Bad Character Injection: Imperceptible Attacks on NLP Models

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

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

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

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

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 were easily noticeable. In this paper, we explore a large class of adversarial examples that can be used to attack text-based models, while still preserving the meaning of the text. We use encoding-specific input perturbations that are imperceptible to the human eye to modify the output of a very wide range of Natural Language Processing (NLP) systems, from neural machine translation pipelines to web search engines. We find that with a single character insertion, an attacker can significantly reduce the performance of one modern transformer model, and by inserting three characters they can degrade most of them. Our attacks work against currently deployed commercial systems – both Google and Microsoft are affected. 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 more conventional applications; and given that such systems are now being deployed rapidly at scale, their architects and operators need to pay urgent attention to this.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Security and privacy → Software security engineering**; **Domain-specific security and privacy architectures**; • **Information systems →** Information retrieval query processing.

KEYWORDS

adversarial machine learning, NLP, text-based models, text encodings, search engines

ACM Reference Format:
Nicholas Boucher*, Ilia Shumailov*, Nicolas Papernot, and Ross Anderson. 2021. Bad Character Injection: Imperceptible Attacks on NLP Models. In *CCS '21: ACM Conference on Computer and Communications Security, November 14–19, 2021, Seoul, South Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/DOI

1 INTRODUCTION

Do x and х 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 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. Such attacks initially targeted visual models used in image classification, 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. Such perturbations are imperceptible to users of text-based systems, but the bytes used to encode them can change the output drastically.

We have found that machine-learning models which 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

[1]Unicode character U+200D
[2]translate.google.com

Table 1: Examples of Imperceptible Perturbations in Machine Translation

Input	Visible	Google En→Fr	Back translation
Send money to account 4321	4321	Envoyer de l'argent sur le compte 4321	Send money to account 4321
Send money to account 9U+00084321	4321	Veuillez envoyer de l'argent au 94321	Send money to account 94321
Send money to account U+202B1U+20282U+20283U+202B4	4321	Veuillez envoyer de l'argent au 1234	Send money to account 1234

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 [22]. Although such tricks were known to engineers designing spam filters, they were not a main concern. However the rapid deployment of NLP systems in a large range of applications from machine translation through copyright enforcement to the filtering of hate speech is suddenly creating a host of high-value targets with capable motivated opponents.

The main contribution of this work is to explore and develop a class of 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 first, the long history of attacks on a variety of systems that exploited unsanitized input, and second, that a number of the NLP[3] systems we study are provided by firms that also provide mail services with spam filters. We give some examples of both targeted and untargeted attacks in Table 1.

Our findings present an attack vector that must be considered when designing any system processing natural language with text-based inputs that may ingest modern text encodings, whether directly from a text API or by parsing documents to extract text. We then explore a series of defences that can give some protection against this powerful set of attacks. Defence is not entirely straightforward, though, as many applications may not be able to simply discard all words from foreign languages or even all control characters. We discuss the details later.

This paper makes the following contributions:

- We present a novel class of imperceptible perturbations for NLP models;
- We present four variants of imperceptible attack against both the integrity and availability of NLP models;
- We show how our imperceptible attacks can degrade the performance of the model to zero and slow it down up to a factor of two with just a handful of character substitutions;
- We evaluate our attacks extensively against a Fairseq translator and against currently deployed Machine Learning as a Service (MLaaS) at Microsoft Azure and Google Cloud;
- We find that all modern MLaaS systems we tested are vulnerable to our attacks;
- We present some defences against these attacks and discuss why defence can be complex.

[3]It is worth noting that NLP is one of a few domains where it is expected for the model to parse inputs it may know nothing about. Computer vision engineers would not ask their ML model to operate directly on raw bytes and learn to parse encoding headers.

2 MOTIVATION

People have already experimented with adversarial attacks on NLP models. 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, they look out of place, while paraphrasing 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 two examples. First, as a 'crypto' example, suppose that Alice and Bob are divorcing, and are agreeing instructions to send to their agent Carlos about the sale of their holiday home, knowing that he will read them through a translation service. Bob cheats Alice by creating an instruction "Pay half the proceeds to Alice's lawyer's account no. 123" in such a way that Google Translate will render it as a different account number, causing Carlos to send the money to Bob instead. Second, as a real-world example, consider a nation-state whose language is not spoken by the staff at large social media companies responsible for content moderation, as described by the Facebook whistleblower Sophie Zhang [16]. If the government wanted to make it difficult for them to block a campaign to incite attacks on minorities, it could prevent the machine translation of inflammatory sentences. Indeed, even developed nation states rely on machine-translation systems to monitor foreign media as they don't employ enough foreign linguists.

Furthermore, the same perturbations would prevent proper search engine indexing, thus making malicious content hard to locate in the first place. We found that search engines do not parse out invisible characters and can be maliciously targeted with well-crafted queries. At the time of writing, Googling "The meaning of life" returns approximately 990 million results. However, searching for the visually identical string "The meaning of life" (containing 250 invisible "zero width joiner" characters[4]) returns exactly none.

Although in this paper we target NLP models specifically, our text-encoding attacks are not limited to machine-translation models. Search engines are vulnerable too, regardless of whether they use machine learning or more traditional indexing methods. We expect that the same techniques can be used to smuggle hate speech past content filters, though for ethical reasons we have not performed such experiments.

