CmdCaliper: A Semantic-Aware Command-Line Embedding Model and Dataset for Security Research

Anonymous EMNLP submission

Abstract

This research addresses command-line embedding in cybersecurity, a field obstructed by the lack of comprehensive datasets due to privacy and regulation concerns. We propose the first dataset of similar command lines, named CmdDataset, for training and unbiased evaluation. The training set is generated using a set of large language models (LLMs) comprising 28,520 similar command-line pairs. Our testing dataset consists of 2,807 similar command-line pairs sourced from authentic command-line data.

In addition, we propose a command-line embedding model named CmdCaliper, enabling the computation of semantic similarity with command lines. Performance evaluations demonstrate that the smallest version of CmdCaliper (30 million parameters) suppresses state-of-the-art (SOTA) sentence embedding models with ten times more parameters across various tasks (e.g., malicious command-line detection and similar command-line retrieval).

Our study explores the feasibility of data generation using LLMs in the cybersecurity domain. Furthermore, we release our proposed command-line dataset, embedding models' weights and all program codes to the public. This advancement paves the way for more effective command-line embedding for future researchers.

1 Introduction

Sentence embeddings, which map diverse sentences into a unified semantic feature space, are critical for various NLP applications such as classifier training, visualization (van der Maaten and Hinton, 2008), and retrieval-augmented generation (RAG) (Lewis et al., 2020). In cybersecurity, command lines provide invaluable information for detecting malicious attacks by comparing them with known historical malicious command lines from a semantic perspective. However, the flexibility in command-line syntax and structure poses challenges for fully leveraging this information. For example, as shown in Fig. 1, one can still correlate the two command lines according to their outputs despite the different appearances. To achieve this, using a robust embedding model to calculate the semantic similarity of command lines is promising. However, the grammatical differences between command lines and natural language sentences hinder the direct application of sentence embedding models to command-line tasks. Furthermore, one main challenge exacerbates the difficulty of research in command-line embedding: the scarcity of datasets specifically designed for command-line embedding tasks, both for training models and for fairly evaluating the performance of different methods.

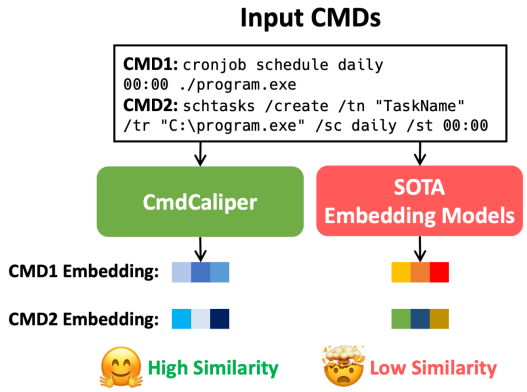

Figure 1: After fine-tuning our proposed similar command-line pair dataset, CmdDataset, our proposed command-line embedding model, CmdCaliper, can effectively embed command lines based on their semantics rather than solely on appearance.

To address the aforementioned challenges, this paper introduces the first comprehensive dataset, CmdDataset, which includes semantically similar pairs of command lines for both training and evaluating command-line embedding methodologies. Inspired by the successes of data synthesis by LLMs (Wang et al., 2023b,a), the similar command-line pairs in our training set are automati-

cally generated from a set of diverse command-line seed initialized from multiple real-world sources by a total of six distinct LLMs trained on diverse datasets (§ 3.1). This facilitates a broader range of command-line generation. For the testing set of CmdDataset, to prevent training data leakage, we directly employed a totally different data source instead of synthesizing command lines by LLMs, as done in the training set. (§ 3.2)

To the end, our training set consists of 28,520 similar command-line pairs, totaling 55,909 unique command lines, and our testing set comprises 2,807 similar command-line pairs, totaling 5,576 unique command lines. Our dataset analysis and human evaluation results (§ 5) demonstrate that our pipeline can generate highly diverse and high-quality similar command-line pairs.

Based on our proposed dataset, CmdDataset, we also developed the first embedding model specialized for command-line embeddings, called CmdCaliper. By encouraging semantically similar samples to come closer and simultaneously increasing the distance between semantically dissimilar samples in the embedding space, CmdCaliper can embed command lines into vectors from a semantic perspective. As demonstrated in Fig. 1, even when command lines differ in appearance, CmdCaliper can still position them closely in the embedding space based on their semantic meanings.

Our evaluation results (§6) demonstrate that even the smallest version of CmdCaliper, with approximately 0.03 billion parameters, can surpass SOTA sentence embedding models with ten times more parameters (0.335 billion parameters) across various command-line specific tasks, such as malicious command-line detection, similar command-line retrieval, and command-line classification.

Our contribution is threefold. First, we propose the first dataset of similar command-line pairs named CmdDataset, which allows for training and performance evaluation. Through detailed validation of the dataset's effectiveness, we believe it is well-suited for further command-line research. Secondly, we explore the potential of using LLMs to synthesize data in the cybersecurity domain. Our experiments demonstrate that LLMs can indeed generate high-quality and diverse data. Lastly, we propose the first semantic command-line embedding model, CmdCaliper. Our evaluations reveal that a command-line-specific embedding model significantly enhances performance across various downstream tasks compared to generic sentence embedding models. We open-source the entire dataset, model weights, and all program codes under BSD License at AnonymousRepo[1]

2 Related Work

2.1 Semantic Embedding

Early works of sentence embedding such as Word2Vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014), require the training of a predefined static embedding lookup table to fuse into the embedding vector of different sentences.

