An empirical study of two Bitcoin artifacts through deep learning

Richard Tindell, Alex Mitchell, Nathan Sprague, and Xunhua Wang

James Madison University, Harrisonburg VA 22807, USA
{tindelrj, mitch5aj}@dukes.jmu.edu, {spragunr, wangxx}@jmu.edu

Abstract. Human artifacts like technical papers and computer programs often carry the individual styles of their creators. If retrieved properly, such style information from the artifacts can be used to categorize the artifacts, compare the relative "similarities" among artifacts, and may even be used for tracing the authorship of a new artifact.
Bitcoin is a peer-to-peer cryptocurrency and its author(s) goes/go by the pseudonym of Satoshi Nakamoto. In this article, we use deep learning to study the styles of two Bitcoin artifacts: the first version of Bitcoin's source code, v0.1.0, which was released in early 2009, and the original Bitcoin white paper, which is dated Oct. 2008. Both studies use the deep learning technique, which first utilizes extensive computing power to generate a neural network model from labelled training data and then uses the model to predict the authorship of unknown data. For the Bitcoin source code artifact, the data set is a set of cryptography software that were built around 2008/2009 and it has 16 known labels. Our model achieves 89.1% validation accuracy and our prediction results show that the Bitcoin source code is likely produced by multiple authors and Hal Finney is *not* one of them. For the Bitcoin white paper, we compiled a second data set of financial cryptography papers that are in the same knowledge domain. This data set has 436 known labels. Our model achieves 55.1% validation accuracy and it has identified four technical papers that are "similar" to the Bitcoin white paper.

Keywords: Financial cryptography · Bitcoin · deep learning · anonymity · authorship attribution · code stylometry

1 Introduction

Bitcoin is a peer-to-peer cryptography currency that does *not* require a trusted central bank to create digital money or detect counterfeit & double-spending. Unlike various digital currencies before it, Bitcoin gained wide public acceptance quickly, has sustained several waves of rise and fall, and will likely stay active in the foreseeable future. Bitcoin's design was published as a white paper, under the name of Satoshi Nakamoto, in Oct. 2008 at a web site [31] and Satoshi Nakamoto is obviously a pseudonym. Bitcoin's initial implementation, version *0.1.0*, was released, also under the name of Satoshi Nakamoto, in Jan. 2009 and it includes both a binary executable and some source code files in C++. Satoshi

Nakamoto communicated with the outside world in email and posted to technical forums, but often took caution to use non-identity-revealing pseudonyms, for example with email addresses like *satoshi@vistomail.com* and *satoshin@gmx.com* and online pseudonyms like *satoshi* at bitcointalk.org. (It remains a question whether these "satoshi" are the same person(s); there are claims that Satoshi's email accounts and social media accounts have been hacked.)

Naturally, the true identity of Satoshi has aroused much public interests. In March 2014, Newsweek published an investigative report [19], claiming that Satoshi Nakamoto is Dorian Nakamoto, a Californian who subsequently vehemently denied the claim. So did Satoshi in an anonymous post after the Newsweek publication [32]. On 2 May 2016, the BBC and The Economist published an article [40], in which Craig Wright, an Australian, self-revealed to be Satoshi Nakamoto. As a companion proof [43], Wright provided a digital signature on a message which can be verified by a public key that Satoshi has left in the public block chain. Since only Satoshi has his/her/their private key, only Satoshi is capable of generating a digital signature on a new message. However, later, it has been revealed that the digitally signed message that Wright provided can be extracted from an existing Bitcoin transaction in the public block chain and thus cannot be used to prove the identity of Satoshi [25]. In April 2019, Wright successfully registered US copyright in both the Bitcoin white paper [31] and the code for Bitcoin 0.1, to which the US Copyright Office further clarified that "In a case in which a work is registered under a pseudonym, the Copyright Office does not investigate whether there is a provable connection between the claimant and the pseudonymous author" [41].

This leads to the following questions: Is Wright really Satoshi Nakamoto? Can the author(s) of Bitcoin be traced at all, since there are claims that Satoshi's email accounts were hacked and so were likely Satoshi's social media accounts? Can we find out who Satoshi is purely through the public traces, such as the white paper and the Bitcoin computer programs, left by Satoshi?[1] Can we mine these artifacts to answer the above questions? In addition to the above claims on Satoshi Nakamoto, there have been some other speculations on Bitcoin's authorship. For example, Hal Finney, a cryptography engineer who exchanged public discussions with Satoshi in the early stage of Bitcoin, is considered by many as Satoshi Nakamoto [37]. Is this claim accurate?

Deep learning [26, 9, 21, 1] in recent years has seen big success in multiple applications such as computer vision, speech recognition, auto-piloting, and fraud detection in credit card transactions. Multiple open-source deep learning tools, including scikit-learn, Keras, PyTorch, and TensorFlow, are available. Can these tools be applied to Satoshi's artifacts for authorship tracing?

The Bitcoin white paper is written in English, a natural language. English word-based deep learning has seen wide application and has also been used, in a couple of earlier efforts, for tracing the author(s) of the Bitcoin white paper;

[1] A public web site called the Satoshi Nakamoto Institute [29] has archived the email messages from Satoshi, posts to public forums claimed from Satoshi, and the earlier versions of the Bitcoin software.

more on this later in Section 2.2. The Bitcoin computer program source code, on the other hand, is written in C++ (a formal programming language), has its peculiar characteristics, and needs its own treatment in data mining. There have been multiple studies on source-code-based authorship attribution over a controlled data set [7, 2] (more on this later in Section 2.1 and Section 6) and to our best knowledge, this study is the first reported result on using language-agnostic deep learning on Bitcoin source code with a real-world data set.

In this application-driven research, we explored using existing deep learning techniques for Bitcoin authorship attribution in two ways. Our first attribution is based on the Bitcoin v0.1.0 source code. As a cryptocurrency, the Bitcoin v0.1.0 implementation heavily depends on cryptographic techniques and it uses the OpenSSL library v0.9.8h. It is not unreasonable to assume the Bitcoin developer might be among the developers of the cryptographic libraries. We started by building a source code data set of cryptographic libraries with preselected known authors. To avoid the potential trap of evolving coding style, care was taken to use those cryptographic libraries that were developed at roughly the same time as Bitcoin v0.1.0. For those authors with too few source code samples, mutants were generated so that the data set is balanced for deep learning. We next used this data set to train a neural network. From this trained model authorship predictions for the Bitcoin source code were made. This data set does include code from Hal Finney but not code from Craig Wright, due to its unavailability. Our results show that contrary to one popular belief [37], the relative similarity between Bitcoin code and Finney's code is *not* smaller than other similarities in the data set, showing that Finney is not particularly likely Satoshi. We wish this will settle the Finney argument once and for all. Our results also show that the Bitcoin software 0.1.0 was likely produced by multiple authors, instead of single person.

