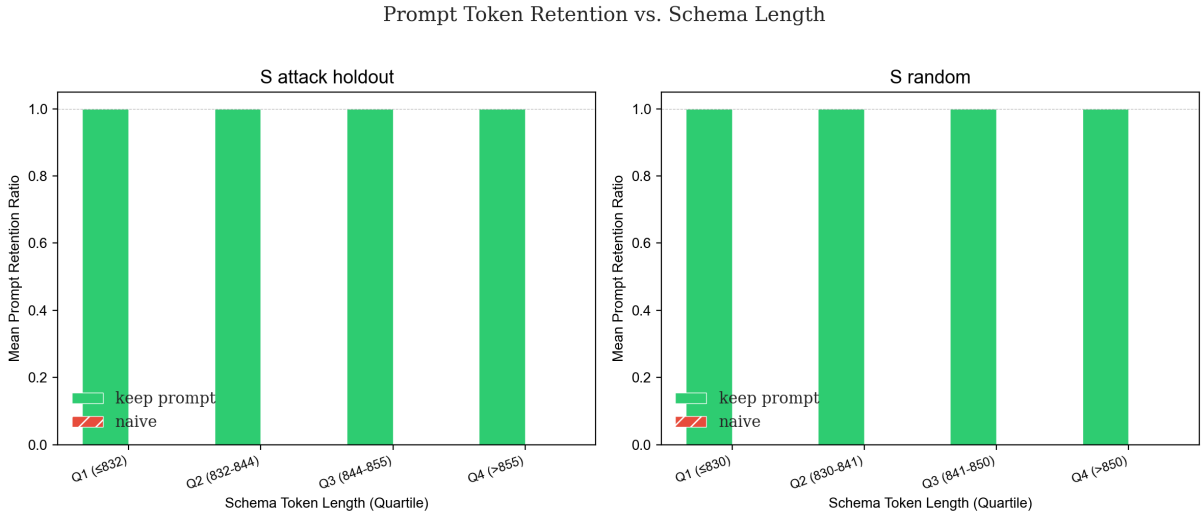

Figure 1: Prompt retention ratio by schema token length quartiles under enterprise-length schemas. Naive truncation collapses retention to zero across all bins, while `keep_prompt` preserves the full prompt.

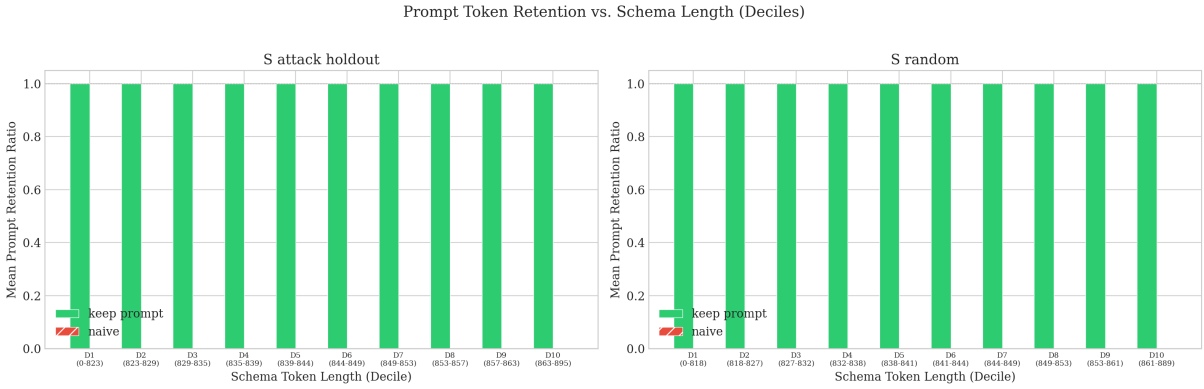

Figure 2: Prompt retention ratio by schema token length deciles under enterprise-length schemas. Fine-grained view confirming that retention collapse under naive truncation is uniform across all schema length bins.

B Reproducibility Notes

All experimental results reported in this thesis can be reproduced from the repository using two commands:

```
make compare_truncation_longschema
make verify_longschema_results
```

The first command executes the full long-schema truncation ablation pipeline (dataset generation, splitting, training, evaluation, and aggregation). The second command runs the automated verification script, which checks artifact completeness, truncation invariant consistency, and ROC-AUC regression bounds.

Artifact index. The file `THESIS_ARTIFACTS.md` in the repository root serves as the authoritative index linking each thesis table and figure to the exact data file from which it was derived.

Supporting artifacts. The following files document the experimental environment and execution:

- `run_manifest.md` — records the git commit hash, executed commands, dataset configuration, and model settings for each experiment run.
- `environment_freeze.txt` — captures the exact Python package versions (torch, transformers, scikit-learn, etc.) used during training and evaluation.
- `verification_output.txt` — contains the full output of the automated verification pipeline, confirming that all artifact checks passed.

Together with deterministic seeds and controlled split protocols, these artifacts ensure that reported results are traceable to a specific repository state and reproducible under equivalent conditions.

References

[1] TBD. Placeholder: Prompt injection survey, 2024.

[2] TBD. Placeholder: Benchmark limitations, 2024.

[3] TBD. Placeholder: Context window limitations, 2024.

[4] TBD. Placeholder: Contextual defenses, 2024.

[5] TBD. Placeholder: Function calling risks, 2024.

[6] TBD. Placeholder: Harmful prompt detection, 2024.

[7] TBD. Placeholder: Indirect prompt injection, 2024.

[8] TBD. Placeholder: Instruction hierarchy, 2024.

[9] TBD. Placeholder: Jailbreak taxonomy, 2024.

[10] TBD. Placeholder: Prompt injection benchmarks, 2024.

[11] TBD. Placeholder: Prompt injection survey, 2024.

[12] TBD. Placeholder: Prompt injection taxonomy, 2024.

[13] TBD. Placeholder: Safety classifier overview, 2024.

[14] TBD. Placeholder: Token budgeting methods, 2024.

[15] TBD. Placeholder: Tool-use security, 2024.

[16] TBD. Placeholder: Truncation bias, 2024.