Recent works leverage well-trained language models such as BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020) as pre-trained models to globally embed inputs (i.e., the words of sentences) into embedding vectors while considering contextual relationships to achieve impressive downstream performance. To fine-tune such models, a contrastive learning scheme (van den Oord et al., 2018) is adopted, as demonstrated in (Gao et al., 2021; Chuang et al., 2022; Zeng et al., 2022; Neelakantan et al., 2022; Wang et al., 2022, 2023a)

In the cybersecurity domain, semantic embedding is crucial for computing semantic similarity across various data types, including logs and command lines. For log-based data, Golczynski et al. (Golczynski and Emanuello, 2021) introduced an autoencoder-based model to convert log data into embedding vectors for anomaly detection. Log2Vec (Liu et al., 2019) creates a heterogeneous graph for each log dataset and uses the random walk method with Word2Vec to embed log data. LogBert (Guo et al., 2021) leverages BERT for anomaly detection, clustering the embedding vectors of normal samples to increase the separation from abnormal samples.

For command-line based data, Ongun et al. (Ongun et al., 2021) adapted the tokenization methodology for command-line-based data and followed the Word2Vec approach to train a predefined embedding lookup table for a large amount of command-line data. Conversely, Dong et al. (Dong et al., 2023) directly adopted Word2Vec for the command-line embedding.

2.2 LLMs in the Cybersecurity

In the field of cybersecurity, many researchers (Motlagh et al., 2024; Divakaran and Peddinti, 2024) have also explored leveraging

[1] https://anonymous.4open.science/r/CmdCaliper/README.md

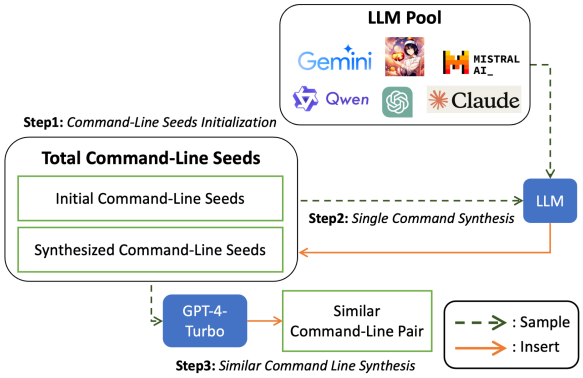

Figure 2: The illustration of the pipeline for automatically generating a dataset of similar command-line pairs using the Self-Instruct algorithm with a pool of LLMs.

LLMs(OpenAI, 2023; Anthropic, 2024; Touvron et al., 2023) for malicious code generations.

Moskal et al. (Moskal et al., 2023) use LLMs to generate executable code for agent actions, facilitating automated cyber campaigns. McKee et al. (McKee and Noever, 2023) explore using LLMs as honeypots by generating executable commands to simulate Linux, Mac, and Windows terminals. Chatzoglou et al. (Chatzoglou et al., 2023) demonstrate ChatGPT's ability to generate malicious code that can evade detection.

3 CmdDataset: The First Command-Line Similarity Dataset

Despite the impressive performance of existing sentence embedding models (Li et al., 2023; Gao et al., 2021; Neelakantan et al., 2022), no embedding model has been designed specifically for command lines. We believe this is due to the unavailability of a large, diversified dataset with adequate annotations for effective training and unbiased evaluation.

In this section, we primarily focus on introducing the first command-line similarity dataset, named CmdDataset. This training set comprises 28,520 pairs of command lines automatically generated by a pool of LLMs. In contrast, the testing set contains 2,807 pairs of command lines collected from real-world attack scenarios.

3.1 Training Set Synthesis by LLMs

Collecting large-scale unlabeled or small-scale annotated command-line data is challenging due to two main factors. Firstly, labeling command-line datasets requires specialized cybersecurity knowledge, making it more stringent and costly than labeling images or natural language sentences. Secondly, privacy concerns involving company or personal information in real-world command lines discourage sharing, complicating efforts to gather diverse, large-scale datasets.

The automatic data generation process known as Self-Instruct (Honovich et al., 2023; Taori et al., 2023; Wang et al., 2023b) has proven effective in acquiring a comprehensive and diverse corpus of instructional data for fine-tuning LLMs. This process utilizes a powerful pre-trained large-scale language model, such as ChatGPT or Claude 3.

Our research is inspired by the substantial success of LLMs in code generation (Rozière et al., 2023; Patil et al., 2023) that exhibits similar structures to command lines and strong comprehension in the cybersecurity domain, such as malware generation (Pa et al., 2023; Botacin, 2023; Charan et al., 2023; Chatzoglou et al., 2023). Based on these capabilities, we adapt the Self-Instruct method to synthesize a substantial number of similar command-line pairs using LLMs. Our data synthesis pipeline comprises three stages: 1) Initial Seeds Collection, 2) Single Command Line Synthesis using a Pool of LLMs, and 3) Similar Command Line Synthesis, as illustrated in Fig. 2.

3.1.1 Initial Seeds Collection

Incorporating randomness into prompts is crucial for enabling LLMs to synthesize diverse command lines. This is achieved using initial seeds, which consist of a diverse set of command lines. During each synthesis iteration, a subset of these seeds is sampled to construct the prompt, diversifying it and encouraging a varied output from LLMs. To ensure high-quality initial seeds, we collected 2,061 diverse Windows command lines from multiple sources (e.g., public red-team exercises and Windows commands documentation). For more details on the initial-seed collection, see Appendix B.