Our second study followed the Bitcoin technical paper [31], which was first published in Oct. 2008 and then officially in Mar. 2009. This technical paper describes, among other things, the high-level design of Bitcoin, including Bitcoin *transaction*, *block*, *proof-of-work*, and *incentives*. These concepts fall well in the domain of financial cryptography. As a result, the author(s) of the Bitcoin technical paper may well be among the authors of the proceedings of the financial cryptography conferences and related papers. To follow this lead, we compiled a second data set of technical papers, with 436 known labels/author-combinations, from several sources, including most papers in financial cryptography 1997 through financial cryptography 2012, and the technical writings of Hal Finney [12, 13, 14, 15, 17], Wei Dai [11], Adam Back [3], and Craig Wright [43].

We next used this data set to train a neural network model. From this trained model authorship predictions for the Bitcoin technical paper were made. Our results show that the Bitcoin white paper has styles "similar" to four papers.

The remainder of this article is organized as follows. In Section 2, we review priori work related to this research. Section 3 gives a high-level description of our research approach. In Section 4, we present the details of our study on the

Bitcoin source code, including the data collection, balancing, the deep learning model, the results, and their interpretation. Similar details for the Bitcoin white paper are given in Section 5. In Section 6, we further discuss the results and implications of this research. A summary of this research is given in Section 7.

2 Related work

Stylometry aims to find the author(s) of a novel, a poem, a music piece, or a paint, through identifying styles and patterns in them. Previous stylometry examples include the successful identification of the authors for The Federalist Papers [30] and for confirming the collaboration between William Shakespear and Fletcher and Christopher Marlowe [27, 28]. Existing techniques for stylometry include *lexical analysis* to count frequencies of terms and words, more complex *statistics* such as Gaussian statistics, and *neural networks*.

2.1 Code stylometry

Stylometry has also been extended to textual computer programs such as C/C++ source code [35, 22, 7, 2]. It has been observed that just like novelists, painters, and music composers, software developers leave their footprints in source code and this can be used for authorship tracing.

There are two studies with best reported results in this line. Both studies aim for large dataset with thousands of authors and high accuracy. Caliskan-Islam et al. [7] takes a language-dependent approach and it first extracts *layout*, *lexical*, and *abstract syntax tree-based syntactic* features from C/C++ source code. Next, it uses a random forest classifier to de-anonymize C/C++ source code. This research defines $120,000$ layout-based, lexical, and syntactic features but only sends a small subset of features to the random forest classifier. On the Google Code Jam (GCJ) dataset with 1600 programmers, this approach reports 92.83% accuracy. Among the 928 important features identified in [7], 1% are layout (i.e. *shallow*, human-friendly) features, 55% are lexical (i.e. *intermediate-level*) features, and 44% are syntactic (i.e. *deep*, more machine-oriented) features.

Abuhamad et al. [2] takes a language-agnostic approach and uses deep learning based on multiple recurrent natural networks (RNN) layers to extract machine-oriented, statistical features. Next, it sends these features to a random forest classifier for authorship attribution. Like [7], Abuhamad et al. [2] uses the Google Code Jam dataset with 1600 authors, with seven files per author, and reports an accuracy of 96%. Abuhamad et al. [2] also tests their approach on chosen real-world code samples from 1987 public repositories on GitHub, with 745 C programmers and 10 samples per author; this research reports 94.38% accuracy.

Both studies report results on the GCJ controlled data set, which may be very different from real-world data sets; see Section 6 for more details on this. Also, the aforementioned code stylometry techniques work on normal computer code by general programmers, who when writing code typically do not take measures

to hide their identities. This can be considered as a *benign* situation for source code authorship attribution.

However, in some situations, a software developer may deliberately take measures to hide their identities, for example, with pseudonyms, no identity-revealing comments, or no comments at all. Such examples include TrueCrypt and Bitcoin. Even worse, a computer program may be developed to transform source code in a semantics-preserving manner to defeat authorship attribution [38]. Quiring et al. [38] considers an adversary who has a black-box access to the machine learning-based attribution method. In this powerful attack, the adversary does not know the training data or the algorithm of the attribution method but it can send any source code to the attribution method and get both the prediction result and the corresponding prediction score back. Under this attack, Quiring et al. [38] shows that a Monte-Carlo tree-based computer program can be developed to effectively defeat the authorship attribution methods of both [7] and [2], two of the best authorship attribution studies.

It is our belief that the Bitcoin authorship attribution problem does *not* completely fall within this worst-case scenario and is more likely somewhere between the benign case and the worst-case scenario. Both the Bitcoin software v0.1.0 and the technical paper [31] were developed, in 2008/2009, before deep-learning became popular [26].

2.2 Text-based Bitcoin authorship attribution

There has never been lack of interest in tracing the Bitcoin author(s). In addition to the events in Section 1, earlier efforts in tracing the authorship of the Bitcoin white paper include [10, 24, 39, 23, 42, 4, 5, 6].

Chon [10] built a data set of 27 technical papers by 5 known authors and used support vector machine, random forest, and Gaussian Naive Bayes to trace the author(s) of the Bitcoin white paper. Hubbs [24] compiled a data set of writings, including blogs, papers, and published articles, by 7 known authors and used multiple classifiers to trace the author(s) of the Bitcoin white paper. Ramesh and Watson [39] built a data set of write-ups by 7 known authors and used bidirectional LSTM to trace the author(s) of the Bitcoin white paper.

In a different line, Grey [23] studied and compared the human-friendly linguistic features, instead of machine-oriented features, in the Bitcoin white paper and Nick Szabo's writing. Also in this line is a study by Watson [42], which uses a computer program to perform unique word analysis on the Bitcoin white paper and potential candidates' write-ups.

Our work in Section 5 belongs to the camp of machine learning and uses a much bigger data set, with 436 known labels, of more structured and formal texts extracted from peer-reviewed articles.