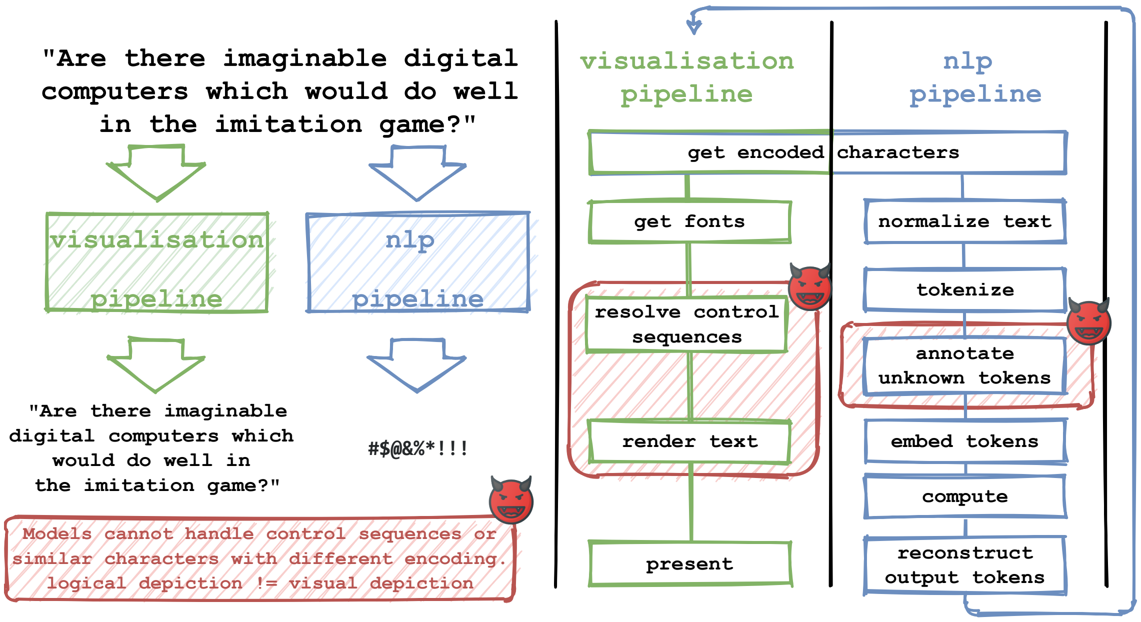

"Are there imaginable digital computers which would do well in the imitation game?"
visualisation pipeline
nlp pipeline
"Are there imaginable digital computers which would do well in the imitation game?"
#$@&%*!!!
Models cannot handle control sequences or similar characters with different encoding. logical depiction != visual depiction
visualisation pipeline
nlp pipeline
get encoded characters
get fonts
resolve control sequences
render text
present
normalize text
tokenize
annotate unknown tokens
embed tokens
compute
reconstruct output tokens

Figure 1: shows typical visualisation and NLP processing pipelines. The latter are built to ignore unknown characters, often replacing them with <unk> tokens during computation. The gap can be exploited by an attacker who inserts control sequences or homoglyphs.

3 RELATED WORK

3.1 Adversarial Examples

Machine-learning techniques are vulnerable to many large classes of attacks [42], with one major class being adversarial examples. These are inputs to models which, during inference, cause the model to output an incorrect result [41]. 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 [17, 25, 41]. In the black-box setting, where the model is unknown, the enemy can transfer adversarial examples from another model [31], or approximate gradients by observing output labels and confidence [7].

Training data can also be poisoned to manipulate the accuracy of the model for specific inputs [20, 28]. Bitwise errors can be introduced during inference to reduce the model's performance [18]. Inputs can also be chosen to maximize the time or energy a model takes during inference [37], or to expose confidential training data via inference techniques [9]. In other words, adversarial examples can affect the *integrity*, *availability* and *confidentiality* of machine-learning systems [6, 32, 37].

3.2 NLP Models

Natural language processing (NLP) systems are designed to process human language. Machine translation was proposed as early as 1949 [44] 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 [14], eventually yielding to neural networks [4], then recurrent neural networks (RNNs) because of their ability to reference past context [21]. 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 [43].

Transformers are a form of encoder-decoder model [8, 40] and 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 [35].

3.3 Adversarial NLP

Early adversarial ML research focused on image classification [5, 17], and the search for adversarial examples in NLP systems began later, targeting sequence models [33]. Adversarial examples

[4]Unicode character U+200D

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 both 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 [26]. Researchers have proposed using word-based input swaps with synonyms [34] or character-based swaps with semantic constraints [15]. These methods aim to constrain the perturbations to a set of transformations that a human is less likely to notice. Neural machine-translation models generally perform poorly on noisy inputs such as misspellings [3], but such perturbations 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 of large corpora of text [45], and used to systematically generate adversarial examples for machine translation systems [19]. One can also search for neighbors of the input sentence in an embedded space [47]; these examples often result in low-performance translations, making them good candidates for adversarial examples. Although paraphrasing can indeed help preserve semantics, humans can often see it. 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 [1]. Reinforcement learning can be used to efficiently generate adversarial examples for translation models [48]. There have even been efforts to combine academic NLP adversarial techniques into easily consumable toolkits available online [27], making these attacks relatively easy to use.

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>` [26]. However, all methods proposed for generating `<unk>` use visible characters.

3.4 Unicode

Unicode is a character set designed to standardize the electronic representation of text [13]. 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 which 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.

3.5 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 [10].

The 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 lowercase Latin 'l'[5] is often rendered near-identical to the uppercase Latin 'I'[6]. 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 [39]. 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*[7]. 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 a[8]) to its Punycode equivalent *xn–pypl-53dc.com* to highlight a potentially dangerous ambiguity. However, Punycode can introduce new opportunities for deception. For example, consider the URL *xn–google.com*, which decodes to 䕮䕵䕶䕱.com. Furthermore, Punycode does not solve cross-script homoglyph encoding vulner-abilities 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, means we need a system for managing character order with mixed character sets. For Unicode, this is the Bidirectional (Bidi) Algorithm [11]. Unicode specified 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.

[5]ASCII value `0x6C`
[6]ASCII value `0x49`
[7]https://www.zdnet.com/article/paypal-alert-beware-the-paypai-scam-5000109103
[8]Unicode character U+0430

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 generalise. While the Unicode Consortium does publish a set of software components for Unicode support[9], many operating systems, platforms, and other software ecosystems have different implementations. For example, GNOME produces Pango[10], Apple produces Core Text[11], while Microsoft produces a Unicode implementation for Windows[12].

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.

4 BACKGROUND

4.1 Attack Taxonomy

In this paper, we explore the class of imperceptible attacks based on Unicode and other coding 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 text-based NLP models.

We define these *imperceptible perturbations* as modifications of an encoded string of text which result in either:

- No modifications by a correctly implemented rendering engine in rendering the perturbed input compared to the unperturbed input for the underlying text encoding[13], or
- 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 between images of the renderings of two strings via a computer vision model . We formally define attack conditions in Section 4.3.

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

1. **Invisible Characters**: Valid characters which have no visible rendered glyph are used to adversarially perturb the input to a model without changing the rendering at all.
2. **Homoglyphs**: Unique characters which render to the same or visually similar glyphs are used to adversarially perturb the input to a model.
3. **Reorderings**: Directionality control characters are used to override the default rendering order of glyphs, allowing adversarial reordering of encoded bytes input to a model.
4. **Deletions**: Deletion control characters, such as the backspace, are injected into a string to remove nearby characters from its visual rendering.

