Bad Characters: Imperceptible NLP Attacks

Nicholas Boucher
University of Cambridge
Cambridge, United Kingdom
nicholas.boucher@cl.cam.ac.uk

Ilia Shumailov
University of Cambridge
Vector Institute, University of Toronto
ilia.shumailov@cl.cam.ac.uk

Ross Anderson
University of Cambridge
University of Edinburgh
ross.anderson@cl.cam.ac.uk

Nicolas Papernot
Vector Institute, University of Toronto
Toronto, Canada
nicolas.papernot@utoronto.ca

***Abstract*—Several years of research have shown that machine-learning systems are vulnerable to adversarial examples, both in theory and in practice. Until now, such attacks have primarily targeted visual models, exploiting the gap between human and machine perception. Although text-based models have also been attacked with adversarial examples, such attacks struggled to preserve semantic meaning and indistinguishability. In this paper, we explore a large class of adversarial examples that can be used to attack text-based models in a black-box setting without making any human-perceptible visual modification to inputs. We use encoding-specific perturbations that are imperceptible to the human eye to manipulate the outputs of a wide range of Natural Language Processing (NLP) systems from neural machine-translation pipelines to web search engines. We find that with a single imperceptible encoding injection – representing one invisible character, homoglyph, reordering, or deletion – an attacker can significantly reduce the performance of vulnerable models, and with three injections most models can be functionally broken. Our attacks work against currently-deployed commercial systems, including those produced by Microsoft and Google, in addition to open source models published by Facebook and IBM. This novel series of attacks presents a significant threat to many language processing systems: an attacker can affect systems in a targeted manner without any assumptions about the underlying model. We conclude that text-based NLP systems require careful input sanitization, just like conventional applications, and that given such systems are now being deployed rapidly at scale, the urgent attention of architects and operators is required.**

***Index Terms*—adversarial machine learning, NLP, text-based models, text encodings, search engines**

I. Introduction

Do x and x look the same to you? They may look identical to humans, but not to most natural-language processing systems. How many characters are in the string "123"? If you guessed 100, you're correct. The first example contains the Latin character x and the Cyrillic character h, which are typically rendered the same way. The second example contains 97 zero-width non-joiners[1] following the visible characters. Indeed, the title of this paper contains 1000 invisible characters imperceptible to human users.

Several years of research have demonstrated that machine-learning systems are vulnerable to adversarial examples, both theoretically and in practice [1]. Such attacks initially targeted visual models used in image classification [2], though there has been recent interest in natural language processing and other applications. We present a broad class of powerful adversarial-example attacks on text-based models. These attacks apply input perturbations using invisible characters, control characters and homoglyphs – distinct character encodings that share similar glyphs. These perturbations are imperceptible to human users of text-based systems, but the bytes used to encode them can change the output drastically.

We have found that machine-learning models that process user-supplied text, such as neural machine-translation systems, are particularly vulnerable to this style of attack. Consider, for example, the market-leading service Google Translate[2]. At the time of writing, entering the string "paypal" in the English to Russian model correctly outputs "PayPal", but replacing the Latin character a in the input with the Cyrillic character а incorrectly outputs "папа" ("father" in English). Model pipelines are agnostic of characters outside of their dictionary and replace them with `<unk>` tokens; the software that calls them may however propagate unknown words from input to output. While that may help with general understanding of text, it opens a surprisingly large attack surface.

Simple text-encoding attacks have been used occasionally in the past to get messages through spam filters. For example, there was a brief discussion in the SpamAssassin project in 2018 about how to deal with zero-width characters, which had been found in some sextortion scams [3]. Although such tricks were known to engineers designing spam filters, they were not a primary concern. However, the rapid deployment

[1]Unicode character U+200C

[2]translate.google.com

TABLE I
IMPERCEPTIBLE PERTURBATIONS IN VARIOUS NLP TASKS

Input Rendering	Input Encoding	Task	Output
Send money to account 1234	Send money to account U+202E4321	Translation (EN→FR)	Envoyer de l'argent au compte 4321 *(Send money to account 4321)*
You are a coward and a fool.	You akU+8re aqU+8 AU+8coward and a fovU+8JU+8ol.	Toxic Content Detection	8.2% toxic *(96.8% toxic unperturbed)*
Oh, what a fool I feel! / I am beyond proud.	Oh, what a U+200BfoU+200Bol IU+200B U+200BU+200Bfeel! / I am beyond proud.	Natural Language Inference	0.3% contradiction *(99.8% contradiction unperturbed)*

of NLP systems in a large range of applications, from machine translation [4] to copyright enforcement [5] to hate speech filtering [6], is suddenly creating a host of high-value targets that have capable motivated opponents.

The main contribution of this work is to explore and develop a class of imperceptible encoding-based attacks and to study their effect on the NLP systems that are now being deployed everywhere at scale. Our experiments show that many developers of such systems have been heedless of the risks; this is surprising given the long history of attacks on many varieties of systems that have exploited unsanitized inputs. We provide a set of examples of imperceptible attacks across various NLP tasks in Table I. As we will later describe, these attacks take the form of invisible characters, homoglyphs, reorderings, and deletions injected via a genetic algorithm that maximizes a loss function defined for each NLP task.

Our findings present an attack vector that must be considered when designing any system processing natural language that may ingest text-based inputs with modern encodings, whether directly from an API or via document parsing. We then explore a series of defences that can give some protection against this powerful set of attacks, such as discarding certain characters prior to tokenization, limiting intraword character set mixing, and leveraging rendering and OCR for preprocessing. Defence is not entirely straightforward, though, as application requirements and resource constraints may prevent the use of specific defences in certain circumstances.

This paper makes the following contributions:

- We present a novel class of imperceptible perturbations for NLP models;
- We present four black-box variants of imperceptible attacks against both the integrity and availability of NLP models;
- We show that our imperceptible attacks degrade performance against task-appropriate benchmarks for six models implementing machine translation, toxic content detection, and textual entailment classification to near zero and slow inference down by at least a factor of two with just a handful of character substitutions;
- We evaluate our attacks extensively against both open source models and Machine Learning as a Service (MLaaS) offerings provided by Facebook, IBM, Microsoft, and Google, finding that all tested systems were vulnerable;
- We present defences against these attacks, and discuss why defence can be complex.

II. MOTIVATION

Researchers have already experimented with adversarial attacks on NLP models [7]–[18]. However, up until now, such attacks were noticeable to human inspection and could be identified with relative ease. If the attacker inserts single-character spelling mistakes [8]–[10], [14], they look out of place, while paraphrasing [11] often changes the meaning of a text enough to be noticeable. The attacks we discuss in this paper are the first class of attacks against modern NLP models that are imperceptible and do not distort semantic meaning.

Our attacks can cause significant harm in practice. Consider three examples. First, suppose that Alice hacks Bob's Office365 account and changes his invoice template so that it still appears to say 'Pay account no. 123', but imperceptibly perturbed so that Google Translate will render it as a different account number. Bob then sends these booby-trapped invoices to his customers, and when Carlos reads one in Spanish, he sends the money to Alice instead. Second, consider a nation-state whose primary language is not spoken by the staff at a large social media company performing content moderation – already a well-documented challenge [19]. If the government of this state wanted to make it difficult for moderators to block a campaign to incite violence against minorities, it could use imperceptible perturbations to stifle the efficacy of both machine-translation and toxic-content detection of inflammatory sentences.

Third, the ability to hide text in plain sight, by making it easy for humans to read but hard for machines to process, could be used by many bad actors to evade platform content filtering mechanisms and even impede law-enforcement and intelligence agencies. The same perturbations even prevent proper search engine indexing, making malicious content hard to locate in the first place. We found that production search engines do not parse our invisible characters and can be maliciously targeted with well-crafted queries. At the time of initial writing, Googling "The meaning of life" returned approximately 990 million results. Prior to responsible disclosure, searching for the visually identical string containing 250 invisible "zero width joiner" characters[3] returned exactly none.

[3]Unicode character U+200D

TABLE II
TAXONOMY OF ADVERSARIAL NLP ATTACKS IN ACADEMIC LITERATURE.

Attack	Features			Integrity		Availability
	Imperceptible	Semantic Similarity	Blackbox	Classification	Translation	DoS
RNN Adversarial Sequences [7]				✓		
Synthetic and Natural Noise [8]			✓		✓	
DeepWordBug [9]			✓	✓		
HotFlip [10]				✓		
Syntactically Controlled Paraphrase [11]		✓	✓	✓		
Natural Adversarial Examples [12]			✓	✓	✓	
Natural Language Adversarial Examples [13]		✓	✓	✓		
TextBugger [14]			✓	✓		
seq2seq Adversarial Perturbations [15]		✓			✓	
Probability Weighted Word Saliency [16]		✓		✓		
Sponge Examples [17]			✓	✓	✓	✓
Reinforced Generation [18]		✓	✓		✓	
Imperceptible Perturbations	✓	✓	✓	✓	✓	✓

III. RELATED WORK

A. Adversarial Examples

Machine-learning techniques are vulnerable to many large classes of attack [20], with one major class being adversarial examples. These are inputs to models which, during inference, cause the model to output an incorrect result [1]. In a white-box environment – where the adversary knows the model – such examples can be found using a number of gradient-based methods which typically aim to maximize the loss function under a series of constraints [1], [2], [21]. In the black-box setting, where the model is unknown, the adversary can transfer adversarial examples from another model [22], or approximate gradients by observing output labels and, in some settings, confidence [23].