3 Deep learning for Bitcoin artifacts

Both the Bitcoin software v0.1.0 source code and the Bitcoin technical paper, after text extraction from the PDF file, can be considered as *text*, defined as a

sequence of *characters* or mostly English *words*. Text has been processed and classified by neural network-based deep learning very well with recurrent neural network (RNN), such as the Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), and one-dimensional convolutional neural networks (1D convnets) [9, chap 6]. Deep learning works by first training a neural network model with large amount of known raw data, without much human intervention, and then using the model for prediction for unknown raw data. Deep learning techniques are good at taking raw data/text and automatically producing discriminating statistical features for classification.

In this research, we will apply existing deep learning techniques to both Bitcoin artifacts for their authorship tracing. The texts in those two Bitcoin artifacts are in different subcategories. The source code in the Bitcoin software v0.1.0 was written in C++. Typical C++ source code files contain the code itself, as character strings, and some companion comments, both of which may contain information for authorship tracing. The source code files may also contain additional *lexical and layout features* that could be informational for authorship identification [7]. Example lexical features include the keyword length, comment length, token length, and line length; example layout features include tab length, space length, and empty length.

In contrast, the Bitcoin white paper is a PDF file generated by *PDF-XChange (PDFTools4.exe v4.0.0201.0000)*. Due to the PDF generation process, lexical and layout information in the original source document could have been lost in the PDF file. As a result, the most personal identifiable information in the Bitcoin white paper PDF file probably lies in its content as words.

To study the two Bitcoin artifacts with deep learning, we will need to build a data set for each first. From a data set, a neural network may retrieve multiple types of *discriminators* for classification, including the direct content of the data, such as programming language keywords or English words, and the styles of the authors in the data set. *Not* all such classification discriminators are appropriate for authorship attribution. For example, two technical papers, one about Bitcoin by Alice (called label A) and the other not about Bitcoin by Bob (called label B), can be used to train a neural network. The Bitcoin subject may be chosen by the neural network as a classification discriminator. When a new article of unknown author about Bitcoin is sent to the resulting model for prediction, it may be classified as A but the new article could be written by Bob, hence a wrong authorship attribution. However, when an appropriate data set is used, such as one with items in the same subject category and in sufficient amount, the deep learning-based classifier will discover internal representations that are useful for discriminating between artifacts and these discriminators are more likely about the stylistic similarity of the artifact authors.

In the next two sections, we shall describe how our data sets are chosen, balanced, and processed. Our computer programs to generate neural network models are based on TensorFlow [1] and are written in Python. TensorFlow was chosen over other deep learning libraries in this research for its availability to us and *not* for any particular technical reasons.

4 Source code tracing: tracing the authorship of Bitcoin v0.1.0

In this section, we shall investigate how to use deep learning techniques to trace the author(s) of the Bitcoin software v0.1.0. We start by building a set of source code libraries with known authors that are the potential developers for Bitcoin v0.1.0. We then develop methods to balance this data set and make sure that each known author has enough *samples*. Next, we use this data set to train a deep learning model and in the end use the trained model for prediction.

4.1 Data collection

Bitcoin software v0.1.0 was built on the top of several software libraries, including OpenSSL, Berkeley DB, Boost, and wxWidgets. It bases its cryptographic functions, such as *elliptic-curve key pair generation*, *digital signing*, *signature verification*, and *cryptographic hashing*, on OpenSSL. Bitcoin v0.1.0 also includes explicit instructions to exclude encryption routines from OpenSSL, as Bitcoin does not use encryption. For elliptic-curve digital signature algorithm (ECDSA) [34], it does not use the default parameters in the ECDSA standard. Instead, it uses the Standards for Efficient Cryptography (SEC) parameter *secp256k1* [8]. For cryptographic hashing, it uses both SHA256 [33] and RIPEMD160.

All these point to the fact that the Bitcoin v0.1.0 developers have significant knowledge in cryptography, might have contributed to public cryptographic libraries, or are in the league of these library developers. For Bitcoin v0.1.0 authorship attribution, we collected a set of public cryptographic libraries developed in C/C++ around the time frame of 2008/2009, the time that Bitcoin v0.1.0 was released, as shown in Table 1.

In Table 1, Hal Finney was chosen because his early involvement in email discussions with Satoshi Nakamoto. Until today, Finney has been believed by many to be Satoshi [37]. However, public code by Finney was not common and the only code, as a single file, attributed to him is found at Github as *bc_key*, which is indeed related to Bitcoin.

Some cryptographic libraries such as OpenSSL and Cryptlib are products of multiple authors. Fortunately, the files in these libraries are well marked with author names and thus separated into different data items in Table 1.

Two libraries, *libgcrypt* and *gnupg*, were developed by the same author Werner Koch. Also, the library NSS was ostensibly developed by multiple authors and deserve attention. The author(s) of the TrueCrypt library deliberately masks their/his/her identities. We tried but failed to find the source code of any computer programs written by Craig Wright.

For these chosen libraries, extra steps have been taken to clean them up for duplicate source code files.

4.2 Data balancing with mutants

Figure 6 of the Appendix section gives the total numbers of *.c, .h, .cpp, or .hpp* source files in the *original* libraries of Table 1. These numbers are very

Table 1. A list of existing cryptographic libraries when Bitcoin was first released

Package Name	Author(s)	Chosen version	Release date	Notes
Bitcoin	unknown	0.1.0	Jan. 2009	
bc_key	Hal Finney		Feb. 9, 2011	from Github
CryptoPP	Wei Dai	5.6.0	Mar. 15, 2009	
Cryptlib	Peter Gutmann	3.4.5	Oct. 6, 2010	
	Brian Gladman		≈ Jan. 31, 2006	From cryptlib
OpenSSL	Eric Young	0.9.7m	Feb. 23, 2007	Files are separated in terms of authors
	Stephen Henson			
	Ben Laurie			
	Richard Levitte			
	Geoff Thorpe			
libgcrypt	Werner Koch	1.4.3	≈ Jan. 22, 2009	
libmcrypt	Nikos Mavroyanopoulos	2.5.8	Feb. 19, 2007	
Botan	Jack Lloyd	1.8.0	Dec. 08, 2008	
NSS	Group	3.9.2	≈ Apr. 21, 2008	
TrueCrypt	Anonymous, group	6.1	Oct. 31, 2008	
LUKS	Clemens Fruhwirth	1.1.1	Aug. 12, 2008	
gnupg	Werner Koch	2.0.9	Mar. 26, 2008	

imbalanced. More specifically, *bc_key* by Hal Finney has only one file; within OpenSSL, 4 files were attributed to Ben Laurie, 9 for Richard Levitte, and 9 for Geoff Thorpe. On the other hand, the CryptoPP package has 243 source files (for Wei Dai), Cryptlib has 246 files for Peter Gutmann and 12 files for Brian Gladman, Botan has 586 files (for Jack Lloyd); inside OpenSSL, there are 550 files for Eric Young and 111 files for Stephen Henson.