This class of imperceptible text-based attacks on NLP models represents an abstract class of attacks independent of different text encoding implementations. For the purpose of concrete examples and experimental results, we will assume the common Unicode encoding scheme, but the results presented go across to any other scheme with a sufficiently large character and control-sequence set.

There exist further classes of similar attacks which use typos or paraphrasing, rather than visual equivalence, and have already been discussed in the literature [19]. Such perturbation attacks do not meet our strict definition of imperceptibility, but can be used in parallel with the attacks discussed in this paper.

The imperceptible text-based attacks described in this paper can be used against a broad range of NLP models. For machine translation, as we describe in Section 4.3, imperceptible perturbations can be used to manipulate the target translation. For search engines, as we discuss in Section 7.3, they can be used both to degrade the quality of search results and to manipulate the indexing of target documents. For NLP models of all kinds, they can be used to generate sponge examples [37] – inputs which maximize runtime – for denial-of-service (DoS) attacks. They can be used against NLP components of larger systems; for example, we have used them to stop forum software automatically translating comments made in a minority language, thereby retaining the privacy that speakers of that language historically enjoyed against members of the majority community. Attacks can be as simple to implement as copying invisible substrings.

4.2 NLP Pipeline

Modern NLP pipelines (as is shown in blue in Figure 1) have evolved over the past decades and now include a large number of different optimisations to improve performance. Before beginning model inference, text-based input undergoes a number of preprocessing steps. Typically a *tokenizer* is applied first to separate words and punctuation in a task-meaningful way. For the experiments in this paper, we use a common tokenizer from the Moses toolkit [23]. Tokenized words are then encoded. Early models performed encoding with dictionaries mapping tokens to encoded embeddings. In this setting, tokens not seen during training are replaced with a special `<unk>` embedding. Many modern models now apply Byte Pair Encoding (BPE) before dictionary lookups. BPE, a common data compression technique, identifies common subwords in tokens. This often results in increased performance, as it allows the model to capture additional knowledge about language semantics from morphemes [36]. Each of the pre-processing methodologies described above is commonly used in modern deployed NLP models.

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

4.3 Attack Methodology

We approach the generation of adversarial samples as an optimisation 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, for translation tasks such as WMT we assume Y to be a

[9]http://site.icu-project.org
[10]https://pango.gnome.org
[11]https://developer.apple.com/documentation/coretext
[12]https://docs.microsoft.com/en-us/windows/win32/intl/unicode
[13]Note that correctness refers to compliance to a standard, i.e. the Unicode standard.

sequence of characters, whereas for categorization tasks such as MNLI we assume three-category output. Furthermore, we assume a strong black-box threat model where adversaries have access to model output but cannot observe the internals of the system. 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 also fixes the output of $f(p(\mathbf{x}))$ to match the target $\mathbf{T}$.
- **Availability Attack**: The adversary aims to find p such that $time(f(p(\mathbf{x}))) >> time(f(\mathbf{x}))$, where $time$ measures the runtime of f.

Furthermore, we define two additional constraints on the perturbation function p:

- **Budget**: A budget b such that $dist(\mathbf{x}, p(\mathbf{x})) < b$. The function $dist$ may refer to any distance metric, such as Levenshtein distance.
- **Visual Budget**: A visual budget v such that $visualdist(\mathbf{x}, p(\mathbf{x})) < v$. The function $visualdist$ measures visual similarity between renderings of two texts, such as the VGG perceptual loss between string renderings.

The attack algorithm is shown in Algorithm 1. We define the optimization problem as finding a set of operations over input text, where each operation corresponds to the injection of one short sequence of Unicode characters that performs a single imperceptible perturbation of the chosen subclass. Due to the discrete nature of the operation we use a gradient-free optimisation method – differential evolution. This randomly initializes a set of candidates and evolves them over many iterations, ultimately selecting the best-performing traits.

4.4 Invisible Characters

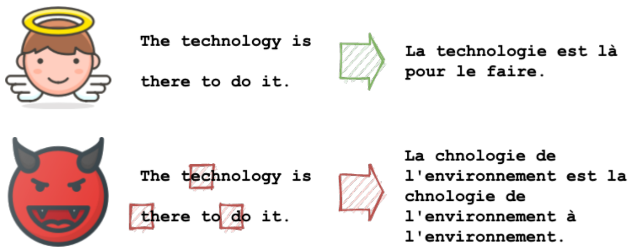

The technology is there to do it.
La technologie est là pour le faire.
The technology is there to do it.
La chnologie de l'environnement est la chnologie de l'environnement à l'environnement.

Figure 2: Attack with invisible characters. Input is on the left, whereas resulting model output is on the right of the arrows. The attacker adds imperceptible spaces between characters as indicated by the red boxes. For example, the first box shows that a hidden characters between 'e' and 'c'.

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[14] (ZWSP).

Examples of an attack using invisible characters are shown in Figure 2. It is important to note that characters lacking a glyph definition for a specific font do not automatically render 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 targeted at 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 'other' characters by printing the corresponding Unicode code point as base 10 numerals, in which case even the zero-width space character would be visible. 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[15] 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.

Although invisible characters do not produce a rendered glyph, they nevertheless represent a valid 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 to a human user. We found that these bytes alter the 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.

4.5 Homoglyphs

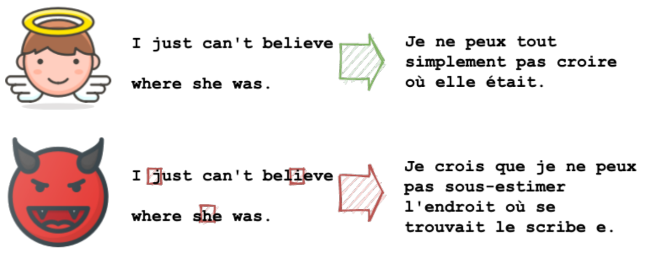
Homoglyphs are characters that render to the same glyph or to a visually similar one. This often occurs when portions of the same written script are used across different language families. For example, consider the Latin letter 'A' used in English. The very similar character 'A' is used in the Cyrillic 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 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 along with the Unicode Security Mechanisms technical report [12]

[14]Unicode character U+200B

[15]http://unifoundry.com/unifont/index.html



I just can't believe where she was.

Je ne peux tout simplement pas croire où elle était.

I ϳust can't belіeve where sһe was.

Je crois que je ne peux pas sous-estimer l'endroit où se trouvait le scribe e.

Figure 3: Attack with homoglyphs. Input is on the left, whereas resulting model output is on the right of the arrows. The attacker replaces characters with similar looking ones highlighted by the red boxes. Here, the attacker replaces 'j' with U+03F3, 'i' with U+0456 and 'h' with U+04BB.