Training data can also be poisoned to manipulate the accuracy of the model for specific inputs [24], [25]. Bitwise errors can be introduced during inference to reduce the model's performance [26]. Inputs can also be chosen to maximize the time or energy a model takes during inference [17], or to expose confidential training data via inference techniques [27]. In other words, adversarial algorithms can affect the *integrity*, *availability* and *confidentiality* of machine-learning systems [17], [28], [29].

B. NLP Models

Natural language processing (NLP) systems are designed to process human language. Machine translation was proposed as early as 1949 [30] and has become a key sub-field of NLP. Early approaches to machine translation tended to be rule-based, using expert knowledge from human linguists, but statistical methods became more prominent as the field matured [31], eventually yielding to neural networks [4], then recurrent neural networks (RNNs) because of their ability to reference past context [32]. The current state of the art is the Transformer model, which provides the benefits of RNNs and CNNs in a traditional network via the use of an attention mechanism [33].

Transformers are a form of encoder-decoder model [34], [35] that map sequences to sequences. Each source language has an encoder that converts the input into a learned interlingua, an intermediate representation which is then decoded into the target language using a model associated with that language.

Regardless of the details of the model used for translation, natural language must be encoded in a manner that can be used as its input. The simplest encoding is a dictionary that maps words to numerical representations, but this fails to encode previously unseen words and thus suffers from limited vocabulary. N-gram encodings can increase performance, but increase the dictionary size exponentially while failing to solve the unseen-word problem. A common strategy is to decompose words into sub-word segments prior to encoding, as this enables the encoding and translation of previously unseen words in many circumstances [36].

C. Adversarial NLP

Early adversarial ML research focused on image classification [2], [37], and the search for adversarial examples in NLP systems began later, targeting sequence models [7]. Adversarial examples are inherently harder to craft due to the discrete nature of natural language. Unlike images in which pixel values can be adjusted in a near-continuous and virtually imperceptible fashion to maximize loss functions, perturbations to natural language are more visible and involve the manipulation of more discrete tokens.

More generally, source language perturbations that will provide effective adversarial samples against human users need to account for semantic similarity [15]. Researchers have proposed using word-based input swaps with synonyms [16] or character-based swaps with semantic constraints [10]. These methods aim to constrain the perturbations to a set of transformations that a human is less likely to notice. Both neural machine-translation [8] and text classification [9], [14] models generally perform poorly on noisy inputs such as misspellings, but such perturbations create clear visual artifacts that are easier for humans to notice.

Using different paraphrases of the same meaning, rather than one-to-one synonyms, may give more leeway. Paraphrase sets can be generated by comparing machine back-translations

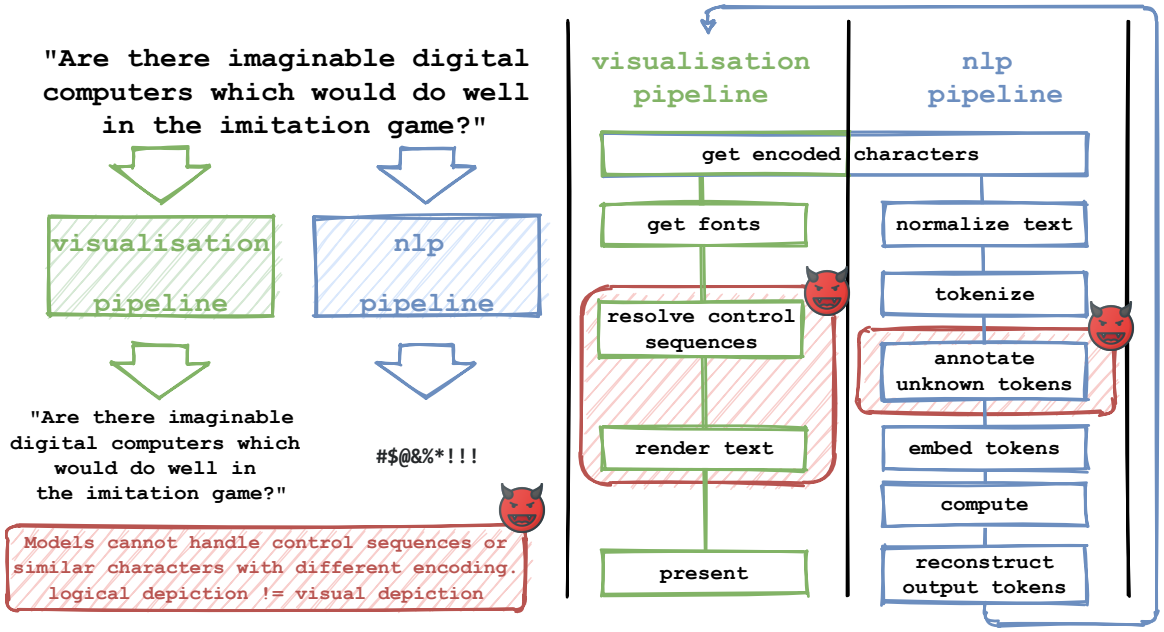

Fig. 1. Typical text visualization and NLP processing pipelines, the gap between which is the basis for imperceptible perturbation adversarial examples.

of large corpora of text [38], and used to systematically generate adversarial examples for machine-translation systems [11]. One can also search for neighbors of the input sentence in an embedded space [12]; these examples often result in low-performance translations, making them candidates for adversarial examples. Although paraphrasing can indeed help preserve semantics, humans often notice that the results look odd. Our attacks on the other hand do not introduce any visible perturbations, use fewer substitutions and preserve semantic meaning perfectly.

Genetic algorithms have been used to find adversarial perturbations against inputs to sentiment analysis systems, presenting an attack viable in the black-box setting without access to gradients [13]. Reinforcement learning can be used to efficiently generate adversarial examples for translation models [18]. There have even been efforts to combine academic NLP adversarial techniques into easily consumable toolkits available online [39], making these attacks relatively easy to use. Unlike the techniques described in this paper, though, all existing NLP adversarial example techniques result in human-perceptible visual artifacts within inputs.

While BLEU is often used to asses various accuracy metrics in natural language settings, less common similarity metrics such as chrF may provide stronger results for adversarial examples. Michel et al. also propose that unknown tokens `<unk>`, which are used to encode text sequences not recognized by the natural language encoder in NLP settings, can be leveraged to make compelling source language perturbations due to the flexibility of the characters which encode to `<unk>` [15]. However, all methods proposed so far for generating `<unk>` use visible characters.

We present a taxonomy of adversarial NLP attacks in Table II.

D. Unicode

Unicode is a character set designed to standardize the electronic representation of text [40]. As of the time of writing, it can represent 143,859 characters across many different languages and symbol groups. Characters as diverse as Latin letters, traditional Chinese characters, mathematical notation and emojis can all be represented in Unicode. It maps each character to a code point, or numerical representation.

These numerical code points, often denoted with the prefix `U+`, can be encoded in a variety of ways, although UTF-8 is the most common. This is a variable-length encoding scheme that represents code points as 1-4 bytes.

A font is a collection of glyphs that describe how code points should be rendered. Most computers support many different fonts. It is not required that fonts have a glyph for every code point, and code points without corresponding glyphs are typically rendered as an ‘unknown’ placeholder character.

E. Unicode Security

As it has to support a globally broad set of languages, the Unicode specification is quite complex. This complexity can lead to security issues, as detailed in the Unicode Consortium’s technical report on Unicode security considerations [41].

One primary security consideration in the Unicode specification is the multitude of ways to encode homoglyphs, which are unique characters that share the same or nearly the same glyph. This problem is not unique to Unicode; for example,

in the ASCII range, the rendering of the lowercase Latin 'l'[4] is often nearly identical to the uppercase Latin 'I'[5]. In some fonts, character sequences can act as pseudo-homoglyphs, such as the sequences 'rn' and 'm' in most sans serif fonts.

Such visual tricks provide a tool in the arsenal of cyber scammers [42]. The earliest example we found is that of *paypaI.com* (notice the last domain name character is an uppercase 'I'), which was used in July 2000 to trick users into disclosing passwords for *paypal.com*[6]. Indeed, significant attention has since been given to homoglyphs in URLs [43]–[46]. Some browsers attempt to remedy this ambiguity by rendering all URL characters in their lowercase form upon navigation, and the IETF set a standard to resolve ambiguities between non-ASCII characters that are homoglyphs with ASCII characters. This standard, called Punycode, resolves non-ASCII URLs to an encoding restricted to the ASCII range. For example, most browsers will automatically re-render the URL *pаypаl.com* (which uses the Cyrillic а[7]) to its Punycode equivalent *xn–pypl-53dc.com* to highlight a potentially dangerous ambiguity. However, Punycode can introduce new opportunities for deception. For example, the URL *xn–google.com* decodes to four semantically meaningless traditional Chinese characters. Furthermore, Punycode does not solve cross-script homoglyph encoding vulnerabilities outside of URLs. For example, homoglyphs have in the past caused security vulnerabilities in various non-URL areas such as certificate common names.

Unicode attacks can also exploit character ordering. Some character sets (such as Hebrew and Arabic) naturally display in right-to-left order. The possibility of intermixing left-to-right and right-to-left text, as when an English phrase is quoted in an Arabic newspaper, necessitates a system for managing character order with mixed character sets. For Unicode, this is the Bidirectional (Bidi) Algorithm [47]. Unicode specifies a variety of control characters that allow a document creator to fine-tune character ordering, including two bidi override characters that allow complete control over display order. The net effect is that an adversary can force characters to render in a different order than they are encoded, thus permitting the same visual rendering to be represented by a variety of different encoded sequences.