3.1.2 Single Command Line Synthesis with a Pool of LLMs

Beyond the diversity of command-line seeds, the ability of LLMs to generate sufficiently varied data is also crucial. In the original Self-Instruct pipeline (Wang et al., 2023b), GPT-3 (Brown et al., 2020) was adopted for data generation, which confines all generated data to the distribution of GPT-3's training data. We extended this method by constructing a pool of distinct LLMs to aid in data generation. This is intuitive because the distribution of training data varies among different LLMs, leading

to differences in the nature of the command lines they preferentially generate. The LLM pool of our pipeline comprises the following models: Mixtral 8x7B (Jiang et al., 2024), WizardLM-13B-v1.2 (Xu et al., 2024), Gemini-1.0-Pro (Team et al., 2023), Claude-3-Haiku (Anthropic, 2024), Qwen1.5-14B-Chat (Bai et al., 2023), and GPT-3.5-Turbo (OpenAI, 2022).

After constructing the LLM pool, we iteratively synthesize command lines by randomly sampling 12 command lines from the total command-line seeds—which include previously synthesized command lines and initial seeds—for prompt composition, as shown in Fig. 6. We then instruct a randomly selected LLM from the pool to synthesize four new command lines distinct from those in the prompt. Valid command lines are extracted from the LLM responses. Due to the randomness of generation, the LLMs may not always produce four new valid command lines. These valid new command lines are added to the total command-line seeds for the next iteration. The generation process is halted after synthesizing 28,520 command lines.

In this step, LLMs may synthesize non-executable command lines with minor syntax errors. However, for our dataset of similar command-line pairs, 'similar' refers to command lines sharing the same purposes or intentions, typically based on associated executable files, arguments, and argument values. Therefore, even with syntax errors, the command lines should still convey the same purpose or intention as their correct versions.

3.1.3 Similar Command Line Synthesis

After collecting 28,520 command lines, we instructed GPT-4-Turbo (OpenAI, 2023) to generate a similar command line for each. The prompting template for this instruction is displayed in Fig. 7. Here, 'similar' refers to sharing the same purpose or intention, rather than merely having a similar appearance. This distinction is crucial, as in real-world scenarios, attackers may use different command lines or obfuscation techniques to achieve the same goal. We showcase several pairs of generated similar command lines in Table. 9, demonstrating the efficacy of utilizing LLMs for this purpose.

3.2 Real-World Testing Set Collection

To create a testing set that can fairly and comprehensively evaluate different methods, better reflect real-world usage scenarios, and avoid training data leakage, we neither directly partitioned the training set for the testing set, nor did we use LLMs to generate entirely new command lines from the initial seeds as the training set collection pipeline. Instead, we employed Splunk Attack data (Splunk), a dataset curated from various attacks, as the source for our testing set. This allows us to evaluate various approaches in real-world scenarios, as it includes many malicious command lines corresponding to various MITRE ATT&CK (mitre) techniques, covering multiple distinct attack vectors.

Initially, we extracted 12,723 unique command lines from the Splunk Attack data. However, many command lines had similar meanings but differed slightly in appearance, leading to data duplication and potential evaluation inaccuracies. To address this concern, we generated explanations using ChatGPT and then converted these explanations into embeddings using GTE-Large (Li et al., 2023). We used these embeddings to remove command lines with semantically similar content, resulting in a final testing set of 2,807 command lines. For more details about the deduplication process, please refer to Appendix A. We then followed the similar command line synthesis step proposed in Sec. 3.1.3 to instruct GPT-4-turbo to generate the corresponding similar command line for each command line.

For each command line from the original set of 2,807, we used the explanation embeddings to identify the 1,000 least similar command lines from the remaining 2,806 as negative command lines. Additionally, we designated the generated similar command line as the positive command line. This method avoids evaluation inaccuracies by preventing the inclusion of semantically similar command lines among the negatives.

4 CmdCaliper: A Semantic-Aware Command-Line Embedding Model

Utilizing the proposed dataset CmdDataset, a command-line embedding model can be trained with a contrastive objective, like sentence embedding methodologies, as seen in (Gao et al., 2021; Chuang et al., 2022; Neelakantan et al., 2022). Given an embedding model, denoted as E, the procedure of embedding a command line c_i into an embedding vector e_i can be described as $e_i = E(c_i)$. In each training iteration, several similar pairs are randomly sampled from the training set to form a batch, which contains k similar command-line pairs $\{(x_i, x_i^+)\}_{i=1}^k$, where (x_i, x_i^+) represents the i^{th} similar command-line pair. Within the batch,

	Training Set	Testing Set
Num of command-line pair	28,520	2,807
Num of unique command line	55,909	5,576
Max. command-line length	3,464	7,502
Min. command-line length	3	2
Avg. command-line length	91.635	96.301
Std. of command-line length	60.794	196.675

Table 1: The statistic information of CmdDataset.

	Coverage Rate (%)
Windows Commands	73.52
Windows Common File Extensions	70.67

Table 2: Coverage rates of CmdDataset across all Windows Commands and Windows common file extensions.