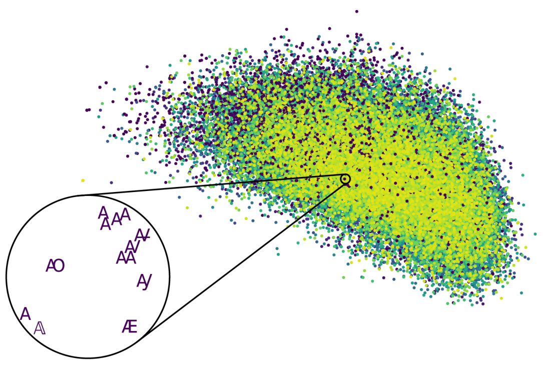


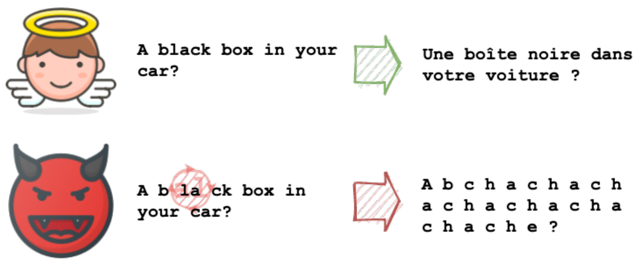
Figure 4: Clustering of Unicode homoglyphs according to the Unicode Security Confusables document, plotted as a 2D PCA of Unifont glyph images via a VGG16 model.

to draw attention to similarly rendered characters. The first[16] 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[17] 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 [38] 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 reproducability.

[16]https://www.unicode.org/Public/security/latest/intentional.txt
[17]https://www.unicode.org/Public/security/latest/confusables.txt

4.6 Reorderings



A black box in your car?

Une boîte noire dans votre voiture ?

A b la ck box in your car?

A b c h a c h a c h a c h a c h a c h a c h a c h e ?

Figure 5: Attack with reordering characters. Input is on the left, whereas resulting model output is on the right of the arrows. The attacker adds characters in an inversed order and then uses Bidi reverse direction control sequences to rotate the text. When rendered, although characters are reversed in order, but the model replaces reverses with <unk>, preserving the text appearing in the wrong order.

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 [11] 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 reorder-control characters is shown in Figure 5. In an adversarial setting, Bidi override characters allow the encoded ordering of characters to be modified and still render the same way. These reorderings are a form of imperceptible perturbation in the context of inputs to text-based models. We will show that they affect the performance of text-based NLP models.

Unlike invisible-character and homoglyph attacks, the subclass of reordering attacks is not font-dependent, but relies on the implementation of the Unicode Bidi Algorithm. (Technically, a font could choose to render Bidi override characters, making this attack font-dependent, but this is unlikely as these characters are not intended to be rendered directly.) It is also possible that there are bugs in a given system's implementation of the Bidi algorithm, but exploiting bugs is beyond the scope of this paper[18].

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. Table 3 in the Appendix illustrates an example of the ways a fixed visual string can be reordered through manipulation of the Bidi algorithm.

[18]Indeed we find that Pango often disagrees with CoreText on how particular control sequences are rendered.

4.7 Deletions

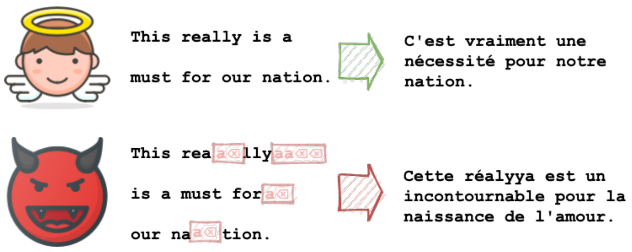

This really is a
must for our nation.
C'est vraiment une nécessité pour notre nation.
This reaa⌫llyaa⌫⌫
is a must fora⌫
our naa⌫tion.
Cette réalyya est un incontournable pour la naissance de l'amour.

Figure 6: Attack with backspace characters. Input is on the left, whereas resulting model output is on the right of the arrows. The attacker adds characters and follows them with U+0008 backspaces. When rendered, such characters are removed, but the model replaces backspaces with <unk>, preserving the text appearing prior.

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 examples of an attack using backspace characters is shown in Figure 6. Deletion characters are not generally font-dependent, as Unicode does not allow glyph specification for the basic control characters inherited from ASCII including BS, DEL, and CR. They are also not dependent on uncommon features in text rendering engines, so this attack is generally platform-independent. But it may be hard to exploit because most systems do not copy deleted text to clipboards or rendered it invisibly to the screen. So an attack using deletion perturbations generally requires an adversary to infiltrate Unicode sequences directly into a model input, rather than using standard copy+paste functions.

5 NLP ATTACKS

5.1 Integrity Attack

Machine-translation models are a prime target for imperceptible perturbation attacks. These models typically transform text-based input into an embedded space via an embedded vector transformation upon which character perturbations have a direct affect.

Regardless of the tokenizer or dictionary used in the embedding, the system is unlikely to handle Unicode perturbations gracefully in the absence of specific defences. 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 not ignored by most models and affects the output translation. In fact, it negatively affects the performance of the translation model.

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 the model's performance. The specific cause is model-specific, but for attention-based models we expect that tokens in a context of unknown tokens are treated differently. For large perturbations, the embedded input to the model may diverge so far from the unperturbed input that the performance gap is explained by adversarial input values resulting in exception handling, or in slower processing for other reasons.

5.2 Availability Attack

Machine-learning systems can be attacked by workloads that are unusually slow. The inputs generating such computations are known as sponge examples [37]. 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 the users' suspicions. A limit is imposed by the fact that when large batches of sponge examples are input into systems hosting NLP models, and the availability of these models is diminished, the operators may eventually notice a denial-of-service attack.