Lastly, an entire class of vulnerabilities stems from bugs in Unicode implementations. These have historically been used to generate a range of interesting exploits about which it is difficult to generalize. While the Unicode Consortium does publish a set of software components for Unicode support[8], many operating systems, platforms, and other software ecosystems have different implementations. For example, GNOME produces Pango[9], Apple produces Core Text[10], while Microsoft produces a Unicode implementation for Windows[11].

In what follows, we will mostly disregard bugs and focus on attacks that exploit correct implementations of the Unicode standard. We instead exploit the gap between visualisation and NLP pipelines, as illustrated in Figure 1.

IV. Background

A. Attack Taxonomy

In this paper, we explore the class of imperceptible attacks based on Unicode and other encoding conventions which are generally applicable to text-based NLP models. We see each attack as a form of adversarial example whereby imperceptible perturbations are applied to fixed inputs into existing text-based NLP models.

We define these *imperceptible perturbations* as modifications to the encoding of a string of text which result in either:

- No visual modification to the string's rendering by a standards-compliant rendering engine compared to the unperturbed input, or
- Visual modifications sufficiently subtle to go unnoticed by the average human reader using common fonts.

For the latter case, it is alternatively possible to replace human imperceptibility as indistinguishability by a computer vision model between images of the renderings of two strings, or a maximum pixel-wise difference between such rendering.

We consider four different classes of imperceptible attack against NLP models:

1) **Invisible Characters**: Valid characters which by design do not render to a visible glyph are used to perturb the input to a model.
2) **Homoglyphs**: Unique characters which render to the same or visually similar glyphs are used to perturb the input to a model.
3) **Reorderings**: Directionality control characters are used to override the default rendering order of glyphs, allowing reordering of the encoded bytes used as input to a model.
4) **Deletions**: Deletion control characters, such as the backspace, are injected into a string to remove injected characters from its visual rendering to perturb the input to a model.

These imperceptible text-based attacks on NLP models represent an abstract class of attacks independent of different text-encoding standards and implementations. For the purpose of concrete examples and experimental results, we will assume the near-ubiquitous Unicode encoding standard, but we believe our results to be generalizable to any encoding standard with a sufficiently large character and control-sequence set.

Further classes of text-based attacks exist, as detailed in Table I, but all other attack classes produce visual artefacts.

The imperceptible text-based attacks described in this paper can be used against a broad range of NLP models. As we explain later, imperceptible perturbations can manipulate

[4]ASCII value `0x6C`
[5]ASCII value `0x49`
[6]zdnet.com/article/paypal-alert-beware-the-paypai-scam-5000109103
[7]Unicode character U+0430
[8]site.icu-project.org
[9]pango.gnome.org
[10]developer.apple.com/documentation/coretext
[11]docs.microsoft.com/en-us/windows/win32/intl/unicode

machine translation, break toxic content classifiers, degrade search engine querying and indexing, and generate sponge examples [17] for denial-of-service (DoS) attacks, among other possibilities.

B. NLP Pipeline

Modern NLP pipelines have evolved through decades of research to include a large number of performance optimizations. Text-based inputs undergo a number of pre-processing steps before model inference. Typically a *tokenizer* is first applied to separate words and punctuation in a task-meaningful way, an example being the Moses tokenizer [48] used in the Fairseq models evaluated later in this paper. Tokenized words are then encoded. Early models used dictionaries to map tokens to encoded embeddings, and tokens not seen during training were replaced with a special `<unk>` embedding. Many modern models now apply Byte Pair Encoding (BPE) or the WordPiece algorithm [49] before dictionary lookups. BPE, a common data compression technique, and WordPiece both identify common subwords in tokens. This often results in increased performance, as it allows the model to capture additional knowledge about language semantics from morphemes [50]. Both of these pre-processing methodologies are commonly used in deployed NLP models, including all three open source models published by Facebook and IBM evaluated in this paper.

As we show in Figure 1, modern NLP pipelines process text in a very different manner from text-rendering systems, even when dealing with the same input. While the NLP system is dealing with the semantics of human language, the rendering engine is dealing with a large, rich set of different control characters. This structural difference between what models see and what humans see is what we exploit in our attacks.

C. Attack Methodology

We approach the generation of adversarial samples as an optimization problem. Assume an NLP function $f(\mathbf{x}) = \mathbf{y} : X \rightarrow Y$ mapping textual input $\mathbf{x}$ to $\mathbf{y}$. Depending on the task, Y is either a sequence of characters, words, or hot-encoded categories. For example, translation tasks such as WMT assume Y to be a sequence of characters, whereas categorization tasks such as MNLI assume Y to be one of three categories. Furthermore, we assume a strong black-box threat model where adversaries have access to model output but cannot observe the internals. This makes our attack realistic: we later show it can be mounted on existing commercial ML services. In this threat model, an adversary's goal is to imperceptibly manipulate f using a perturbation function p.

These manipulations fall into two categories:

- **Integrity Attack**: The adversary aims to find p such that $f(p(\mathbf{x})) \neq f(\mathbf{x})$. For a targeted attack, the adversary further constrains p such that the perturbed output matches a fixed target $\mathbf{t}$: $f(p(\mathbf{x})) = \mathbf{t}$.
- **Availability Attack**: The adversary aims to find p such that $time(f(p(\mathbf{x}))) > time(f(\mathbf{x}))$, where *time* measures the inference runtime of f.

Algorithm 1: Imperceptible perturbations adversarial example via differential evolution.

Input: text $\mathbf{x}$, attack $\mathcal{A}$ with input bounds distribution $\mathcal{B}_\mathcal{A}$, NLP task $\mathcal{T}$, target $\mathbf{y}$, perturbation budget β, population size s, evolution iterations m, differential weight $F \in [0, 2]$, crossover probability $CR \in [0, 1]$
Result: Adversarial example visually identical to $\mathbf{x}$ against task $\mathcal{T}$ using attack $\mathcal{A}$

Randomly initialize population $\mathbf{P} := \{\mathbf{p_0}, \ldots, \mathbf{p_s}\}$, where $\mathbf{p_n} \sim \mathcal{B}_\mathcal{A}(\mathbf{x})$
if availability attack **then**
  $\mathcal{F}(\cdot) = \text{execution\_time}(\mathcal{T}(\mathcal{A}(\mathbf{x}, \cdot)))$
else if integrity attack **then**
  if targeted attack **then**
    $\mathcal{F}(\cdot) = \text{levenshtein\_distance}(\mathbf{y}, \mathcal{T}(\mathcal{A}(\mathbf{x}, \cdot)))$
  else
    $\mathcal{F}(\cdot) = \text{levenshtein\_distance}(\mathcal{T}(\mathbf{x}), \mathcal{T}(\mathcal{A}(\mathbf{x}, \cdot)))$
  end if
end if
for $i := 0$ **to** m **do** ▷ $\mathcal{U}$ is uniform dist.
  for $j := 0$ **to** s **do**
    $\mathbf{p_a}, \mathbf{p_b}, \mathbf{p_c} \xleftarrow{\text{rand}} \mathbf{P}$ s.t. $j \neq a \neq b \neq c$
    $R \sim \mathcal{U}(0, |\mathbf{p_j}|)$
    $\hat{\mathbf{p}}_\mathbf{j} := \mathbf{p_j}$
    for $k := 0$ **to** $|\mathbf{p_j}|$ **do**
      $r_j \sim \mathcal{U}(0, 1)$
      if $r_j < CR$ **or** $R = k$ **then**
        $\hat{\mathbf{p}}_{\mathbf{j_k}} = \mathbf{p_{a_k}} + F \times (\mathbf{p_{b_k}} - \mathbf{p_{b_k}})$
      end if
    end for
    if $\mathcal{F}(\hat{\mathbf{p}}_\mathbf{j}) \geq \mathcal{F}(\mathbf{p_j})$ **then**
      $\mathbf{p_j} = \hat{\mathbf{p}}_\mathbf{j}$
    end if
  end for
end for
$\bar{\mathbf{f}} := \{\mathcal{F}(\mathbf{p_0}), \ldots, \mathcal{F}(\mathbf{p_s})\}$
return $\mathcal{A}(\mathbf{x}, \mathbf{p}_{\text{argmax}(\bar{\mathbf{f}})})$

We also define a constraint on the perturbation function p:

- **Budget**: A budget b such that $dist(\mathbf{x}, p(\mathbf{x})) \leq b$. The function *dist* may refer to any distance metric.

We define the attack as optimizing a set of operations over the input text, where each operation corresponds to the injection of one short sequence of Unicode characters to perform a single imperceptible perturbation of the chosen class. The length of the injected sequence is dependent upon the class chosen and attack implementation; in our evaluation we use one character injections for invisible characters and homoglyphs, two characters for deletions, and ten characters for reorderings, as later described. We select a gradient-free optimization method – differential evolution [51] – to enable this attack to work in the black-box setting without having to recover approximated gradients. This approach randomly

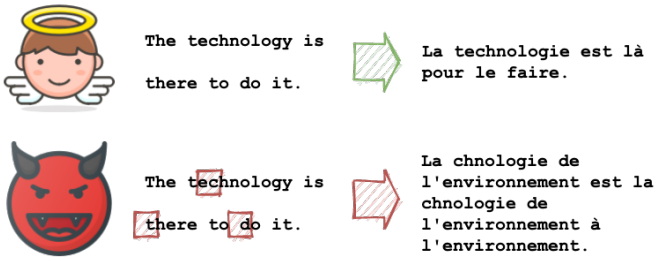

Fig. 2. Attack using invisible characters. Example machine translation input is on the left with model output on the right. Invisible characters are denoted by red boxes, such as between the 'e' and 'c'.