These files, if not further processed before sending to a machine learning model for training, will inevitably skew the model to be trained toward labels/authors with more files and hence also skew the prediction results. Enforcing a simple threshold (such as 7, as done in [2]) on known authors and dropping those who with smaller files does not work either, as this threshold is likely larger than 1 and thus disqualify labels such as that of *bc_key*, which carries non-trivial weight in Bitcoin source code authorship attribution.

For those known labels/authors with too few samples, one way to overcome the above dilemma is to generate, from the small number of files available, more mutant files that have programming styles very close to the sample files and use the mutant files in model training and validation.

C/C++ .c and .cpp source files may include a section of the *#include* preprocessor directive, a section of the *#define* preprocessor directive for constants and/or macros, some struct definitions, global and static variables, some function declarations, and some function definitions. Not all of these sections appear in a single C/C++ source file. A mutant can be generated by switching the

internal order of multiple #include lines/statements, the internal order of multiple functions, or both. When not enough functions or include statements are available, the internal order of constant/macro definitions can be switched. We believe that mutants generated this way are natural and have a programming style very close to the original files, as they have very close layout-based, lexical, and syntactic features.

Care must be used in dealing with conditional compilation directives in C/C++ source files. Conditional compilation directives may group together multiple #include statements, function definitions, or even a block of statements within a function into different compilation cases (for example, for different hardware platforms). Such conditional compilation directives may appear almost everywhere in a source code file. Switching the internal order of multiple #include statements in the same compilation case may be fine but cross-case switching could be problematic, as the resulting code may not compile or function correctly, is unnatural, and thus should be avoided. In our mutant generation, only the multiple #include statements within the innermost conditional compilations and the multiple function definitions within the innermost conditional compilations are permuted to generate mutants.

Often, given a source file, multiple mutants can be generated through statement permutation and they could be just a subset of all possible mutants. As a whole, each such mutant file is a *different* sequence of characters. However, whenever possible, a mutant should have maximal differences from other chosen mutants from the same source file. In this way, even when a mutant is split into multiple segments in model training and validation, the differences among segments will likely be very different, which helps model training and improves the soundness of model validation results.

It is also worth noting that this mutant generation strategy only works for files that have enough information. It is our estimate that the original files for each label in the data set of Table 1 does have enough information and even the single file bc_key.c, which has 17 functions. After mutation generation, each label/author of Table 1 has at least 100 source files.

4.3 Data preprocessing, modeling, and validation

Content-wise, a C/C++ source file comprises of comments, including copyright notice, and the source code. Both comments and source code are a sequence of characters but they differ in one important way: comments are often in a natural language while source code is in a formal language. Comments and pure code, together with the layout and lexical characteristics of the file, could form three relatively independent inputs to a deep learning model.

However, for the data set of Table 1, not all these three inputs have the same significance in training a deep learning model. Through extensive testing, we observed that comments and lexical/layout features play a very small role compared to the source code in deep learning training and validation. As a result, we adopted a model solely on the pure character-based source code, as shown in Figure 1.

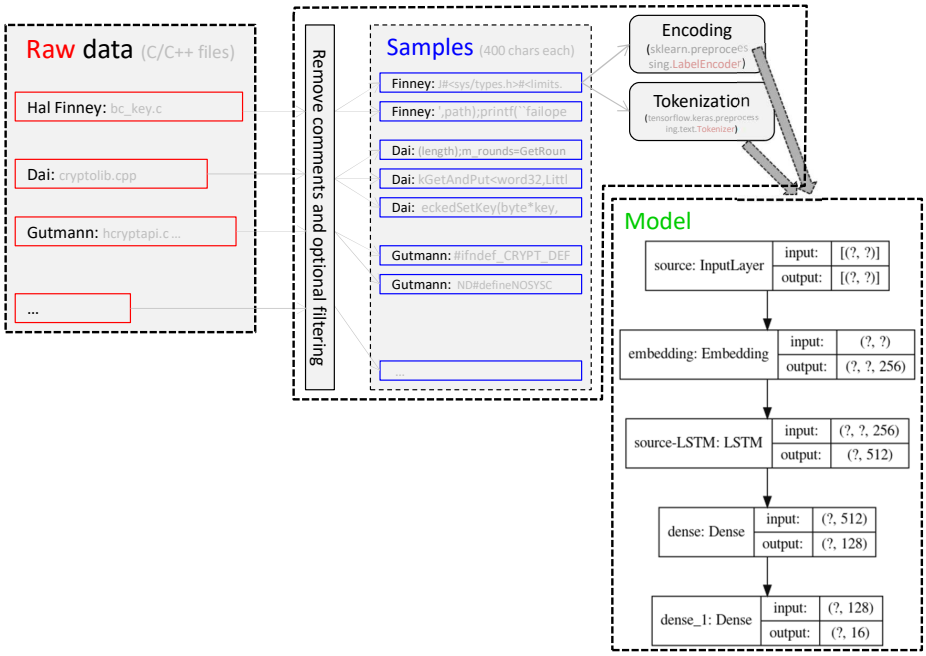

Fig. 1. Deep learning model based on pure source code

In Figure 1, on the left are the raw data in the format of C/C++ files and they are balanced through mutants, as described in Section 4.2, so that each label has at least 100 files. Next, each raw file is preprocessed in a series of steps, first by removing the comment lines, filtering out C/C++ language keywords such as "*break, case, include, public, private, protected, int, long, float,*" and then splitting into smaller *samples* (for example, each with 400 characters). For each label, 1500 *unique* samples are chosen and they are sent to the deep learning model, with 80% of the selected samples are randomly chosen for model training and the rest are used for model validation.

On the right of Figure 1 is the neural network, which includes one Long Short-Term Memory (LSTM) layer with a dimensionality of 512 for the output space and two dense layers, with 128 and 16 as their dimensionality of the output space respectively. (There are 16 known authors in Table 1.) A softmax activation function is applied to the output layer and the network is trained using cross-entropy loss. These hyperparameters are chosen after repeated train-and-validation trials.