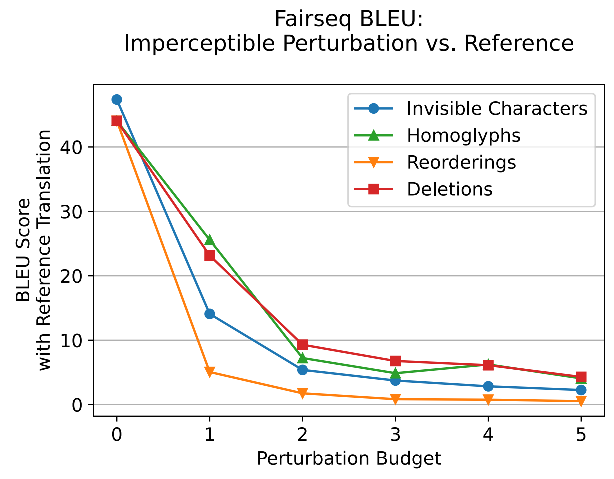


Figure 7: BLEU Scores of Imperceptible Perturbations vs. Unperturbed WMT Data on Fairseq EN-FR model

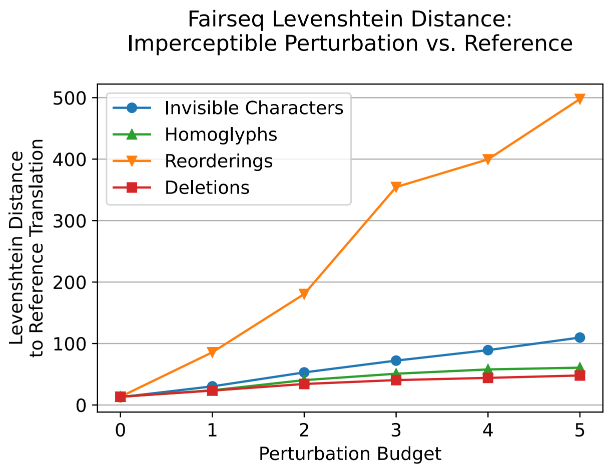


Figure 8: Levenshtein Distances Between Imperceptible Perturbations and Unperturbed WMT Data on Fairseq EN-FR model

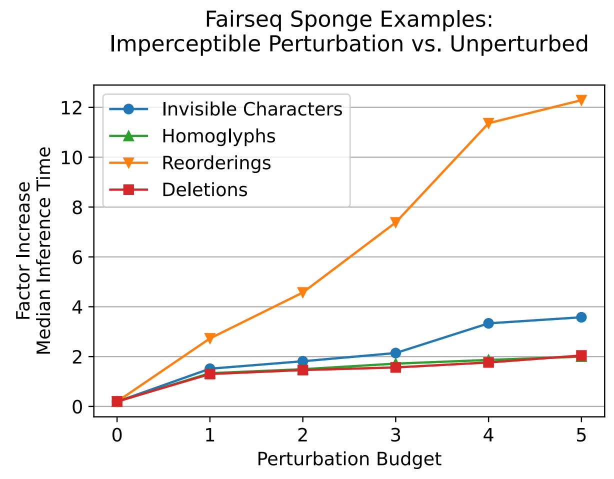


Figure 9: Fairseq sponge example average inference time

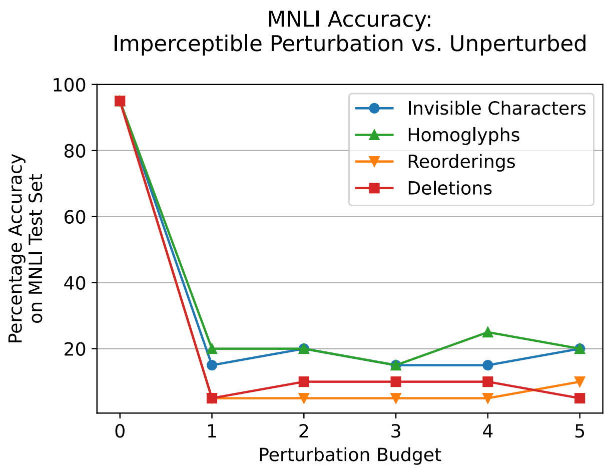


Figure 10: Accuracy of Fairseq MNLI model with imperceptible perturbations

6 EVALUATION

6.1 Attack Performance

We evaluate the performance of each subclass of imperceptible perturbation attacks against Fairseq [29], Facebook AI Research's open source ML toolkit for sequence modeling. We repeat each experiment five times with increasing numbers of perturbations to present results that vary with perturbation budget. [19] The definition of perturbation budget varies with each attack subclass and is defined explicitly in the subsequent sections.

All experiments were performed in a black-box setting in which unlimited model evaluations are permitted, but access into the assessed model's weights or state is not permitted. Our threat model is similar to the interactive black-box setup in [37] and represents one of the strongest threat models. We want to note that the attacks presented are not model-specific and should in theory work against any NLP models that do not have invisible or control characters in its dictionary.

For each subclass of perturbation, we defined an objective function that seeks to maximize the distance between the assessed model's outputs for perturbed and unperturbed inputs. We used Levenshtein distance in these objective functions, but in practice any metric may be chosen. We then performed differential evolution against this objective function. We found that the optimization converged quickly, and thus used a population size of 32 with maximum three iterations in this genetic algorithm. The current parameters already disrupt model operation, and increasing these parameters will help an attacker to find even more effective perturbations.

[19] A link to the source code will be placed here before publication.

Algorithm 1: Imperceptible perturbations via differential evolution

```
Input: text $\mathbf{x}$, set of hidden characters, homoglyphs,
imperceptible characters $\mathbb{H}$, perturbation budget $b$, visual
perturbation budget $v$, NLP task $f$, pool size $p$, evolution
iterations $I$, scale factor s
Randomly initialize population $\mathbf{P} = \{\mathbf{k_0}, \dots, \mathbf{k_p}\}$,
where $\mathbf{k_n} = \{(i_{n0}, c_{n0}), \dots, (i_{nb}, c_{nb})\}$
Function apply takes $\mathbf{x}$ and $\mathbf{k_n}$, applying operation $c_n$ at $i_n$,
where $c_n \in \mathbb{H}$
for $j = 0$ to $I$ do
  for $l = 0$ to $p$ do
    $(\mathbf{k_{r1}}, \mathbf{k_{r2}}, \mathbf{k_{r3}})$ = randomly sampled from $\mathbf{P}$ where
    $l \neq r1 \neq r2 \neq r3$
    $\mathrm{C} = \mathbf{k_{r1}} + s * (\mathbf{k_{r2}} - \mathbf{k_{r3}})$
    $\mathbf{m} = randomofsize(\mathbf{k_l})$
    $\mathrm{C} = \mathbf{m} * \mathbf{k_l} + (1 - \mathbf{m}) * \mathrm{C}$
    if availability attack then
      $Fit$ = time
    else if integrity attack then
      $Fit$ = levenshtein
    end if
    if $Fit(f(apply(\mathbf{x}, \mathrm{C})) > Fit(f(apply(\mathbf{x}, \mathbf{k_l})))$ then
      $\hat{\mathbf{k}}_\mathbf{l} = \mathrm{C}$
    else
      $\hat{\mathbf{k}}_\mathbf{l} = \mathbf{k_l}$
    end if
  end for
  $\mathbf{P} = \{\hat{\mathbf{k}}_\mathbf{l} \mid l = 1, 2, \dots, p\}$
end for
```

We performed two series of experiments across two different classes of text-based machine learning tasks: machine translation and recognizing textual entailment. In the following sections, we describe these experiments.