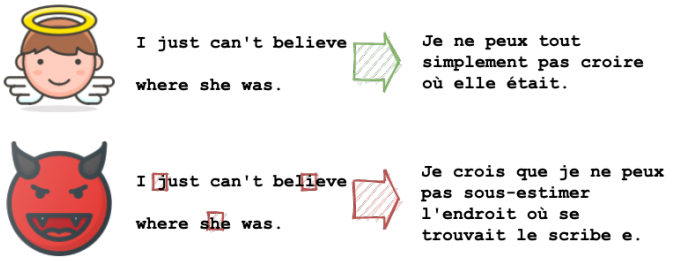

Fig. 3. Attack using homoglyphs. Example machine translation input is on the left with model output on the right. Homoglyphs are highlighted with red boxes, where *j* is replaced with U+3F3, *i* with U+456 and *h* with U+4BB.

initializes a set of candidates and evolves them over many iterations, ultimately selecting the best-performing traits.

The attack algorithm is shown in Algorithm 1. It takes as a parameter function $\mathcal{A}$ which, given an input string and perturbation encoding, returns a perturbed string, allowing the algorithm to be used for all four classes of imperceptible perturbations.

D. Invisible Characters

Invisible characters are encoded characters that render to the absence of a glyph and take up no space in the resulting rendering. Invisible characters are typically not font-specific, but follow from the specification of an encoding format. An example in Unicode is the zero-width space character[12] (ZWSP). An example of an attack using invisible characters is shown in Figure 2.

It is important to note that characters lacking a glyph definition in a specific font are not typically treated as invisible characters. Due to the number of characters in Unicode and other large specifications, fonts will often omit glyph definitions for rare characters. For example, Unicode supports characters from the ancient Mycenaean script Linear B, but these glyph definitions are unlikely to appear in fonts targeting modern languages such as English. However, most text-rendering systems reserve a special character, often □ or 🞏, for valid Unicode encodings with no corresponding glyph. These characters are therefore visible in rendered text.

In practice, though, invisible characters are font-specific. Even though some characters are designed to have a non-glyph rendering, the details are up to the font designer. They might, for example, render all traditionally invisible characters by printing the corresponding Unicode code point as a base 10 numeral. Yet a small number of fonts dominate the modern world of computing, and fonts in common use are likely to respect the spirit of the Unicode specification. For the purposes of this paper, we will determine character visibility using GNU's Unifont[13] glyphs. Unifont was chosen because of its relatively robust coverage of the current Unicode standard, its distribution with common operating systems, and its visual similarity to other common fonts.

[12]Unicode character U+200B

[13]unifoundry.com/unifont/index.html

Although invisible characters do not produce a rendered glyph, they nevertheless represent a valid encoded character. Text-based NLP models operate over encoded bytes as inputs, so these characters will be "seen" by a text-based model even if they are not rendered to anything perceptible by a human user. We found that these bytes alter model output. When injected arbitrarily into a model's input, they typically degrade the performance both in terms of accuracy and runtime. When injected in a targeted fashion, they can be used to modify the output in a desired way, and may coherently change the meaning of the output across many NLP tasks.

E. Homoglyphs

Homoglyphs are characters that render to the same glyph or to a visually similar glyph. This often occurs when portions of the same written script are used across different language families. For example, consider the Latin letter 'A' used in English. The very similar character 'А' is used in the Cyrillic alphabet. Within the Unicode specification these are distinct characters, although they are typically rendered as homoglyphs.

An example of an attack using homoglyphs is shown in Figure 3. Like invisible characters, homoglyphs are font-specific. Even if the underlying linguistic system denotes two

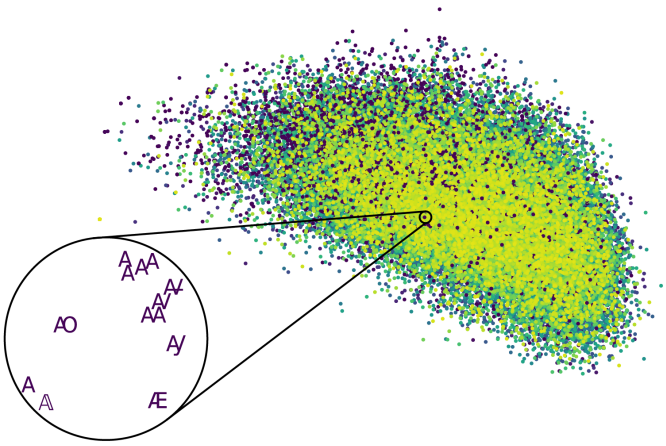

Fig. 4. Clustering of Unicode homoglyphs according to the Unicode Security Confusables document, plotted as a 2D PCA of Unifont glyph images via a VGG16 model.

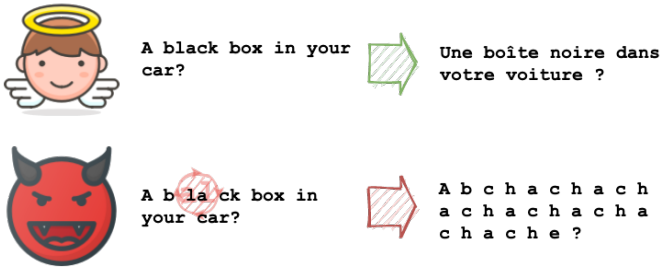

Fig. 5. Attack using reorderings. Example machine translation input is on the left with model output on the right. The red circle denotes the string is encoded in reverse order surrounded by Bidi override characters.

characters in the same way, fonts are not required to respect this. That said, there are well-known homoglyphs in the most common fonts used in everyday computing.

The Unicode Consortium publishes two supporting documents with the Unicode Security Mechanisms technical report [52] to draw attention to similarly rendered characters. The first[14] defines a mapping of characters that are intended to be homoglyphs within the Unicode specification and should therefore map to the same glyph in font implementations. The second document[15] defines a set of characters that are likely to be visually confused, even if they are not rendered with precisely the same glyph.

For the experiments in this paper, we use the Unicode technical reports to define homoglpyh mappings. We also note that homoglyphs, particularly for specific less common fonts, can be identified using an unsupervised clustering algorithm against vectors representing rendered glyphs. To illustrate this, we used a VGG16 convolution neural network [53] to transform all glyphs in the Unifont font into vectorized embeddings and performed various clustering operations. Figure 4 visualizes mappings provided by the Unicode technical reports as a dimensionality-reduced character cluster plot. We find that the results of well-tuned unsupervised clustering algorithms produce similar results, but have chosen to use the official Unicode mappings in this paper for reproducibility.

F. Reorderings

The Unicode specification supports characters from languages that read in both the left-to-right and right-to-left directions. This becomes nontrivial to manage when such scripts are mixed. The Unicode specification defines the Bidirectional (Bidi) Algorithm [47] to support standard rendering behavior for mixed-script documents. However, the specification also allows the Bidi Algorithm to be overridden using invisible direction-override control characters, which allow near-arbitrary rendering for a fixed encoded ordering.

An example of an attack using reorderings is shown in Figure 5. In an adversarial setting, Bidi override characters allow the encoded ordering of characters to be modified without affecting character rendering thus making them a form of imperceptible perturbation.

Unlike invisible character and homoglyph attacks, the class of reordering attacks is font-independent and relies only on the implementation of the Unicode Bidi Algorithm. Bidi algorithm implementations sometimes differ in how they handle specific override sequences, meaning that some attacks may be platform or application specific in practice, but most mature Unicode rendering systems behave similarly. Algorithm 2 defines an algorithm for generating 2^{n-1} unique reorderings for strings of length n using nested Bidi override characters. At the time of writing, it has been tested to work against the

Algorithm 2: Generation of 2^{n-1} visually identical strings via Unicode reorderings.

```
Input: string x of length n
Result: Set of 2^(n-1) visually identical reorderings of x

struct { string one, two; } Swap
string PDF := 0x202C, LRO := 0x202D
string RLO := 0x202E, PDI := 0x2069
string LRI := 0x2066

procedure SWAPS (body, prefix, suffix)
  Set orderings := { concatenate(prefix, body, suffix) }
  for i := 0 to length(body)-1 do
    Swap swap := { body[i+1], body[i] }
    orderings.add([prefix, body[:i],
                   swap, body[i+1:], suffix])
    orderings.union(SWAPS(suffix, [prefix, swap], null))
    orderings.union(SWAPS([prefix, swap], null, suffix))
  end for
  return orderings
end procedure

procedure ENCODE (ordering)
  string encoding := ""
  for element in ordering do
    if element is Swap
      swap = ENCODE([LRO, LRI, RLO, LRI,
                     element.one, PDI, LRI,
                     element.two, PDI, PDF,
                     PDI, PDF])
      encoding = concatenate(encoding, swap)
    else if element is string
      encoding = concatenate(encoding, element)
  end for
  return encoding
end procedure

Set orderings := { }
for ordering in SWAPS(x, null, null) do
  orderings.add(ENCODE(ordering))
end for
return orderings
```

[14] unicode.org/Public/security/latest/intentional.txt
[15] unicode.org/Public/security/latest/confusables.txt

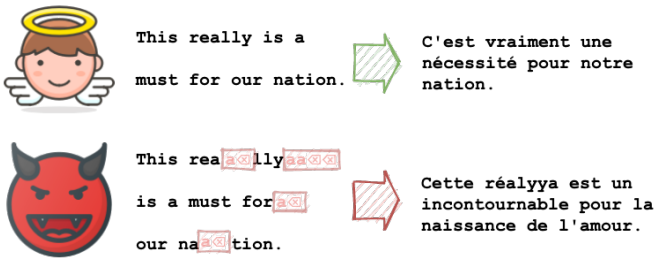

Fig. 6. Attack using deletions. Example machine translation input is on the left with model output on the right. The red boxes highlight injected characters followed by backspace characters.