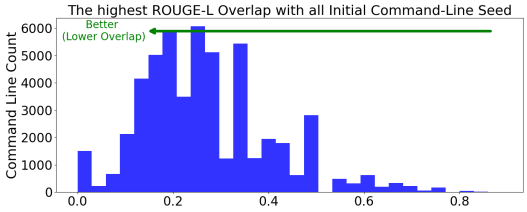

Figure 3: The distribution of the highest ROUGE-L overlap score between the generated command lines and the initial command-line seeds.

the similar command line x_i^+ is regarded as a positive sample of sample x_i, thereby encouraging the associated embedding vectors to be closer within the feature space. Conversely, other samples $\{x_j^+, \forall j \in \{1, 2, \ldots, k\} \setminus \{i\}\}$ are treated as in-batch negatives (Sohn, 2016; Neelakantan et al., 2022; Gao et al., 2021), encouraging the embedding vectors to be farther. The InfoNCE loss (van den Oord et al., 2018), denoted as $\mathcal{L}_{info}$, can be calculated as follows:

$$\mathcal{L}_{info} = -\sum_{i=1}^{k} \log \frac{\exp(\frac{E(x_i) \cdot E(x_i^+)}{\tau})}{\sum_{j=1}^{k} \exp(\frac{E(x_i) \cdot E(x_j^+)}{\tau})}, \quad (1)$$

where τ is a hyperparameter that $\tau \in \mathbb{R}^+$.

5 Evaluation on CmdDataset

5.1 Experimental Settings

In the entire CmdDataset synthesis pipeline, to enhance the diversity of the generated command lines, we follow the approach outlined in (Wang et al., 2023a) and set the temperature parameter to 1 for all LLMs to encourage more random outputs.

5.2 The Statistics of the Dataset

The statistical information for the training and testing sets of the CmdDataset is presented in Table 1. Thanks to the real-world sources of our testing set, the standard deviation of the testing data significantly differs from that of the training set, enabling a more generalized and accurate evaluation.

5.3 The Diversity of the Synthesized Command Line

In this experiment, we aim to assess the diversity of these synthesized command lines. We calculated their coverage across all Windows commands[2] and common file name extensions in a clean Windows 10 virtual machine, as demonstrated in Table 2. To avoid bias, we excluded our manually formulated initial seeds and focused only on the command lines generated by LLMs. Overall, our synthesized command lines achieve a coverage rate of 73.52% out of 306 unique Windows commands and 70.67% out of 75 common file extensions. For more details about the coverage rate calculation process, please refer to Appendix C.

We conducted an in-depth analysis of the differences between the generated command lines and the initial command-line seeds, which served as a foundational starting point for constructing the command-line dataset. For each generated command line, we calculated the highest ROUGE-L overlap (Lin, 2004) which ranges from 0 to 1 among all command-line seeds. A higher ROUGE-L score indicates a greater overlap between the generated command lines and the initial command-line seeds. The distribution of ROUGE-L scores is illustrated in Fig. 3. These findings suggest that the command lines synthesized by our pipeline (Sec. 3.1) are not limited to minor tweaks of the original command-line seeds. On the contrary, they are capable of producing a broad range of command lines, some of which may exhibit significant differences from the initial seeds.

5.4 The Diversity within the Similar Command-Line Pairs

In this section, we examined the distribution of ROUGE-L overlap for each pair of similar command lines, as shown in Fig. 4. These metrics help determine whether similar command-line pairs are derived from minor modifications to arguments or entirely different commands achieving a similar objective. Notably, our findings reveal that most

[2] Windows Command-line reference A-Z

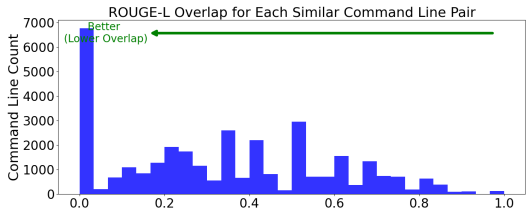

Figure 4: The distribution of ROUGE-L overlap score for all similar command-line pairs.

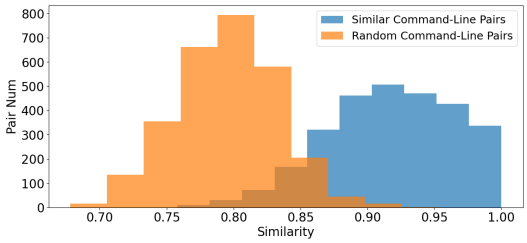

Figure 5: The histogram of the explanation similarity between random command-line pairs and similar command-line pairs in the testing set of CmdDataset.

ROUGE-L scores are low, nearing zero, suggesting that the "similarity" in our command-line dataset is not solely based on lexical similarities but rather reflects genuine semantic similarities. This vital understanding enables us to train command-line embedding models from a semantic perspective and subsequently evaluate the performance of different command-line embedding models.

5.5 The Quality of the LLM's Command-Line Explanations

In Sec. 3.2, we utilize ChatGPT (OpenAI, 2022) to generate explanations for command lines and employ GTE-Large (Li et al., 2023) to convert these explanations into embeddings for data processing. In this evaluation, our focus is on assessing the quality of the explanations generated by LLM.

We randomly selected 200 command lines and their explanations from our training set. An expert (collaborator of this work) with over three years of cybersecurity experience assessed the correctness of each explanation, providing scores as positive, neutral, or negative, while ignoring minor syntax errors. Normalizing the expert's scores against the highest possible total, we obtained a score of **98.25%**, indicating that most command-line explanations accurately describe their purposes. Several good and bad examples are listed in Table 10.

5.6 The Quality of the Similar Command-Line Pairs

In this experiment, we investigate whether the command-line pairs generated by LLMs are truly similar in terms of semantics. Leveraging the findings presented in Sec. 5.5, which demonstrate a high alignment between the explanations generated by LLMs and those provided by human experts, we follow the same experimental settings, utilizing identical prompts and instructing ChatGPT (OpenAI, 2022) to generate explanations for each command line in the testing set.