6.2 Machine Translation Experiments

For the machine-translation task, we used an English-French transformer model pre-trained on WMT14 data [30]. We utilized the corresponding WMT14 test set data to provide reference translations for each adversarial example.

For each experiment, we crafted adversarial examples for 20 sentences of less than 50 characters in length, and repeated adversarial generation for perturbations budgets of 1 through 5. We selected these parameters due to the high computational and energy costs of adversarial example generation with genetic algorithms in the black-box environment.

For the adversarial examples generated, we compare the BLEU scores and Levenshtein distances of the resulting translation against the reference translation in Figure 7 and Figure 8, respectively.

The perturbation budget is defined per subclass of perturbation. For invisible characters, the budget is defined as the maximum number of invisible characters injected into the example. For these experiments, the injected character was either a ZWSP[20] or ZWJ[21] character. For homoglyphs, the perturbation budget is defined as the maximum number of character replacements. Homoglyph to character mappings were taken from the `intentionals` document provided with the Unicode Security Mechanisms technical report [12]. For reorderings, the perturbation budget is defined as the number of *swaps* injected into the document. We define a swap as a series of Bidi override control characters which causes two arbitrary adjacent substrings to render in a flipped order against a target Bidi Algorithm implementation. For these experiments, we used a sequence of ten control characters[22] which we crafted to perform arbitrary swaps against web page rendered text on the Chromium platform, as of the time of writing[23]. For deletions, we define the perturbation budget as the number of pairs of single (deletion control character, to-be-deleted character) pairs to be injected into the example. For these experiments, we selected BKSP[24] and 'a' as the deletion control character, and the target character to be deleted, respectively.

To assess whether the trends suggested by decreasing BLEU score and increasing Levenshtein distance represent decreased translation quality, we had a French-language speaker qualitatively annotate 300 random samples of imperceptible adversarial example translations taken uniformly from all perturbation budgets, attack classes and model providers. The annotations indicated that 77% of resulting translations did not mean the same as the reference translations. Furthermore, increasing the perturbation budget helps change the meaning. For attacks of budget 1, 2, 3 and 4 we get 58%, 54%, 82% and 89% semantically different translations respectively.

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. Sponge example results against the Fairseq English-French model are visualized in Figure 9. Although the slowdown is not as significant as with the sponge examples in [37], our attacks are semantically meaningful, use small budgets and will not be noticeable to human eyes.

6.3 Textual Entailment Experiments

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

For the recognizing textual-entailment task, we performed experiments using the pre-trained RoBERTa model [24] fine-tuned on the MNLI corpus [46]. Like with the machine translation experiments, we used 20 entries from the corpus' corresponding test set to craft adversarial attacks. The objective function strove to maximize misclassification.

The results from this experiment are in Figure 10, where we show correct classification rates as a function of the perturbation budget. Performance drops significantly even with a budget of 1.

[20] Unicode character U+200B
[21] Unicode character U+200D
[22] `U+202D U+2066 U+202E U+2066` substring_one `U+2069 U+2066` substring_two `U+2069 U+202C U+2069 U+202C`
[23] https://www.chromium.org
[24] Unicode character U+0008

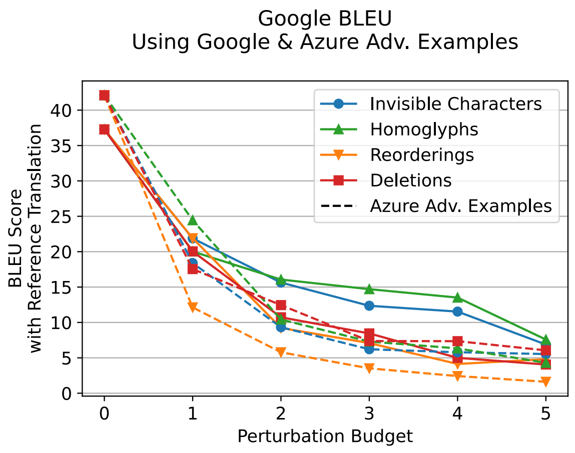


Figure 11: BLEU Scores of Azure's imperceptible adversarial example on Google Translate

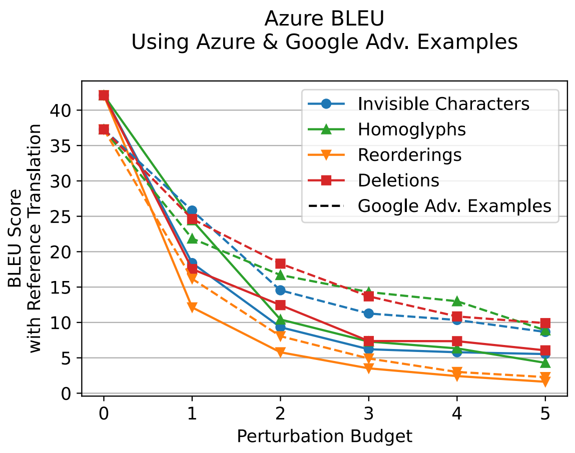


Figure 12: BLEU Scores of Google Translate's imperceptible adversarial example on Microsoft Azure

6.4 MLaaS Case Studies

To show that imperceptible perturbation attacks are viable in the real world, we performed a series of case studies on two popular Machine Learning as a Service (MLaaS) offerings: Google Translate and Microsoft Azure ML. The configurations of these attacks were identical to the experiments against the Fairseq model, except that inference calls were replaced with the appropriate API call. The BLEU results of tests against Google Translate are in Figure 11 and against Microsoft Azure ML in Figure 12. The corresponding Levenshtein results can be found in Appendix A.

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 is plotted as dotted lines in Figure 12 and Figure 11. These results show that imperceptible adversarial examples can be transferred between models.