Unicode implementation in Chromium[16].

Reordering attacks are particularly insidious when used with data that retains semantic validity with minor reorderings, such as standard numerals. Consider, for example, the string "Please send money to account 1234." With a single reordering, this can be rendered as "Please send money to account 2134." It is common for isolated reordering-control characters to be discarded in NLP model inference as they are often embedded as a generic <unk> token. Therefore, banking instructions passed through NLP pipelines such as machine translation before being visualized to a user can lead to malicious results.

G. Deletions

A small number of control characters in Unicode can cause neighbouring text to be removed. The simplest examples are the backspace (BS) and delete (DEL) characters. There is also the carriage return (CR) which causes the text-rendering algorithm to return to the beginning of the line and overwrite its contents. For example, encoded text which represents "Hello **CR**Goodbye World" will be rendered as "Goodbye World".

An example of an attack using deletions is shown in Figure 6. Deletion attacks are font-independent, as Unicode does not allow glyph specification for the basic control characters inherited from ASCII including BS, DEL, and CR. In general, deletion attacks are also platform independent as there is not significant variance in Unicode deletion implementations. However, these attacks can be harder to exploit in practice because most systems do not copy deleted text to the clipboard. As such, an attack using deletion perturbations generally requires an adversary to submit encoded Unicode bytes directly into a model, rather than relying on a victim's copy+paste functionality.

V. NLP Attacks

A. Integrity Attack

Regardless of the tokenizer or dictionary used in an NLP model, systems are unlikely to handle imperceptible perturbations gracefully in the absence of specific defences. Integrity attacks against NLP models exploit this fact to achieve degraded model performance in either a targeted or untargeted fashion.

[16]chromium.org

Although the response to such perturbations varies between models, the most likely pipeline is that all unfamiliar characters are embedded a special <unk> vector representing all unknown tokens. This <unk> vector, although a special case, is typically treated no differently than any other character by the model after the embedding transformation.

The specific affect on input embedding transformation depends on the class of perturbation used:

- **Invisible characters** (between words): Invisible characters are transformed into <unk> embeddings between properly-embedded adjacent words.
- **Invisible characters** (within words): In addition to being transformed into <unk> embeddings, the invisible characters may cause the word in which it is contained to be embedded as multiple shorter words, interfering with the standard processing.
- **Homoglyphs**: If the token containing the homoglyph is present in the model's dictionary, a word that contains it will be embedded with the less-common, and likely lower-performing, vector created from such data. If the homoglyph is not known, the token will be embedded as <unk>.
- **Reorderings**: In addition to the Bidi-override characters each being treated as an invisible character, the other characters input into the model will be in the underlying encoded order rather than the rendered order.
- **Deletions**: In addition to deletion-control characters each being treated as an invisible character, the deleted characters encoded into the input are still validly processed by the model.

Each of these modifications to embedded inputs degrades a model's performance. The cause is model-specific, but for attention-based models we expect that tokens in a context of <unk> tokens are treated differently.

B. Availability Attack

Machine-learning systems can be attacked by workloads that are unusually slow. The inputs generating such computations are known as sponge examples [17]. Originally, Shumailov et al. used a genetic algorithm to generate sponge examples of a given constant size. They found that they could slow down translation significantly, but the algorithmically-created sponge examples ended up being semantically meaningless. Indeed, these examples very often ended up using Chinese characters as inputs to models that were assuming inputs in other languages.

In this paper we show that sponge examples can be constructed in a targeted way, both with fixed and increased input size. For a fixed-size sponge example, an attacker can replace individual characters with characters that look just the same, but take longer to process. If an increase in input size is tolerable, the attacker can also inject invisible characters, forcing the model to take additional time to process these additional steps in its input sequence.

Such attacks may be carried out more covertly if the visual appearance of the input does not arouse users' suspicions. If

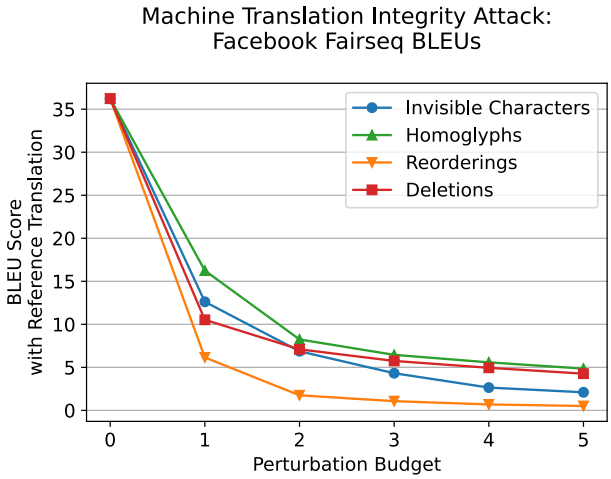

Fig. 7. BLEU Scores of Imperceptible Perturbations vs. Unperturbed WMT Data on Fairseq EN-FR model

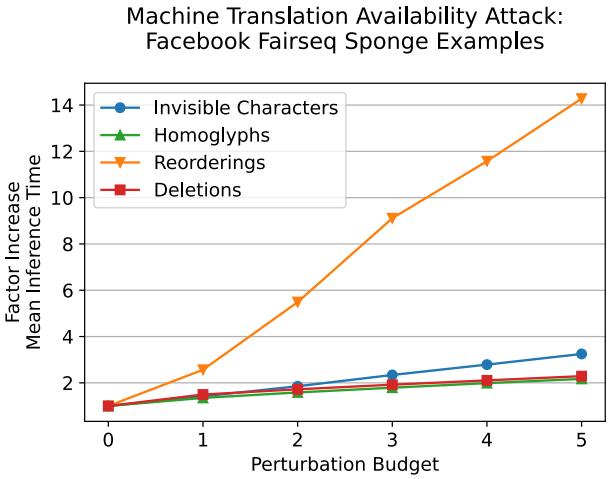

Fig. 8. Fairseq sponge example average inference time

launched in parallel at scale, the availability of hosted NLP models may be degraded, suggesting that a distributed denial-of-service attack may be feasible on text-processing services.

VI. Evaluation

A. Experiment Setup

We evaluate the performance of each class of imperceptible perturbation attack – invisible characters, homoglyphs, reorderings, and deletions – against three NLP tasks: machine translation, toxic content detection, and textual entailment classification. We perform these evaluations against a collection of three open-source models and three closed-source, commercial models published by Google, Facebook, Microsoft, and IBM. We repeat each experiment with perturbation budget values varying from zero to five.

All experiments were performed in a black-box setting in which unlimited model evaluations are permitted, but accessing the assessed model's weights or state is not permitted. This represents one of the strongest threat models for which attacks are possible in nearly all settings, including against commercial Machine-Learning-as-a-Service (MLaaS) offerings. Every model examined was vulnerable to imperceptible perturbation attacks. We believe that the applicability of these attacks should in theory generalize to any text-based NLP model without adequate defences in place.

We perform untargeted attacks against all three NLP tasks examined, and for textual-entailment classification we additionally perform targeted attacks both with output probability access and output label-only settings. We also perform sponge-example attacks against the translation task. The experiments were performed on a cluster of machines each equipped with a Tesla P100 GPU and Intel Xeon Silver 4110 CPU running Ubuntu.

For each class of perturbation, following Algorithm 1 we defined an objective function that seeks to maximize the distance between the assessed model's outputs for perturbed and unperturbed inputs, or, in the case of targeted attacks, seeks to maximize the classification probability of the selected target. We used Levenshtein distance in these objective functions, but in practice any metric may be chosen. We then performed differential evolution, finding that the optimization converged quickly, and thus used a population size of 32 with a maximum of 10 iterations in the genetic algorithm. Increasing these parameters further would likely allow an attacker to find even more effective perturbations; i.e. our experimental results obtain a lower bound.

For the objective functions used in these experiments, invisible characters were chosen from a set including ZWSP, ZWNJ, and ZWJ[17]; homoglyphs sets were chosen according to the relevant Unicode technical report[18]; reorderings were chosen from the sets defined using Algorithm 2, and deletions were chosen from the set of all non-control ASCII characters followed by a BKSP[19] character. We define the unit value of the perturbation budget as one injected invisible character, one homoglyph character replacement, one `Swap` sequence according to the reordering algorithm, or one ASCII-backspace deletion pair.

We have published a command-line tool written in Python to conduct these experiments as well as the entire set of adversarial examples resulting from these experiments.[20] We have also published an online tool for validating whether text may contain imperceptible perturbations and for generating random imperceptible perturbations.[21]

In the following sections, we describe each experiment in detail.