LLMs for Data Synthesis	Coverage Rate (%)
GPT-3.5-Turbo (OpenAI, 2022)	48.75
Mixtral 8x7B (Jiang et al., 2024)	27.5
WizardLM-13B-v1.2 (Xu et al., 2024)	35
Gemini-1.0-Pro (Team et al., 2023)	51.25
Qwen1.5-14B-Chat (Bai et al., 2023)	23.75
Claude-3-Haiku (Anthropic, 2024)	30
LLM Pool (our)	**70**

Table 3: Explanation clusters coverage rates of the command lines synthesized by different LLMs.

After acquiring all explanations, we employ GTE-Large (Li et al., 2023) to embed all explanations into corresponding embedding vectors and calculate the similarity between explanations for each similar command-line pair. Subsequently, we randomly construct an equal number of command-line pairs, totaling 2,807 pairs, with both command lines in each pair randomly sampled from all command lines in the testing set.

Fig. 5 illustrates the similarity distributions of both similar and random command-line pairs. Notably, there is a significant gap in the explanation similarity between these two groups. Specifically, the average explanation similarity of each similar command-line pair exceeds that of 97.483% of the random command-line pairs, indicating the effectiveness of synthesizing similar command lines with LLMs. Note that the range of similarity for GTE-Large is approximately between 0.65 and 1.

5.7 Effectiveness of a Pool of LLMs

In this experiment, we aim to study whether a pool of LLMs can make the synthesized command lines more diverse versus utilizing a single LLM as we described in Sec. 3.1. First, we instruct each LLM in our pool to synthesize 7,500 command lines from the same initial seeds, totaling 52,500 command lines. Using findings from Sec.5.5, we then generate high-quality explanations for all synthesized

command lines. Next, we embed these explanations into vectors using GTE-Large (Li et al., 2023) and cluster them with DBSCAN (Ester et al., 1996), using a maximum distance of 0.08, a minimum of 5 samples per cluster, and cosine similarity as the distance metric. This process resulted in 80 distinct clusters (excluding the noise cluster), each representing a specific purpose or intention based on similar command line explanations. We then computed the coverage rates of the LLM pool and each individual LLM across these clusters to assess the diversity of synthesized command lines.

The results are presented in Table. 3. It is evident that command lines synthesized by the LLM pool cover the highest number of explanation clusters, reaching up to 70%. This highlights the capability of utilizing a pool of LLMs pre-trained on diverse training data to generate a broader range of command lines.

6 Evaluation on CmdCaliper

6.1 Experimental Settings

CmdCaliper was trained on three distinct model scales: small, base, and large, which are initialized from the GTE-small, -base, and -large (Li et al., 2023), respectively. For more details about the hyperparameters and training processes, please refer to to Appendix E.

6.2 Compare with SOTAs

This section compares several SOTA sentence embedding methods using the testing set from CmdDataset. We adopt Mean Reciprocal Ranking@K (MRR) and Top@K metrics for evaluating the performance of CmdCaliper, following the text search task methodology (Muennighoff et al., 2022). For more details about the two evaluation metrics, please refer to Appendix D. Both metrics yield scores between 0 and 100. The results of this comparative analysis are presented in Table 8.

The results indicate that CmdCaliper consistently achieves competitive performance with state-of-the-art (SOTA) models across all evaluation metrics and scales. Specifically, CmdCaliper-Base achieved an MRR@3 score of 87.56, surpassing our pretrained model - GTE-Base (Li et al., 2023) by 9.36 and outperforming all sentence embedding models of comparable size. On a larger scale, CmdCaliper-Large achieved an MRR@3 score of 89.12, surpassing GTE-Large by 4.86. Remarkably, even CmdCaliper-Small, with a mere 0.03B parameters, is comparable with all SOTA embedding models at the large scale (with 0.335B parameters).

Methods	Params (B)	MRR @3	MRR @10	Top @3	Top @10
Levenshtein distance[1]	–	71.23	72.45	74.99	81.83
Word2Vec[2]	–	45.83	46.93	48.49	54.86
$E5_S$[3]	0.03	81.59	82.6	84.97	90.59
GTE_S[4]	0.03	82.35	83.28	85.39	90.84
CmdCaliper $_S$	0.03	**86.81**	**87.78**	**89.21**	**94.76**
BGE-en$_B$[5]	0.11	79.49	80.41	82.33	87.39
$E5_B$	0.11	83.16	84.07	86.14	91.56
GTR_B[6]	0.11	81.55	82.51	84.54	90.1
GTE_B	0.11	78.2	79.07	81.22	86.14
CmdCaliper $_B$	0.11	**87.56**	**88.47**	**90.27**	**95.26**
BGE-en$_L$	0.34	84.11	84.92	86.64	91.09
$E5_L$	0.34	84.12	85.04	87.32	92.59
GTR_L	0.34	88.09	88.68	91.27	94.58
GTE_L	0.34	84.26	85.03	87.14	91.41
CmdCaliper $_L$	0.34	**89.12**	**89.91**	**91.45**	**95.65**

[1](Haldar and Mukhopadhyay, 2011) [2](Mikolov et al., 2013) [3](Wang et al., 2022) [4](Li et al., 2023) [5](Xiao et al., 2023) [6](Ni et al., 2022)

Table 4: Comparison with the SOTAs for different pre-trained language models. Subscript S, B, and L denote the Small, Base, and Large versions respectively.

Models \ r (%)	20	40	60	80
GTR_{Base}	0.793	0.852	0.866	0.903
$E5_{Base}$	0.796	0.859	0.87	0.899
GTE_{Base}	0.8	0.868	0.874	0.903
$CmdCaliper_{Base}$	**0.869**	**0.906**	**0.927**	**0.939**

Table 5: The AUC comparison for different embedding models and different sample rate $r\%$.