7 DISCUSSION

7.1 Ethics

We have closely followed departmental guidelines on conducting research that might affect the services provided by third-party companies. In this paper we have interacted with translation services provided by Google and Microsoft. We used legitimate, well-formed API calls, and paid for the translation service. Overall, we used around 500k queries. To minimise the impact both on services and CO_2 production, we chose small inputs, a small number of iterations and a small pool size. For example, while Microsoft Azure allows inputs of size 10000, we used inputs of less than 50 characters [2]. We ran experiments during the night in the region where the server was located. Finally, we followed the standard responsible disclosure process; we shared the paper draft with affected companies.

7.2 Attack Potential

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

- Alter the output of machine translation systems;
- 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 more common forms of encoding. The resulting vulnerabilities become clear enough when viewing 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 to this attack. 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 strictly constrained.

7.3 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.

An attacks can add invisible characters into the search string to reduce the number of found pages. We previously showed that searching for meaning of life with 250 invisible characters returns no results. In practice we find the same attack working with homoglyphs as well – 'meaning of life' with cyrillic letters result in 6000 results, rather than 900 million.

One can also manipulate search performance: searches for 'googleyandex\x08 \x08 \x08 \x08 \x08 \x08'bring back articles about yandex, while rendering only shows google. Reordering can be used to do the same.

7.4 Defences

We now discuss 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 exposed by non-standard encodings used in current NLP systems.

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. But this is so computationally expensive that it cannot give a general-purpose defence. A more realistic defence will be engineered to suit the application.

7.4.1 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 designed such that they are included in the source-language dictionary, resulting in an embedding vector that is not <unk>.

7.4.2 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 2.

Table 2: Text mixing Latin and Cyrillic linguistic families.

Interword Mixing	Intraword Mixing
Hello папа	Hello пaпa

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. It might be more robust 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.

This does not protect against homoglyphs within the same family; to recall the 'paypai' example from 2000, the lowercase 'l', the digit '1' and and uppercase 'I' are homoglyphs in some fonts. Detecting perturbations of this kind is difficult. One might try to define a metric where similarity means same-language homoglyph replacement, and then try to replace ex-dictionary input words with similar in-dictionary words.

7.4.3 Reordering Defences. For some text-based NLP models with a graphical user interface, reordering attacks can be prevented by stripping all Bidi override characters from user input. 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.

7.4.4 Deletion Defences. It is unlikely that many use cases exist where deletion characters represent a valid input into a model. If users of the model enter text via any graphical form field, deletion characters would be processed by the text-rendering engine before typed text is passed to the model.

However, in environments where 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 in its input.

8 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, such as invisible characters and homoglyphs. Although they have been seen occasionally in the past in the context of 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 attempted a systematic exploration of text-processing exploits based on Unicode. 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 systems. Indeed, they can be used on any text-based ML model which processes natural language. For example, 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 this broad family of powerful attacks.

REFERENCES

[1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating Natural Language Adversarial Examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 2890–2896. https://doi.org/10.18653/v1/D18-1316

[2] Microsoft Azure. [n.d.]. Request limits for Translator. https://docs.microsoft.com/en-us/azure/cognitive-services/translator/request-limits

[3] Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and Natural Noise Both Break Neural Machine Translation. *CoRR* abs/1711.02173 (2017). arXiv:1711.02173 http://arxiv.org/abs/1711.02173

[4] Yoshua Bengio, R. Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3 (2003), 1137–1155.

[5] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 387–402.

[6] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (2018), 317–331.

[7] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. 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*. 15–26.

[8] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* abs/1406.1078 (2014). arXiv:1406.1078 http://arxiv.org/abs/1406.1078

[9] Christopher A. Choquette Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. 2020. Label-Only Membership Inference Attacks. arXiv:2007.14321 [cs.CR]

[10] The Unicode Consortium. 2014. *Unicode Security Considerations.* Technical Report Unicode Technical Report #36. The Unicode Consortium. https://www.unicode.org/reports/tr36/tr36-15.html

[11] The Unicode Consortium. 2020. *Unicode Bidirectional Algorithm.* Technical Report Unicode Technical Report #9. The Unicode Consortium. https://www.unicode.org/reports/tr9/tr9-42.html

[12] The Unicode Consortium. 2020. *Unicode Security Considerations.* Technical Report Unicode Technical Report #39. The Unicode Consortium. https://www.unicode.org/reports/tr39/tr39-22.html

[13] The Unicode Consortium. 2020. The Unicode Standard, Version 13.0. https://www.unicode.org/versions/Unicode13.0.0

[14] Bonnie J. Dorr, Pamela W. Jordan, and John W. Benoit. 1998. *A Survey of Current Paradigms in Machine Translation.* Technical Report LAMP-TR-027. MARYLAND UNIV COLLEGE PARK INST FOR ADVANCED COMPUTER STUDIES. https://apps.dtic.mil/docs/citations/ADA455393

[15] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. 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)*. Association for Computational Linguistics, Melbourne, Australia, 31–36. https://doi.org/10.18653/v1/P18-2006

[16] Sheera Frenkel. Sep 14 2020. Facebook Is Failing in Global Disinformation Fight, Says Former Worker. *New York Times* (Sep 14 2020).

[17] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

[18] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. 2019. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 497–514. https://www.usenix.org/conference/usenixsecurity19/presentation/hong

[19] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial Example Generation with Syntactically Controlled Paraphrase Networks. *CoRR* abs/1804.06059 (2018). arXiv:1804.06059 http://arxiv.org/abs/1804.06059

[20] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–35.

[21] Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, 1700–1709. https://www.aclweb.org/anthology/D13-1176

[22] Cedric Knight. 2018. Evasion with Unicode format characters. In *SpamAssassin - Dev.*

[23] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. 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*. 177–180.

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[25] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv:1706.06083 [stat.ML]

[26] Paul Michel, Xian Li, Graham Neubig, and Juan Pino. 2019. 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)*. Association for Computational Linguistics, Minneapolis, Minnesota, 3103–3114. https://doi.org/10.18653/v1/N19-1314

[27] John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. arXiv:2005.05909 [cs.CL]

[28] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles A Sutton, J Doug Tygar, and Kai Xia. 2008. Exploiting Machine Learning to Subvert Your Spam Filter. *LEET* 8 (2008), 1–9.

[29] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

[30] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling Neural Machine Translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*. Association for Computational Linguistics, Brussels, Belgium, 1–9. https://doi.org/10.18653/v1/W18-6301

[31] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 506–519.

[32] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. 2016. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814* (2016).