From the input samples, we randomly select 80% of the samples for training and the rest for validation. Each training/validation generates a model. Since the network weights are initialized randomly, the final behavior of the models will differ somewhat from one training run to the next. On the same set of samples

we ran the process 81 times. The average training accuracy of these runs reaches 96.9%, with a standard deviation of 0.01; the average validation accuracy is 89.1%, with a standard deviation of 0.018. An early stopping policy, with $1e-2$ min delta and patience of 3, was used. The average epoch number of these runs is 25.67, with a standard deviation of 2.43.

As one example of these 81 runs, the train and validation accuracies and losses are given in Figure 2 and Figure 3 respectively.

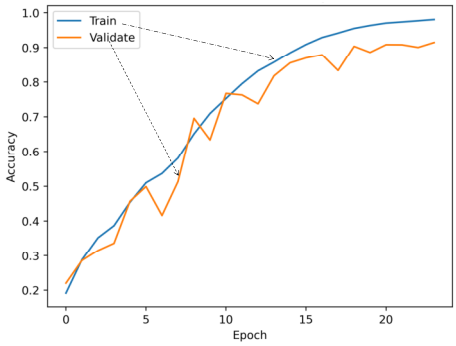

Fig. 2. Example code model accuracy

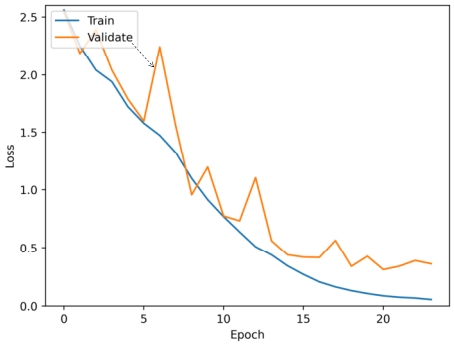

Fig. 3. Example model loss

4.4 Prediction results and interpretation

For each of the 24 source files in Bitcoin software v0.1.0, the 81 models described in Section 4.3 were used to predict its authorship. The prediction probability of each file by each model is above 90%. However, there is variation among the predictions by different models. Table 2 gives the most frequent attribution along with the percentage of models that made that attribution[2].

Care needs to be taken in interpreting the results of Table 2. In this study, we are essentially using the created neural network models to measure the relative "similarities" between the Bitcoin source code v0.1.0 and those 16 libraries in Table 1.

The results in Table 2 show that

1. None of the reported Bitcoin authorship is Hal Finney. In other words, the relative similarities between the source files of Bitcoin v0.1.0 and Hal Finney's code are not smaller than those between Bitcoin v0.1.0 and other libraries in Table 1. Compared to other authors in Table 1, Hal Finney is much less likely to be the developer or one of the developers of Bitcoin

[2] The data set and the corresponding computer programs for this part are available at `https://github.com/wangxx2016/source-code-stylometry`.

Table 2. Bitcoin source code authorship attribution over 81 runs

File Name	Reported Attribution	File Name	Reported Attribution
base58.h	CryptoPP (51%) TrueCrypt (49%)	*net.cpp*	TrueCrypt (52%) CryptoPP (48%)
bignum.h	CryptoPP (40%) TrueCrypt (41%) Botan (28%)	*net.h*	TrueCrypt (67%) CryptoPP (32%)
db.cpp	CryptoPP (70%) TrueCrypt (19%)	*script.cpp*	CryptoPP (54%) TrueCrypt (47%)
db.h	CryptoPP (53%) TrueCrypt (47%)	*script.h*	TrueCrypt (54%) CryptoPP (32%)
headers.h	TrueCrypt (54%) CryptoPP (44%)	*serialize.h*	TrueCrypt (77%)
irc.cpp	TrueCrypt (53%) CryptoPP (43%)	*ui.cpp*	TrueCrypt (36%) Botan (36%) CryptoPP (27%)
irc.h	TrueCrypt (75%)	*ui.h*	Botan (47%) TrueCrypt (25%)
keys.h	Botan (44%) TrueCrypt (40%)	*uibase.cpp*	CryptoPP (43%) TrueCrypt (41%)
main.cpp	CryptoPP (65%) TrueCrypt (35%)	*uibase.h*	CryptoPP (43%) TrueCrypt (41%)
main.h	TrueCrypt (80%) CryptoPP (20%)	*u256int.h*	TrueCrypt (84%)
market.cpp	TrueCrypt (57%) CryptoPP (43%)	*util.cpp*	TrueCrypt (85%)
market.h	TrueCrypt (94%)	*util.h*	TrueCrypt (85%)

source code v0.1.0. This conclusion contradicts a popular belief that Finney is Satoshi.
2. Two source files, *db.cpp* and *main.cpp*, have the largest similarity to CryptoPP.
3. Eight source files, *irc.h*, *main.h*, *market.h*, *net.h*, *serialize.h*, *u256int.h*, *util.cpp*, and *util.h*, have the largest similarity to TrueCrypt.
4. TrueCrypt was developed by anonymous author(s). If TrueCrypt was *not* developed by the author of CryptoPP, Wei Dai, then there are multiple programming styles in Bitcoin v0.1.0 and it is likely that multiple developers have contributed to Bitcoin v0.1.0.

5 Document tracing: tracing the authorship of the Bitcoin white paper

In this section we shall use deep learning to trace the author(s) of another Bitcoin artifact, the Bitcoin white paper [31]. The Bitcoin white paper describes the high-level design of the Bitcoin cryptocurrency. However, there is *no* guarantee that

the Bitcoin white paper author(s) is/are the same person(s) as the author(s) of the Bitcoin source code.

5.1 Data collection

As a peer-to-peer cryptocurrency, Bitcoin is more than just a set of cryptographic techniques. Unlike the numerous cryptocurrencies before it, Bitcoin takes into considerations financial incentives for human beings through its *peer-to-peer* characteristic and the concept of *proof of work*. It more or less falls within the category of financial cryptography. Based on this observation, we chose the data set to include the following technical papers in English: Wei Dai's *b-money* [11], Adam Back's *hashcash* [3], Hal Finney's write-ups [12, 13, 14, 15, 16, 17], and most papers from the proceedings of Financial Cryptography, between 1997 and 2012, organized by International Financial Cryptography Association (IFCA). A semi-technical writeup, Craig Wright's write-up [43], is also included.