B. Machine Translation: Integrity

For the machine translation task, we used an English-French transformer model pre-trained on WMT14 data [54] published

[17] Unicode characters U+200B, U+200C, U+200D
[18] unicode.org/Public/security/latest/intentional.txt
[19] Unicode character U+8
[20] github.com/nickboucher/imperceptible
[21] imperceptible.ml

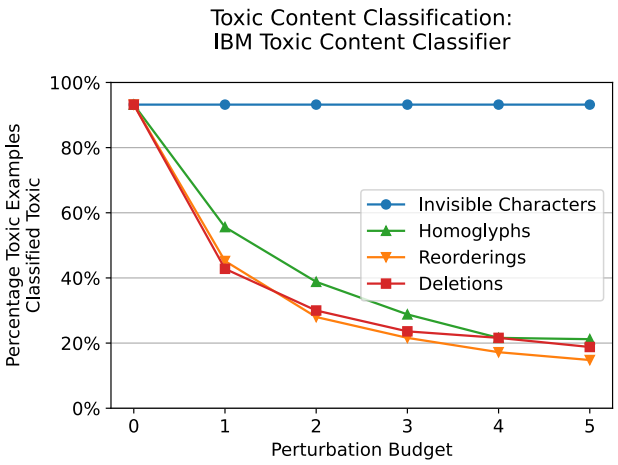

Fig. 9. Percentage of imperceptibly perturbed toxic sentences classified correctly in IBM's Toxic Content Classifier.

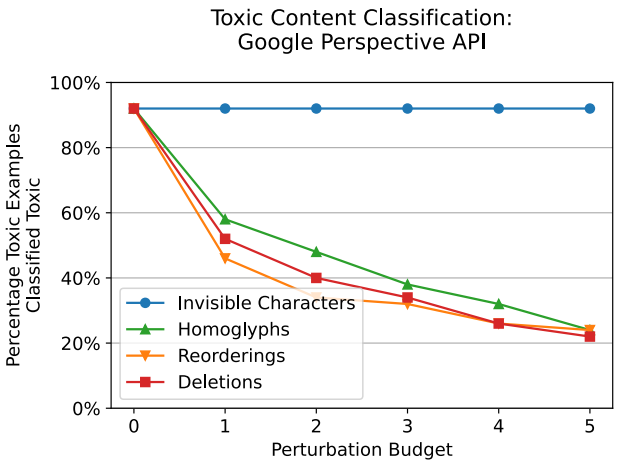

Fig. 10. Percentage of imperceptibly perturbed toxic sentences classified correctly in Google's Perspective API.

by Facebook as part of Fairseq [55], Facebook AI Research's open source ML toolkit for sequence modeling. We utilized the corresponding WMT14 test set data to provide reference translations for each adversarial example.

For the set of integrity attacks, we crafted adversarial examples for 500 sentences and repeated adversarial generation for perturbations budgets of 0 through 5. Each example took, on average, 432 seconds to generate.

For the adversarial examples generated, we compare the BLEU scores of the resulting translation against the reference translation in Figure 7. We also provide the Levenshtein distances between these values in Appendix Figure 18, which increase approximately linearly with reorderings having the largest distance.

C. Machine Translation: Availability

In addition to attacks on machine-translation model integrity, we also explored whether we could launch availability attacks. These attacks take the form of sponge examples, which are adversarial examples crafted to maximize inference runtime.

We used the same configuration as in the integrity experiments, crafting adversarial examples for 500 sentences with perturbation budgets of 0 to 5. Each example took, on average, 420 seconds to generate.

Sponge-example results against the Fairseq English-French model are presented in Figure 8, which shows that reordering attacks are by some ways the most effective. Although the slowdown is not as significant as Shumailov et al achieved by dropping Chinese characters into Russian text [17], our attacks are semantically meaningful and will not be noticeable to human eyes.

D. Machine Translation: MLaaS

In addition to the integrity attacks on Fairseq's open-source translation model, we performed a series of case studies on two popular Machine Learning as a Service (MLaaS) offerings: Google Translate and Microsoft Azure ML. These experiments attest to the real-world applicability of these attacks. In this setting, translation inference involves a web-based API call rather than invoking a local function.

Due to the cost of these services, we crafted adversarial examples targeting integrity for 20 sentences of budgets from 0 to 5 with a reduced maximum evolution iteration value of 3.

The BLEU results of tests against Google Translate are in Figure 13 and against Microsoft Azure ML in Figure 14. The corresponding Levenshtein results can be found in Appendix Figures 16 and 15.

Interestingly, the adversarial examples generated against each platform appeared to be meaningfully effective against the other. The BLEU scores of each service's adversarial examples tested against the other are plotted as dotted lines in Figure 14 and Figure 13. These results show that imperceptible adversarial examples can be transferred between models.

E. Toxic Content Detection

In this task we attempt to defeat a toxic-content detector. For our experiments, we use the open-source Toxic Content Classifier model[22] published by IBM. In this setting, the adversary has access to the classification probabilities emitted by the model.

For this set of experiments, we craft adversarial examples for 250 sentences labeled as toxic in the Wikipedia Detox Dataset [56] with perturbation budgets from 0 to 5. Each example took, on average, 18 seconds to generate.

IBM Toxic Content Classification perturbation results can be seen in Figure 9. Homoglyphs, reorderings, and deletions effectively degrade model performance by up to 75%, but, interestingly, invisible characters do not have an effect on model performance. This could be because invisible characters were present in the training data and learned accordingly, or, more likely, the model uses a tokenizer which disregards the invisible characters we used.

[22] github.com/IBM/MAX-Toxic-Comment-Classifier

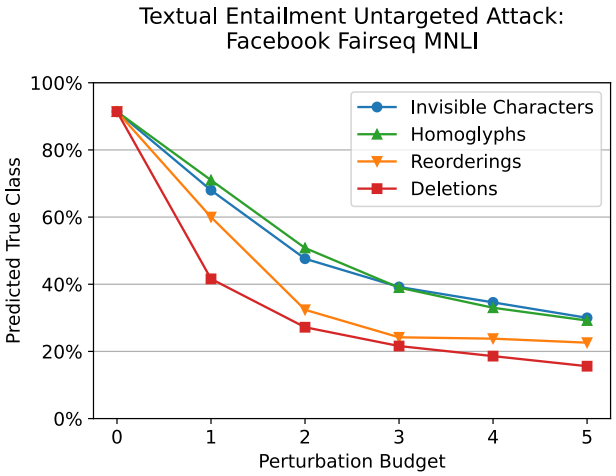

Fig. 11. Untargeted accuracy of Fairseq MNLI model with imperceptible perturbations

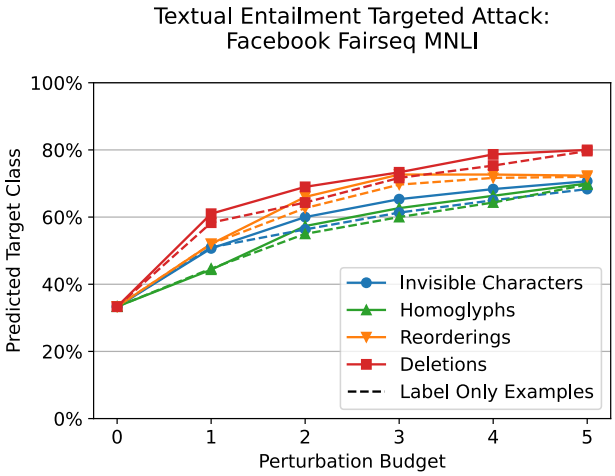

Fig. 12. Targeted accuracy of Fairseq MNLI model with imperceptible perturbations

F. Toxic Content Detection: MLaaS

We repeated the toxic content experiments against Google's Perspective API[23], which is deployed at scale in the real world for toxic content detection. We used the same experiment setting as in the IBM Toxic Content Classification experiments, except that we generated adversarial examples for 50 sentences. The results can be seen in Figure 10.

G. Textual Entailment: Untargeted

Recognizing textual entailment is a text-sequence classification task that requires labeling the relationship between a pair of sentences as entailment, contradiction, or neutral.

For the textual-entailment classification task, we performed experiments using the pre-trained RoBERTa model [57] fine-tuned on the MNLI corpus [58]. This model is published by Facebook as part of Fairseq [55].

For these textual-entailment integrity attacks, we crafted adversarial examples for 500 sentences and repeated adversarial generation for perturbation budgets of 0 through 5. The sentences used in this experiment were taken from the MNLI test set. Each example took, on average, 51 seconds to generate.

The results from this experiment are shown in Figure 11. Performance drops significantly even with a budget of 1.

H. Textual Entailment: Targeted

We repeated the set of textual-entailment classification integrity experiments with targeted attacks. For each sentence, we attempted to craft an adversarial example targeting each of the three possible output classes. Naturally, one of these classes is the correct unperturbed class, and as such we expect the budget $= 0$ results to be approximately 33% successful.

Due to the increased number of adversarial examples per sentence, we crafted adversarial examples for 100 sentences and repeating adversarial generation for perturbation budgets of 0 through 5. The results can be seen in Figure 12.