[33] Nicolas Papernot, Patrick D. McDaniel, Ananthram Swami, and Richard E. Harang. 2016. Crafting Adversarial Input Sequences for Recurrent Neural Networks. *CoRR* abs/1604.08275 (2016). arXiv:1604.08275 http://arxiv.org/abs/1604.08275

[34] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 1085–1097. https://doi.org/10.18653/v1/P19-1103

[35] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural Machine Translation of Rare Words with Subword Units. *CoRR* abs/1508.07909 (2015). arXiv:1508.07909 http://arxiv.org/abs/1508.07909

[36] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural Machine Translation of Rare Words with Subword Units. *CoRR* abs/1508.07909 (2015). arXiv:1508.07909 http://arxiv.org/abs/1508.07909

[37] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. 2020. Sponge Examples: Energy-Latency Attacks on Neural Networks. arXiv:2006.03463 [cs.LG]

[38] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[39] Geoffrey Simpson, Tyler Moore, and Richard Clayton. 2020. Ten years of attacks on companies using visual impersonation of domain names. In *APWG Symposium on Electronic Crime Research (eCrime)*. IEEE.

[40] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. 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., 3104–3112. https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[42] Elham Tabassi, Kevin J Burns, Michael Hadjimichael, Andres D Molina-Markham, and Julian T Sexton. [n.d.]. A Taxonomy and Terminology of Adversarial Machine Learning.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[44] Warren Weaver. 1949. Translation. In *Machine translation of languages: fourteen essays*. Technology Press of the Massachusetts Institute of Technology, Cambridge, MA. https://repositorio.ul.pt/bitstream/10451/10945/2/ulfl155512_tm_2.pdf

[45] John Wieting, Jonathan Mallinson, and Kevin Gimpel. 2017. Learning Paraphrastic Sentence Embeddings from Back-Translated Bitext. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, 274–285. https://doi.org/10.18653/v1/D17-1026

[46] Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. 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)* (New Orleans, Louisiana). Association for Computational Linguistics, 1112–1122. http://aclweb.org/anthology/N18-1101

[47] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating Natural Adversarial Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1BLjgZCb

[48] Wei Zou, Shujian Huang, Jun Xie, Xinyu Dai, and Jiajun Chen. 2020. A Reinforced Generation of Adversarial Examples for Neural Machine Translation. arXiv:1911.03677 [cs.CL]

A CASE STUDY LEVENSHTEIN RESULTS

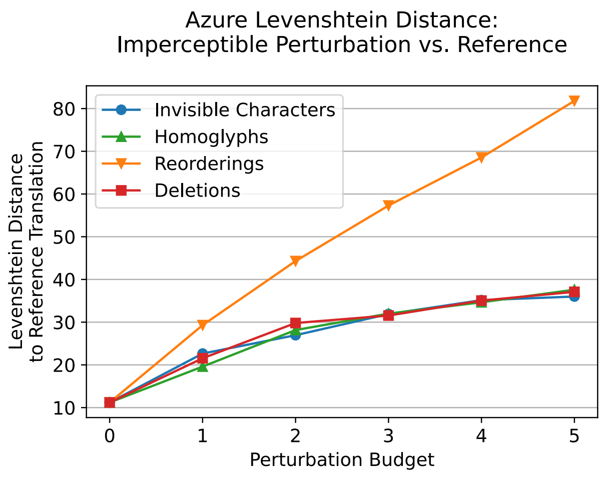


Figure 13: Levenshtein Distances Between Imperceptible Perturbations and Unperturbed WMT Data on Microsoft Azure's EN-FR model

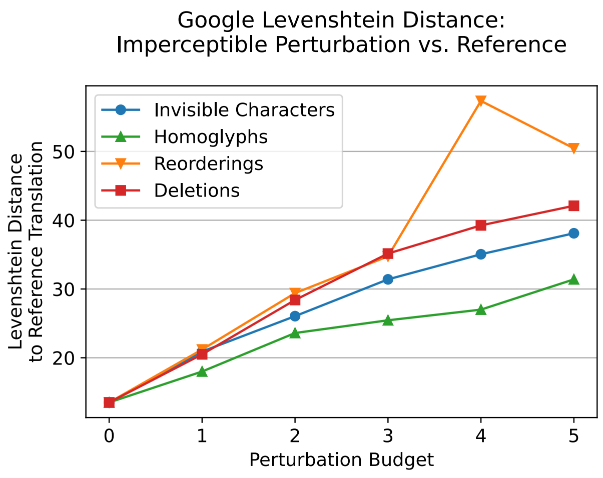


Figure 14: Levenshtein Distances Between Imperceptible Perturbations and Unperturbed WMT Data on Google Translate's EN-FR model

B BIDI INVERSION SEQUENCES

Rendering	Encoded Order	Encoding
1234	1234	1 2 3 4
1234	1243	1 2 U+202B 4 U+2067 3
1234	1324	1 U+2067 3 U+2068 2 U+2029 4
1234	1342	1 U+202B 3 U+2067 4 2
1234	1423	1 U+202B 4 2 U+2067 3
1234	1432	1 U+202B 4 U+2067 3 U+2068 2
1234	2134	U+202B 2 U+2067 1 U+2029 3 4
1234	2143	U+202B 2 U+2066 1 U+2029 U+202B 4 U+2066 3
1234	2314	U+202B 2 U+2067 3 U+2060 1 U+2029 4
1234	2341	U+202B 2 U+2067 3 4 1
1234	2431	U+202B 2 U+2068 4 U+202B 3 U+2028 1
1234	3124	U+202B 3 1 U+2067 2 U+2029 4
1234	3214	U+202B 3 U+2067 2 U+2068 1 U+2029 4
1234	3241	U+202E 3 U+202E 2 4 U+202C U+202C 1
1234	3412	U+202B 3 4 U+2067 1 2
1234	3421	U+202B 3 U+2067 4 U+2068 2 1
1234	4123	U+202B 4 1 2 U+2067 3
1234	4132	U+202E U+202D 4 U+202E 1 3 U+202C U+202C 2 U+202C
1234	4231	U+202B 4 U+2067 2 3 U+2068 1
1234	4213	U+202E U+202D U+202E 4 2 U+202C 1 U+202C 3 U+202C
1234	4312	U+202B 4 3 U+2067 1 U+2068 2
1234	4321	U+202B 4 U+2028 3 U+2028 2 U+2028 1

Table 3: Reordering through Bidi Algorithm manipulation in Google Chrome on MacOS. Although some are correct and behave according to the specification, quite a few are malformed and should not be rendered. Nevertheless, Google Chrome renders them, allowing for adversarial manipulation.