6.3 Semantic-Based Malicious Command-Line Detection

In this section, we approach malicious command-line detection as a retrieval task using the open-source atomic-red-team dataset (Canary), which includes command lines corresponding to 55 different MITRE ATT&CK techniques (mitre) (i.e., each technique describes different command line attack behaviors). We iteratively select $r\%$ of the malicious command lines from each technique as query command lines, while the remaining $1 - r\%$ serve as positive command lines. Command lines from other techniques act as negative command lines. This process is repeated for each technique, and we calculate the average area under curve (AUC). The intuition behind this experiment is that a good embedding model should cluster malicious command lines from the same technique closer together,

Embedding Models	Params (B)	Acc. (%)
$E5_{Base}$	0.11	93.86
GTE_{Base}	0.11	92.8
$CmdCaliper_{Base}$	0.11	**96.37**

Table 6: Accuracy comparison for command-line classification fine-tuned on fixed embedding models.

Model	Params (B)	MRR @3	MRR @10	Top @3	Top @10
Random Initialization	0.11	70.43	72.14	74.49	84.04
$Bert_{Base}$	0.11	82.25	83.38	85.79	92.13
GTE_{Base}	0.11	**87.56**	**88.47**	**90.27**	**95.26**

Table 7: Comparison of the performance of CmdCaliper fine-tuning from the different model.

as they share similar attack behaviors. For more details about the experiment setup, please refer to Appendix F.

The detection results are illustrated in Table. 5. As observed, CmdCaliper-Base significantly outperforms all embedding models not fine-tuned on the command-line dataset. This difference is especially pronounced when the sample ratio is smaller. For instance, at a 20% sample ratio ($r = 20$), CmdCaliper-Base improves upon GTE-Base (Li et al., 2023) by approximately 0.069 in AUC. This suggests that when the query command-line set is smaller, the model requires a deeper understanding of the semantics of command lines.

6.4 Transfer to Command-Line Classification

Fine-tuning an additional module on a pre-trained embedding model for tasks like classification or regression often outperforms training from scratch. This is because well-trained embeddings capture rich, meaningful information that can be used across tasks. In this experiment, we trained a logistic regression classifier for Windows command classification using fixed command-line embeddings from different approaches.

Collecting a labeled command-line dataset poses significant challenges. To address this, we selected seven Windows commands: 'find', 'robocopy', 'msiexec', 'rundll32', 'sc query', 'certutil', and 'print' to synthesize 24,500 training and 24,500 testing command lines. For more details about the the classification dataset synthesis and the experimental setup, please refer to Appendix G.

The results of the classification are presented in Table. 6. As observed, CmdCaliper generally outperformed other sentence embedding models in terms of accuracy for the same model size. For example, CmdCaliper-Base achieved a 3.57% improvement over GTE-Base (Li et al., 2023). These findings highlight the importance of specialized embedding models for command-line data, allowing the models to encode more command-line information into their embedding vectors.

6.5 Does Command-Line Embedding Benefit from Sentence Embedding?

We conducted experiments under three settings: training the Bert-Base (Devlin et al., 2019) network architecture, the pretrained model of GTE-Base (Li et al., 2023), with randomly initialized weights; fine-tuning from Bert-Base; and fine-tuning directly from the GTE-Base model. The results of the comparison are illustrated in Table 7. As observed, the performance of the pretrained model significantly influences the performance of the command-line embedding model. For instance, fine-tuning from the embedding model yields the highest MRR@3, showing a 5.31% improvement compared to direct fine-tuning with BERT.

We believe that the reason command-line embedding models benefit from a good sentence embedding model lies in the fact that, although command lines often have entirely different grammar and structure from general sentences, in many cases, we can still infer some partial meanings of the command lines from semantically meaningful words such as filenames, arguments, or folder names.

7 Conclusion

In this work, we introduce CmdDataset, a dataset of similar command-line pairs. The training set utilizes the impressive capabilities of an LLM pool for automated generation, while the testing set consists of real-world malicious command lines for realistic evaluation. We also present CmdCaliper, the first dedicated command-line embedding model. Our results show that CmdCaliper, specifically designed for command-line processing, outperforms existing sentence embedding methods in various command-line downstream tasks, such as command classification, malicious command line detection, and similar command-line retrieval.

We open-source the dataset, model weights, and all program codes, hoping this study sheds light on future command-line embedding research.

Methods	Hidden Dim	MRR @3	MRR @10	Top @3	Top @10
BGE-en$_{Large}$ [1]	1024	84.11	84.92	86.64	91.09
E5$_{Large}$ [2]	1024	84.12	85.04	87.32	92.59
GTE$_{Large}$ [3]	1024	84.26	85.03	87.14	91.41
OpenAI-text-embedding-3-small [4]	1536	84.35	85.28	87.78	92.8
OpenAI-text-embedding-3-large	3072	89.55	90.19	**92.59**	**96.12**
CyCraft-CmdCraft	1024	**89.76**	**90.32**	**92.59**	95.87

Table 8: Comparison with the SOTAs for different pre-trained language models. Subscript S, B, and L denote the Small, Base, and Large versions respectively.

8 Limitation

Despite the contributions made in this paper, several tasks listed below are worth exploring in the future.

- Support more command-line interpreters: Given their prominence in recent cybersecurity incidents, this study focuses on Windows and PowerShell commands. Compelling statistics (Kutscher, 2023) reveal that over two-thirds of script-based attacks leverage PowerShell. Nevertheless, an investigation into a unified command-line embedding model capable of spanning multiple command-line interpreters presents a promising avenue for future research.