In the first set of targeted textual entailment experiments, we permitted the adversary to access the full set of logits output by the classification model. In other words, the differential evolution algorithm had access to the probability value assigned to each possible output class. We repeated the targeted textual entailment experiments a second time in which the adversary had access to the selected output label only, without probability values. These results are plotted as a dotted line in Figure 12. Label-only attacks appear to suffer only a slight disadvantage, and even this diminishes as perturbation budgets increase.

VII. Discussion

A. Ethics

We followed departmental ethics guidelines closely. We used legitimate, well-formed API calls to all third parties, and paid for commercial products. To minimize the impact both on commercial services and CO_2 production, we chose small inputs, maximum iterations and pool sizes. For example, while Microsoft Azure allows inputs of size 10,000, we used inputs of less than 50 characters [59]. Finally, we followed standard responsible disclosure processes.

B. Attack Potential

Imperceptible perturbations derived from manipulating Unicode encodings provide a broad and powerful class of attacks on text-based NLP models. They enable adversaries to:

- Alter the output of machine translation systems;
- Evade toxic-content detection;
- Invisibly poison NLP training sets;
- Hide documents from indexing systems;
- Degrade the quality of search;
- Conduct denial-of-service attacks on NLP systems.

These perturbations use valid albeit unusual encodings to fool NLP systems which assume common forms of encoding. The resulting vulnerabilities are clear enough when viewing

[23] perspectiveapi.com

text-based natural language processing systems through the lens of system security. Everyone who has worked on web applications knows not to take unconstrained user input as input to SQL queries, or even feed it into finite-length buffers. As NLP systems gain rapid acceptance in a wide range of applications, their developers are going to have to learn the hard lessons that operating-system developers learned from the Morris worm, and that web developers learned during the dotcom boom.

Perhaps the most disturbing aspect of our imperceptible perturbation attacks is their broad applicability: all text-based NLP systems we tested are susceptible. The adversarial implications may vary from one application to another and from one model to another, but all text-based models are based on encoded text, and all text is subject to adversarial encoding unless the coding is suitably constrained.

C. Search Engine Attack

Discrepancies between encoded bytes and their visual rendering affect searching and indexing systems. Search engine attacks fall into two categories: attacks on searching and attacks on indexing.

Attacks on searching result from perturbed search queries. Most systems search by comparing the encoded search query against indexed sets of resources. In an attack on searching, the adversary's goal is to degrade the quality or quantity of results. Perturbed queries interfere with the comparisons.

Attacks on indexing use perturbations to hide information from search engines. Even though a perturbed document may be crawled by a search engine's crawler, the terms used to index it will be affected by the perturbations, making it less likely to appear from a search on unperturbed terms. It is thus possible to hide documents from search engines "in plain sight." As an example application, a dishonest company could mask negative information in its financial filings so that the specialist search engines used by stock analysts fail to pick it up.

D. Defences

We propose a variety of defences against imperceptible perturbation attacks. These defences will not solve every encoding-related issue that arises in NLP; there is a blurry line between encoding issues and general input noise, such as misspellings, which represent an open problem. Our defences can, however, significantly reduce the attack surface that current NLP systems expose to non-standard encodings.

Given that the conceptual source of this attack stems from differences in logical and visual text encoding representation, one catch-all solution is to render all input, interpret it with optical character recognition (OCR), and feed the output into the original text model. This general-purpose defence incurs the computational and energy costs of inference against an additional model, which may not be suitable in some applications. We therefore explore additional defences that may be more appropriate for certain settings.

TABLE III
TEXT MIXING LATIN AND CYRILLIC LINGUISTIC FAMILIES.

Interword Mixing	Intraword Mixing
Hello папа	Hello папа

1) Invisible Character Defences: Generally speaking, invisible characters do not affect the semantic meaning of text, but relate to formatting concerns. For many text-based NLP applications, removing a standard set of invisible characters from the input string prior to inference would block invisible character attacks.

If the application requirements do not allow it to discard such characters, they will have to be dealt with somehow. If there are linguistic reasons why some invisible characters cannot be ignored during inference, the tokenizer must be include them in the source-language dictionary, resulting in an embedding vector other than `<unk>`.

2) Homoglyph Defences: Homoglyph sets typically arise from the fact that Unicode contains many alphabets, some of which have similar characters. While multilingual speakers will often mix words and phrases from different languages in the same sentence, it is very rare for characters from different languages to be used within the same word. That is, interword linguistic family mixing is common, but intraword mixing is much less so. For example, see Table III.

Conveniently, the Unicode specification divides code points into distinct, named blocks such as "Basic Latin". At design time, a model designer can group blocks into linguistic families. But what do you do when you find an input word with characters from multiple linguistic families? If you discard it, that itself creates an attack vector. In many applications, the robust course of action might be to halt and sound an alarm. If the application doesn't permit that, an alternative is to retain only characters from a single linguistic family for each word, mapping all intraword-mixed characters to homoglyphs in the dominant linguistic family.

3) Reordering Defences: For some text-based NLP models with a graphical user interface, reordering attacks can be prevented by stripping all Bidi override characters as the input is displayed to the active user. But this will not work for interfaces that lack a visual user interface, or which mix left-to-right languages such as English with right-to-left ones such as Hebrew. In such applications, it may be necessary to return a warning with the model's output if Bidi override characters were detected in the input.

4) Deletion Defences: We suspect that there may not be many use cases where deletion characters are a valid input into a model. If users enter text via a normal graphical form field, deletion characters would be processed by the text-rendering engine before typed text is passed to the model.

However, if an adversary is able to directly inject encoded text into a model, some attention must be given to deletion attacks. One possible defence would be to pre-process model inputs such that deletion characters are actioned before the

model processes the input. Alternatively, the model could throw an error when deletion characters are detected.

VIII. Conclusion

Text-based NLP models are vulnerable to a broad class of imperceptible perturbations which can alter model output and increase inference runtime without modifying the visual appearance of the input. These attacks exploit language coding features, such as invisible characters and homoglyphs. Although they have been seen occasionally in the past in spam and phishing scams, the designers of the many NLP systems that are now being deployed at scale appear to have ignored them completely.

We have presented a systematic exploration of text-encoding exploits against NLP systems. We have developed a taxonomy of these attacks and explored in some detail how they can be used to mislead and to poison machine-translation, toxic content detection, and textual entailment classification systems. Indeed, they can be used on any text-based ML model that processes natural language. Furthermore, they can be used to degrade the quality of search engine results and hide data from indexing and filtering algorithms.

We propose a variety of defenses against this class of attacks, and recommend that all firms building and deploying text-based NLP systems implement such defenses if they want their applications to be robust against malicious actors.

Acknowledgment

The second author was partially supported with funds from Bosch-Forschungsstiftung im Stifterverband. We would also like to thank Adelin Travers for help with natural language annotation, Markus Kuhn for help with unicode magic, and Darija Halatova for help with visualizations.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[3] C. Knight, "Evasion with unicode format characters," in *SpamAssassin - Dev*, 2018.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[5] D. A. Smith, R. Cordel, E. M. Dillon, N. Stramp, and J. Wilkerson, "Detecting and modeling local text reuse," in *IEEE/ACM Joint Conference on Digital Libraries*, 2014, pp. 183–192.

[6] A. Schmidt and M. Wiegand, "A survey on hate speech detection using natural language processing," in *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 1–10. [Online]. Available: https://www.aclweb.org/anthology/W17-1101

[7] N. Papernot, P. D. McDaniel, A. Swami, and R. E. Harang, "Crafting adversarial input sequences for recurrent neural networks," *CoRR*, vol. abs/1604.08275, 2016. [Online]. Available: http://arxiv.org/abs/1604.08275

[8] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," *CoRR*, vol. abs/1711.02173, 2017. [Online]. Available: http://arxiv.org/abs/1711.02173

[9] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *2018 IEEE Security and Privacy Workshops (SPW)*, May 2018, pp. 50–56.

[10] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "HotFlip: White-box adversarial examples for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 31–36. [Online]. Available: https://www.aclweb.org/anthology/P18-2006

[11] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "Adversarial example generation with syntactically controlled paraphrase networks," *CoRR*, vol. abs/1804.06059, 2018. [Online]. Available: http://arxiv.org/abs/1804.06059

[12] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1BLjgZCb

[13] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2890–2896. [Online]. Available: https://www.aclweb.org/anthology/D18-1316

[14] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/textbugger-generating-adversarial-text-against-real-world-applications/

[15] P. Michel, X. Li, G. Neubig, and J. Pino, "On evaluation of adversarial perturbations for sequence-to-sequence models," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 3103–3114. [Online]. Available: https://www.aclweb.org/anthology/N19-1314

[16] S. Ren, Y. Deng, K. He, and W. Che, "Generating natural language adversarial examples through probability weighted word saliency," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 1085–1097. [Online]. Available: https://www.aclweb.org/anthology/P19-1103

[17] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, "Sponge examples: Energy-latency attacks on neural networks," in *Proceedings of the 6th IEEE European Symposium on Security and Privacy, Vienna, Austria, September 6-10, 2021*. IEEE, 2021. [Online]. Available: https://arxiv.org/abs/2006.03463

[18] W. Zou, S. Huang, J. Xie, X. Dai, and J. Chen, "A reinforced generation of adversarial examples for neural machine translation," 2020.

[19] S. Frenkel, "Facebook is failing in global disinformation fight, says former worker," *New York Times*, Sep 14 2020.

[20] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, "A taxonomy and terminology of adversarial machine learning."

[21] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2019.

[22] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[23] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 15–26.