In this data set, there are 436 unique known *author combinations*. An author combination can be either a single author or a combination of multiple authors. Two different author combinations may share one or more but not all authors. It is assumed that each author combination has its unique and distinguishable style. There is at least one technical paper, as a digital file, for each author combination. Most files are in the format of PDF and their textual content was extracted through optical character recognition. A further grammar check was conducted on the extracted texts to clean them up before they were sent to a neural network.

5.2 Data modeling

Unlike the *character*-based computer source code of Section 4.3, the raw data of this part is based on English *words*. As shown in Figure 4, the raw English word sequences are partitioned into fixed-size, 100-word *samples*. Each of the 436 labels has 40 samples. If the raw word sequence of a label is not long enough, overlapping samples are generated from the word sequence. This preprocessing step guarantees the balance of the training samples.

To take advantage of the nature of English words, as shown in Figure 4, in the neural network model, Glove, the Global vectors for word presentation [36], is used in the untrainable embedding layer to reduce training time.

In the model of Figure 4, a 1024 dimensional bidirectional LSTM is used, which is followed by a 256-dimensional Dense layer.

From the input samples the neural network randomly picks 80% of the samples for training and the rest for validation. Each training/validation generates a model. This process is kind of probabilistic. On the same set of samples we ran the process 100 times. The average *training* accuracy of these runs reaches 96.6%, with a standard deviation of 0.008; the average *validation* accuracy is 55.1%, with a standard deviation of 0.014. The average epoch number of these runs is 14.66, with a standard deviation of 0.956. It should be noted that the

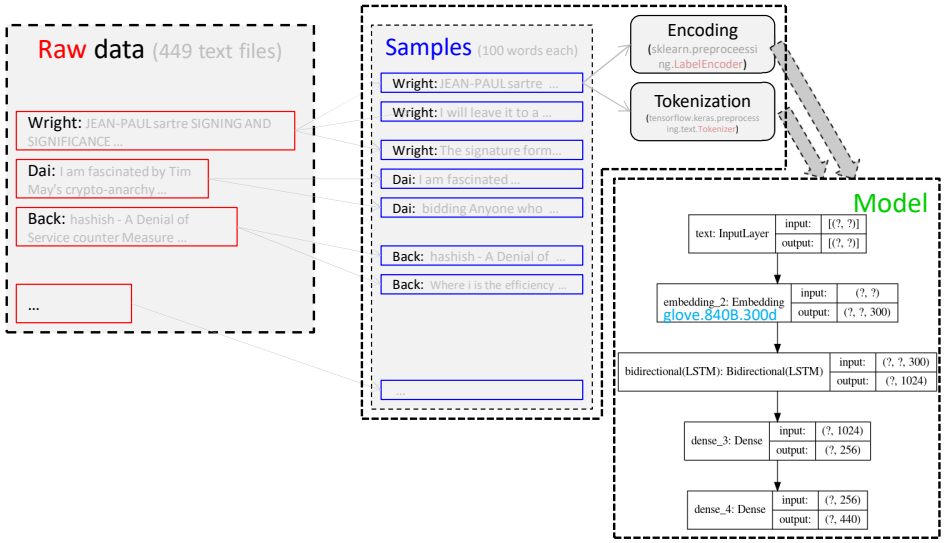

Fig. 4. Deep learning model for text

55.1% validation accuracy is achieved over the 436 labels, compared to 89.1% over 16 labels in Section 4.3.

As one example of these 100 runs, the train and validation accuracies and losses are given in Figure 7 and Figure 8 in the Appendix section respectively.

5.3 Prediction results and interpretation

For the Bitcoin white paper, the 100 models described in Section 5.2 were used to predict its authorship. While the prediction probability by each model is 97.5% on average (with a standard deviation of 0.028), there are variation among the predictions by different models. Figure 5 gives the attribution results, along with the percentage of models that made that attribution[3]. The x-axis is the *predicted labels* (i.e., attribution results), each of which consists of a unique number, the year of financial cryptography proceeding, the ordinal number of the article in the proceeding, and the last names of the author(s), all separated with a hyphen. The y-axis of Figure 5 is the percentage of models that made that prediction.

Figure 5 shows that

1. Among the 436 known labels, the Bitcoin white paper [31] is more similar, in style, to four papers (the leftmost four in Figure 5), each supported by 21%, 14%, 14%, and 11% of the 100 models respectively;
2. Craig Wright's write-up [43] is *not* among the predicted labels;

[3] For this part, the computer programs and a part of the data, with copyrighted materials removed, are available at `https://github.com/wangxx2016/text-stylometry/`.

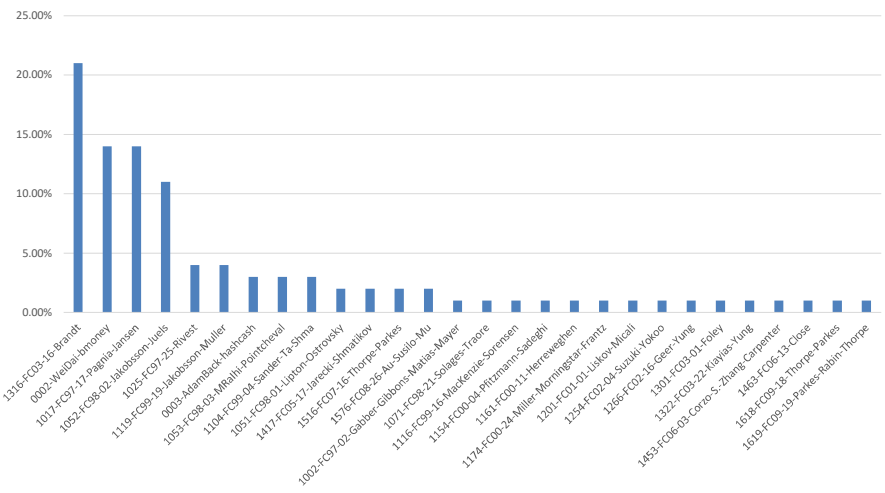

Fig. 5. Bitcoin white paper authorship attribution results

3. Hal Finney's write-ups [12, 13, 14, 15, 16, 17] are *not* among the predicted labels;

6 Discussions

Deep learning is an effective tool for classification and has big potential for both text and source code-based authorship identification. Several points warrant further discussions in using it for real-world authorship identification, especially in tracing the authorship of computer program source code.