[24] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter." *LEET*, vol. 8, pp. 1–9, 2008.

[25] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 19–35.

[26] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 497–514. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/hong

[27] C. A. C. Choo, F. Tramer, N. Carlini, and N. Papernot, "Label-only membership inference attacks," 2020.

[28] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[29] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[30] W. Weaver, "Translation," in *Machine translation of languages: fourteen essays*. Cambridge, MA: Technology Press of the Massachusetts Institute of Technology, Jul. 1949. [Online]. Available: https://repositorio.ul.pt/bitstream/10451/10945/2/ulfl155512_tm_2.pdf

[31] B. J. Dorr, P. W. Jordan, and J. W. Benoit, "A Survey of Current Paradigms in Machine Translation," MARYLAND UNIV COLLEGE PARK INST FOR ADVANCED COMPUTER STUDIES, Tech. Rep. LAMP-TR-027, Dec. 1998. [Online]. Available: https://apps.dtic.mil/docs/citations/ADA455393

[32] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1700–1709. [Online]. Available: https://www.aclweb.org/anthology/D13-1176

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[34] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

[35] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[36] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *CoRR*, vol. abs/1508.07909, 2015. [Online]. Available: http://arxiv.org/abs/1508.07909

[37] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[38] J. Wieting, J. Mallinson, and K. Gimpel, "Learning paraphrastic sentence embeddings from back-translated bitext," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 274–285. [Online]. Available: https://www.aclweb.org/anthology/D17-1026

[39] J. X. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, "Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp," 2020.

[40] T. U. Consortium, "The Unicode Standard, Version 13.0," Mar. 2020. [Online]. Available: https://www.unicode.org/versions/Unicode13.0.0

[41] ——, "Unicode Security Considerations," The Unicode Consortium, Tech. Rep. Unicode Technical Report #36, Sep. 2014. [Online]. Available: https://www.unicode.org/reports/tr36/tr36-15.html

[42] G. Simpson, T. Moore, and R. Clayton, "Ten years of attacks on companies using visual impersonation of domain names," in *APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2020.

[43] E. Gabrilovich and A. Gontmakher, "The homograph attack," *Commun. ACM*, vol. 45, no. 2, p. 128, Feb. 2002. [Online]. Available: https://doi.org/10.1145/503124.503156

[44] T. Holgers, D. E. Watson, and S. D. Gribble, "Cutting through the confusion: A measurement study of homograph attacks," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06. USA: USENIX Association, 2006, p. 24.

[45] MITRE, "CAPEC-632: Homograph Attack via Homoglyphs (Version 3.4)," MITRE, Common Attack Pattern Enumeration and Classification 632, Nov. 2015. [Online]. Available: https://capec.mitre.org/data/definitions/632.html

[46] H. Suzuki, D. Chiba, Y. Yoneya, T. Mori, and S. Goto, "Shamfinder: An automated framework for detecting idn homographs," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 449–462. [Online]. Available: https://doi.org/10.1145/3355369.3355587

[47] T. U. Consortium, "Unicode Bidirectional Algorithm," The Unicode Consortium, Tech. Rep. Unicode Technical Report #9, Feb. 2020. [Online]. Available: https://www.unicode.org/reports/tr9/tr9-42.html

[48] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, 2007, pp. 177–180.

[49] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016.

[50] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *CoRR*, vol. abs/1508.07909, 2015. [Online]. Available: http://arxiv.org/abs/1508.07909

[51] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997. [Online]. Available: https://doi.org/10.1023/A:1008202821328

[52] T. U. Consortium, "Unicode Security Considerations," The Unicode Consortium, Tech. Rep. Unicode Technical Report #39, Feb. 2020. [Online]. Available: https://www.unicode.org/reports/tr39/tr39-22.html

[53] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[54] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," in *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1–9. [Online]. Available: https://www.aclweb.org/anthology/W18-6301

[55] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[56] N. Thain, L. Dixon, and E. Wulczyn, "Wikipedia talk labels: Toxicity," Feb 2017. [Online]. Available: https://figshare.com/articles/dataset/Wikipedia_Talk_Labels_Toxicity/4563973/2

[57] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[58] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018, pp. 1112–1122. [Online]. Available: http://aclweb.org/anthology/N18-1101

[59] M. Azure, "Request limits for translator." [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/translator/request-limits

APPENDIX

A. Machine Translation MLaaS Results

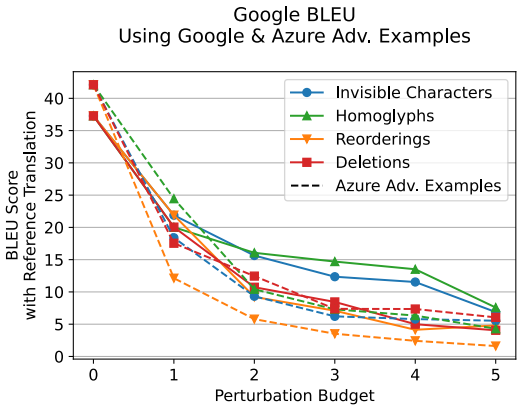

Fig. 13. BLEU Scores of Azure's imperceptible adversarial example on Google Translate

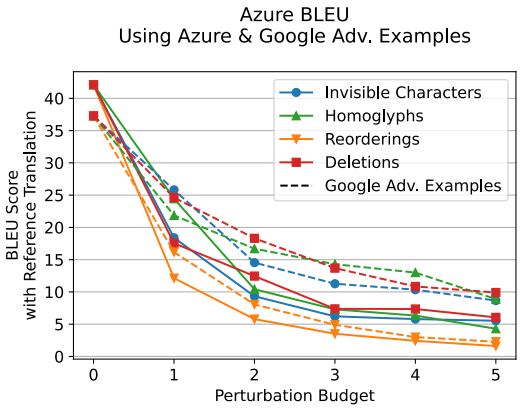

Fig. 14. BLEU Scores of Google Translate's imperceptible adversarial example on Microsoft Azure

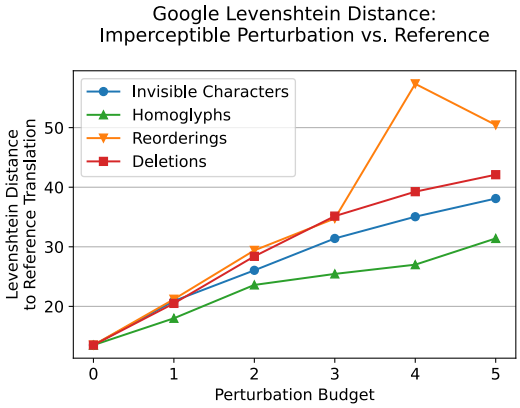

Fig. 15. Levenshtein Distances Between Imperceptible Perturbations and Unperturbed WMT Data on Google Translate's EN-FR model

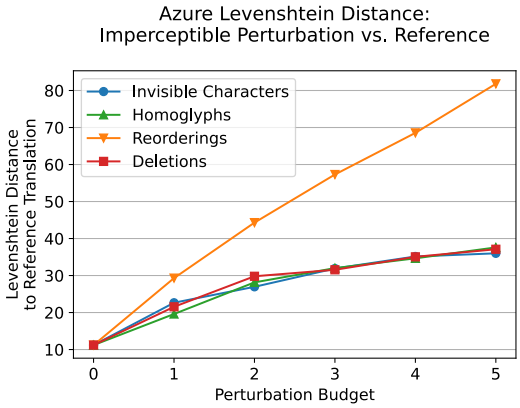

Fig. 16. Levenshtein Distances Between Imperceptible Perturbations and Unperturbed WMT Data on Microsoft Azure's EN-FR model

B. Machine Translation Fairseq Levenshtein Distances

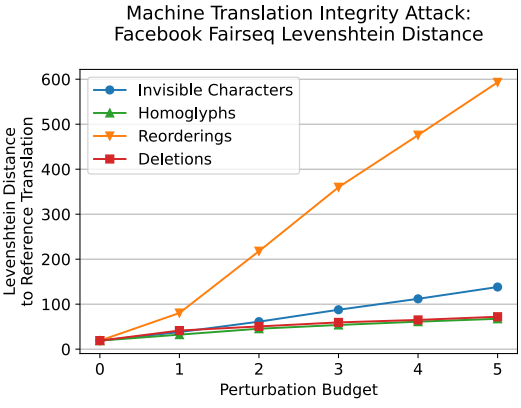

Fig. 17. Levenshtein Distances Between Integrity Attack Imperceptible Perturbations and Unperturbed WMT Data on Fairseq EN-FR model

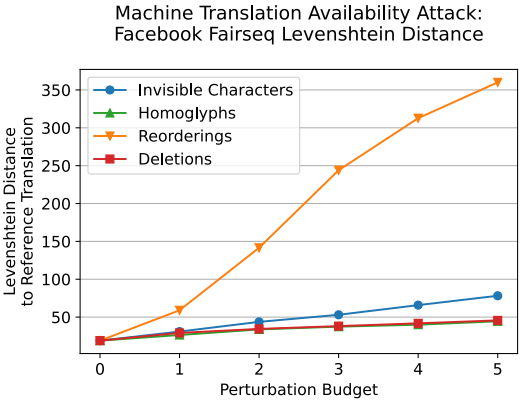

Fig. 18. Levenshtein Distances Between Availability Attack Imperceptible Perturbations and Unperturbed WMT Data on Fairseq EN-FR model