It is worth noting that there are multiple differences between a controlled data set and a real-world data set. The source code in a controlled data set such as the Google Code Jam (GCJ) [20] is developed to solve the same set of problems and as a result, the authorship discriminators in it might be more identifiable than in other real-world data.

Second, in a controlled data set, there are a minimal number of files (for example, seven or ten [2]) for each author and this may not be true in many real-world applications. Often, as shown in Section 4.2, steps are needed to generate mutants for balancing real-world training data.

Third, authorship attribution on a controlled data set is a closed-set classification, where the target author is assumed/known to be in a given set of authors. This is *not* necessarily true for applications like Bitcoin, as it is hard to tell whether Satoshi is among any given set of authors. The Bitcoin authorship attribution is an open-set classification problem [18]. As a result, as shown in Section 4.4 and Section 5.3, the deep learning classification results can help us evaluate negative statements such as Finney did not write the Bitcoin software;

in the open-set setting, they cannot help us evaluate positive statements like Wright has developed the Bitcoin software.

Another point in using deep learning classification for authorship attribution is that the attribution results depend heavily on the chosen data set. This is also true in our studies of the Bitcoin source code in Section 4 and the Bitcoin white paper in Section 5. The current selections of the cryptographic libraries in our first data set and the structured technical papers in our second data set are intuitive and can be expanded in future work, for example, to include less structured writings such as blog and discussion forum posts.

7 Summary

Despite having become a household name, the identities of Bitcoin's creator(s) are not known. To trace the author(s) of the Bitcoin source code and Bitcoin white paper, we compiled two data sets, developed computer programs based on deep learning techniques, used the programs to train models on the two data sets, and then used the models to predict the authors of Bitcoin. The first data set has 16 known labels, the model validation accuracy reaches 89.1%, and the prediction results of our models contradict one popular belief that Hal Finney is Satoshi; the prediction results also indicate that there might be multiple contributors to the code. The second data set has 436 labels, the model validation accuracy reaches 55.1%, and the prediction results on the Bitcoin white paper have identified four technical papers that are more similar than others to the Bitcoin white paper.

Our first data set also provides a useful tool for identifying/excluding possible Satoshi and will be shared so that others may find better ways to leverage the data.

8 Acknowledgments

The authors wish to thank the anonymous reviewers for their insightful comments and the shepherd for the pointed guidance. We also thank Jason Brake and Sam Martins for setting up the environment in the early stage of the project. This work is supported in part by the state of Virginia's Commonwealth Cyber Initiative (CCI) through its Northern Virginia node.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*, pages 265–283, 2016.

[2] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang. Large-scale and language-oblivious code authorship identification. In *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, pages 101–114, 2018.

[3] Adam Back. Hashcash - a denial of service counter-measure, 2002. URL `http://www.hashcash.org/papers/hashcash.pdf`.

[4] bit(bit@ungeared.com). The strange story of Satoshi Nakamoto's spelling choices, December 31 2020. URL `https://ungeared.com/the-strange-story-of-satoshi-nakamotos-spelling-choices-part-1/`.

[5] bit(bit@ungeared.com). Satoshi Nakamoto's spelling paradox solved: Everything was by design, January 11 2021. URL `https://ungeared.com/satoshi-nakamotos-spelling-paradox-solved-everything-was-by-design/`.

[6] bit(bit@ungeared.com). Authorship dispute resolution method applied to uncover Satoshi Nakamoto, January 18 2021. URL `https://ungeared.com/authorship-dispute-resolution-method-applied-to-uncover-satoshi-nakamoto/`.

[7] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry. In *Proceedings of the 24th USENIX Security Symposium*, pages 255–270, August 12-14 2015. URL `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/caliskan-islam`.

[8] Certicom Research. Sec 2: Recommended elliptic curve domain parameters version 2.0. Standards for Efficient Cryptography, January 27 2010. URL `http://www.secg.org/sec2-v2.pdf`.

[9] Francois Chollet. *Deep Learning with Python*. Manning, 2017. ISBN 9781617294433.

[10] Michael Chon. Stylometric analysis: Satoshi Nakamoto, Dec 26 2017. URL `https://towardsdatascience.com/stylometric-analysis-satoshi-nakamoto-294926cdf995`.

[11] Wei Dai. b-money, 1998. URL `http://www.weidai.com/bmoney.txt`.

[12] Hal Finney. Digital cash & privacy, August 19 1993. URL `http://fennetic.net/irc/finney.org/~hal/dig_cash_priv.html`.

[13] Hal Finney. Detecting double spending, October 15 1993. URL `https://nakamotoinstitute.org/detecting-double-spending/`.

[14] Hal Finney. PGP web of trust misconceptions, 30 Mar 1994. URL http://fennetic.net/irc/finney.org/~hal/web_of_trust.html.
[15] Hal Finney. RPOW - reusable proofs of work, 15 Aug 2004. URL https://cryptome.org/rpow.htm.
[16] Hal Finney. Dying outside, 4th Oct 2009. URL https://www.lesswrong.com/posts/bshZiaLefDejvPKuS/dying-outside.
[17] Hal Finney. Bitcoin and me, March 19 2013. URL https://bitcointalk.org/index.php?topic=155054.0.
[18] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, Mar 18 2020.
[19] Leah Goodman. The face behind Bitcoin. Newsweek, March 6 2014. URL http://www.newsweek.com/2014/03/14/face-behind-bitcoin-247957.html.
[20] Google. Google code jam, . URL https://codingcompetitions.withgoogle.com/codejam.
[21] Google. TensorFlow: An end-to-end open source machine learning platform, . URL https://www.tensorflow.org/.
[22] Andrew Gray, Philip Sallis, and Stephen MacDonell. IDENTIFIED (integrated dictionary-based extraction of non-languagedependent token information for forensic identification, examination, and discrimination): A dictionary-based system for extracting source code metrics for software forensics. In *Proceedings of the 1998 International Conference Software Engineering: Education and Practice*, 29-29 Jan. 1998.
[23] Skye Grey. Satoshi Nakamoto is (probably) Nick Szabo, Dec 1 2013. URL https://likeinamirror.wordpress.com/2013/12/01/satoshi-nakamoto-is-probably-nick-szabo/.
[24] Christian Hubbs. Can machine learning unmask Satoshi Nakamoto?, Sept 2017. URL https://www.datahubbs.com/can-machine-learning-unmask-satoshi-nakamoto/.
[25] Dan Kaminsky. Validating Satoshi (or not), May 2 2016. URL https://dankaminsky.com/2016/05/02/validating-satoshi-or-not/.
[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, May 28 2015.
[27] Robert A. J. Matthews and Thomas V. N. Merriam. Neural computation in stylometry I: An application to the works of shakespeare and fletcher. *Literary and Linguistic Computing*, 8(4):203–209, 1993.
[28] Thomas V. N. Merriam and Robert A. J. Matthews. Neural computation in stylometry II: An application to the works of shakespeare and marlowe merriam. *Literary and Linguistic Computing*, 9(1):1–6, 1994.
[29] michael@bitstein.org. The complete Satoshi. URL https://satoshi.nakamotoinstitute.org/.
[30] Frederick Mosteller and David L. Wallace. *Applied Bayesian and Classical Inference: The Case of The Federalist Papers*. Springer, 1984.
[31] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. First released to the cryptography mailing list on October

31, 2008 at https://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html; however, the commonly seen PDF file carries a creation timestamp of March 24, 2009., 2008. URL http://bitcoin.org/bitcoin.pdf.

[32] Satoshi Nakamoto. I am not Dorian Nakamoto, March 7 2014. URL http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source?commentId=2003008%3AComment%3A52186.

[33] National Institute of Standards and Technology. Secure hash standard (SHS). FIPS PUB 180-4, March 2012. URL http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[34] National Institute of Standards and Technology. Digital signature standard (DSS). FIPS PUB 186-4, July 2013. URL http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[35] Paul W. Oman and Curtis R. Cook. Typographic style is more than cosmetic. *Communications of the ACM*, 33(5):506–519, May 1990.

[36] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[37] Protos. Finney 'most likely' bitcoin's Nakamoto, say researchers, Jan 18 2021. URL https://protos.com/bitcoin-creator-satoshi-nakamoto-candidates-analysis-hal-finney/.

[38] Erwin Quiring, Alwin Maier, and Konrad Rieck. Misleading authorship attribution of source code using adversarial learning. In *Proceedings of the 28th USENIX Security Symposium*, August 14-16 2019. URL https://www.usenix.org/system/files/sec19-quiring.pdf.

[39] Varun Ramesh and Jean-Luc Watson. Shakespeare and Satoshi - de-anonymizing writing using BiLSTMs with attention, Dec. 31 2018. URL https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6858026.pdf.

[40] The Economist. Craig Wright reveals himself as Satoshi Nakamoto. The Economist, May 2nd 2016. URL https://www.economist.com/briefing/2016/05/02/craig-wright-reveals-himself-as-satoshi-nakamoto.

[41] The U.S. Copyright Office. Questions about certain bitcoin registrations, May 22 2019. URL https://www.copyright.gov/press-media-info/press-updates.html.

[42] Troy Watson. A fascinating discovery uncovers Satoshi Nakamoto's identity, May 26 2021. URL https://zycrypto.com/exclusive-a-fascinating-discovery-uncovers-satoshi-nakamotos-identity/.

[43] Craig Wright. Jean-Paul Sartre, signing and significance, 02 May 2016. URL https://craigwright.net/blog/math/jean-paul-sartre-signing-and-significance/.

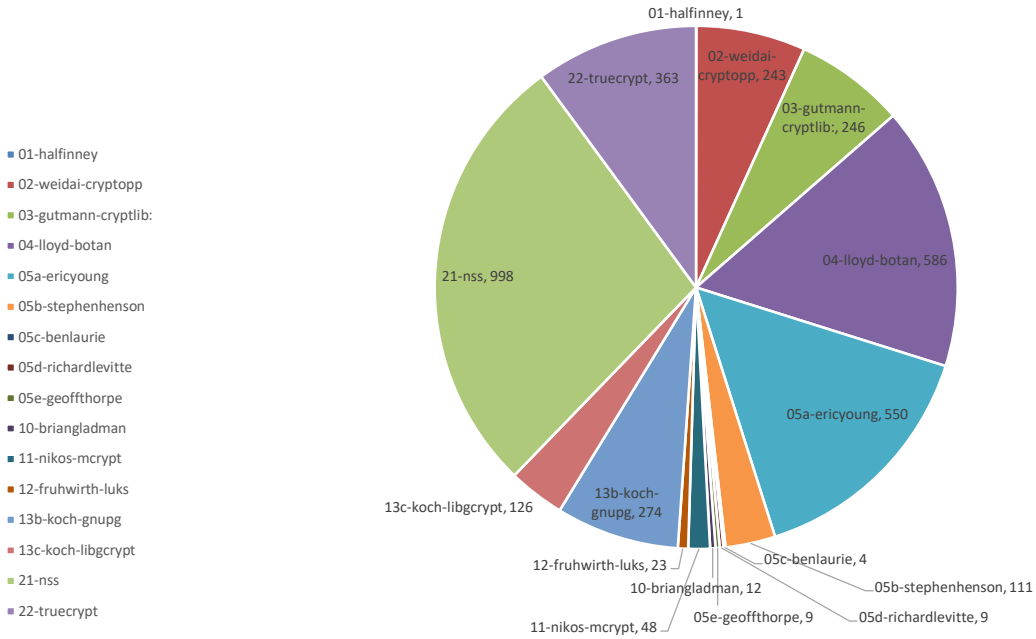

Fig. 6. Data set 1: original file number distribution

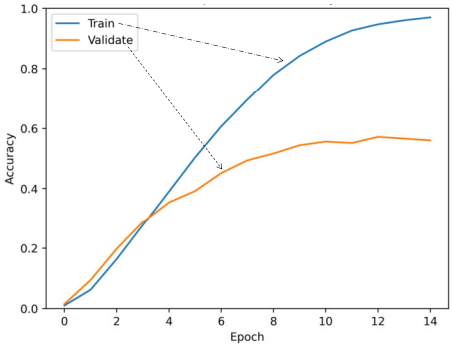

Fig. 7. Example text model accuracy

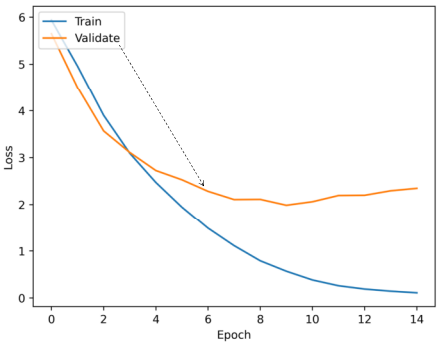

Fig. 8. Example text model loss