- Resilience against command-line obfuscation: While CmdCaliper was trained on semantically similar command-line pairs, providing a certain degree of resilience against obfuscated command lines in this study, attackers often employ more sophisticated command-line obfuscation techniques to evade defense and detection mechanisms. This poses a significant challenge when relying solely on the embedding model for detecting malicious activities.

- Nested command line: We can obtain the corresponding semantic embedding vector by inputting a command line into CmdCaliper. However, a command line itself can be a composition of multiple command lines as well, making it difficult to accurately embed them into the feature space. Techniques such as few-shot learning (Brown et al., 2020) or instruction-finetuned text embeddings (Su et al., 2023) may provide potential solutions for generating command-line embeddings for specific downstream tasks. This area represents another possibility for future investigation.

References

Anthropic. 2024. Claude 3 haiku: our fastest model yet.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Marcus Botacin. 2023. Gpthreats-3: Is automatic malware generation a threat? In *IEEE Security and Privacy Workshops*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 2020 Conference on Advances in Neural Information Processing Systems*.

Red Canary. Atomic red team.

P. V. Sai Charan, Hrushikesh Chunduri, P. Mohan Anand, and Sandeep K Shukla. 2023. From text to mitre techniques: Exploring the malicious use of large language models for generating cyber attack payloads.

Efstratios Chatzoglou, Georgios Karopoulos, Georgios Kambourakis, and Zisis Tsiatsikas. 2023. Bypassing antivirus detection: old-school malware, new tricks.

Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljacic, Shang-Wen Li, Wen tau Yih, Yoon Kim, and James Glass. 2022. DiffCSE: Difference-based contrastive learning for sentence embeddings. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics*.

DARPA. 2019. Transparent computing engagement 5 data release.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Dinil Mon Divakaran and Sai Teja Peddinti. 2024. Llms for cyber security: New opportunities.

Feng Dong, Liu Wang, Xu Nie, Fei Shao, Haoyu Wang, Ding Li, Xiapu Luo, and Xusheng Xiao. 2023. DISTDET: A Cost-Effective distributed cyber threat detection system. In *Proceedings of the 32nd USENIX Security Symposium*.

Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Andrew Golczynski and John A. Emanuello. 2021. End-to-end anomaly detection for identifying malicious cyber behavior through nlp-based log embeddings. In *Proceedings of the 1st International Workshop on Adaptive Cyber Defense*.

Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *Proceedings of the 2021 International Joint Conference on Neural Networks*.

Rishin Haldar and Debajyoti Mukhopadhyay. 2011. Levenshtein distance technique in dictionary lookup methods: An improved approach.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2023. Unnatural instructions: Tuning language models with (almost) no human labor. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Jurgen Kutscher. 2023. M-Trends 2023: Cybersecurity insights from the frontlines. `https://www.mandiant.com/resources/blog/m-trends-2023`.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv:2308.03281*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *The Proceedings of 2004 Text Summarization Branches Out*.

Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. 2019. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*.

Forrest McKee and David Noever. 2023. Chatbots in a honeypot world.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of the 1st International Conference on Learning Representations, Workshop Track Proceedings*.

mitre. Mitre att&ck.

Stephen Moskal, Sam Laney, Erik Hemberg, and Una-May O'Reilly. 2023. Llms killed the script kiddie: How agents supported by large language models change the landscape of network threat testing.

Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, Pejman Najafi, Feng Cheng, and Christoph Meinel. 2024. Large language models in cybersecurity: State-of-the-art.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. 2022. Text and code embeddings by contrastive pre-training. ArXiv:2201.10005.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernandez Abrego, Ji Ma, Vincent Zhao, Yi Luan, Keith Hall, Ming-Wei Chang, and Yinfei Yang. 2022. Large dual encoders are generalizable retrievers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*.

Talha Ongun, Jack W. Stokes, Jonathan Bar Or, Ke Tian, Farid Tajaddodianfar, Joshua Neil, Christian Seifert, Alina Oprea, and John C. Platt. 2021. Living-off-the-land command detection using active learning. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, page 442–455.

OpenAI. 2022. Introducing chatgpt.

OpenAI. 2023. New models and developer products announced at devday.

Yin Minn Pa Pa, Shunsuke Tanizaki, Tetsui Kou, Michel Van Eetenand Katsunari Yoshioka, and Tsutomu Matsumoto. 2023. An attacker's dream? exploring the capabilities of chatgpt for developing malware. In *Proceedings of the 16th Cyber Security Experimentation and Test Workshop*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code.

Kihyuk Sohn. 2016. Improved deep metric learning with multi-class n-pair loss objective. In *Proceedings of the 2016 Conference on Advances in Neural Information Processing Systems*.

Splunk. Splunk attack data repository.

Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen tau Yih, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2023. One embedder, any task: Instruction-finetuned text embeddings. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Jack Krawczyk, Cosmo Du, Ed Chi, Heng-Tze Cheng, Eric Ni, Purvi Shah, Patrick Kane, Betty Chan, Manaal Faruqui, Aliaksei Severyn, Hanzhao Lin, YaGuang Li, Yong Cheng, Abe Ittycheriah, Mahdis Mahdieh, Mia Chen, Pei Sun, Dustin Tran, Sumit Bagri, Balaji Lakshminarayanan, Jeremiah Liu, Andras Orban, Fabian Güra, Hao Zhou, Xinying Song, Aurelien Boffy, Harish Ganapathy, Steven Zheng, HyunJeong Choe, Ágoston Weisz, Tao Zhu, Yifeng Lu, Siddharth Gopal, Jarrod Kahn, Maciej Kula, Jeff Pitman, Rushin Shah, Emanuel Taropa, Majd Al Merey, Martin Baeuml, Zhifeng Chen, Laurent El Shafey, Yujing Zhang, Olcan Sercinoglu, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, Alexandre Frechette, Charlotte Smith, Laura Culp, Lev Proleev, Yi Luan, Xi Chen, James Lottes, Nathan Schucher, Federico Lebron, Alban Rrustemi, Natalie Clay, Phil Crone, Tomas Kocisky, Jeffrey Zhao, Bartek Perz, Dian Yu, Heidi Howard, Adam Bloniarz, Jack W. Rae, Han Lu, Laurent Sifre, Marcello Maggioni, Fred Alcober, Dan Garrette, Megan Barnes, Shantanu Thakoor, Jacob Austin, Gabriel Barth-Maron, William Wong, Rishabh Joshi, Rahma Chaabouni, Deeni Fatiha, Arun Ahuja, Gaurav Singh Tomar, Evan Senter, Martin Chadwick, Ilya Kornakov, Nithya Attaluri, Iñaki Iturrate, Ruibo Liu, Yunxuan Li, Sarah Cogan, Jeremy Chen, Chao Jia, Chenjie Gu, Qiao Zhang, Jordan Grimstad, Ale Jakse Hartman, Xavier Garcia, Thanumalayan Sankaranarayana Pillai, Jacob Devlin, Michael Laskin, Diego

de Las Casas, Dasha Valter, Connie Tao, Lorenzo Blanco, Adrià Puigdomènech Badia, David Reitter, Mianna Chen, Jenny Brennan, Clara Rivera, Sergey Brin, Shariq Iqbal, Gabriela Surita, Jane Labanowski, Abhi Rao, Stephanie Winkler, Emilio Parisotto, Yiming Gu, Kate Olszewska, Ravi Addanki, Antoine Miech, Annie Louis, Denis Teplyashin, Geoff Brown, Elliot Catt, Jan Balaguer, Jackie Xiang, Pidong Wang, Zoe Ashwood, Anton Briukhov, Albert Webson, Sanjay Ganapathy, Smit Sanghavi, Ajay Kannan, Ming-Wei Chang, Axel Stjerngren, Josip Djolonga, Yuting Sun, Ankur Bapna, Matthew Aitchison, Pedram Pejman, Henryk Michalewski, Tianhe Yu, Cindy Wang, Juliette Love, Junwhan Ahn, Dawn Bloxwich, Kehang Han, Peter Humphreys, Thibault Sellam, James Bradbury, Varun Godbole, Sina Samangooei, Bogdan Damoc, Alex Kaskasoli, Sébastien M. R. Arnold, Vijay Vasudevan, Shubham Agrawal, Jason Riesa, Dmitry Lepikhin, Richard Tanburn, Srivatsan Srinivasan, Hyeontaek Lim, Sarah Hodkinson, Pranav Shyam, Johan Ferret, Steven Hand, Ankush Garg, Tom Le Paine, Jian Li, Yujia Li, Minh Giang, Alexander Neitz, Zaheer Abbas, Sarah York, Machel Reid, Elizabeth Cole, Aakanksha Chowdhery, Dipanjan Das, Dominika Rogozińska, Vitaliy Nikolaev, Pablo Sprechmann, Zachary Nado, Lukas Zilka, Flavien Prost, Luheng He, Marianne Monteiro, Gaurav Mishra, Chris Welty, Josh Newlan, Dawei Jia, Miltiadis Allamanis, Clara Huiyi Hu, Raoul de Liedekerke, Justin Gilmer, Carl Saroufim, Shruti Rijhwani, Shaobo Hou, Disha Shrivastava, Anirudh Baddepudi, Alex Goldin, Adnan Ozturel, Albin Cassirer, Yunhan Xu, Daniel Sohn, Devendra Sachan, Reinald Kim Amplayo, Craig Swanson, Dessie Petrova, Shashi Narayan, Arthur Guez, Siddhartha Brahma, Jessica Landon, Miteyan Patel, Ruizhe Zhao, Kevin Villela, Luyu Wang, Wenhao Jia, Matthew Rahtz, Mai Giménez, Legg Yeung, James Keeling, Petko Georgiev, Diana Mincu, Boxi Wu, Salem Haykal, Rachel Saputro, Kiran Vodrahalli, James Qin, Zeynep Cankara, Abhanshu Sharma, Nick Fernando, Will Hawkins, Behnam Neyshabur, Solomon Kim, Adrian Hutter, Priyanka Agrawal, Alex Castro-Ros, George van den Driessche, Tao Wang, Fan Yang, Shuo yiin Chang, Paul Komarek, Ross McIlroy, Mario Lučić, Guodong Zhang, Wael Farhan, Michael Sharman, Paul Natsev, Paul Michel, Yamini Bansal, Siyuan Qiao, Kris Cao, Siamak Shakeri, Christina Butterfield, Justin Chung, Paul Kishan Rubenstein, Shivani Agrawal, Arthur Mensch, Kedar Soparkar, Karel Lenc, Timothy Chung, Aedan Pope, Loren Maggiore, Jackie Kay, Priya Jhakra, Shibo Wang, Joshua Maynez, Mary Phuong, Taylor Tobin, Andrea Tacchetti, Maja Trebacz, Kevin Robinson, Yash Katariya, Sebastian Riedel, Paige Bailey, Kefan Xiao, Nimesh Ghelani, Lora Aroyo, Ambrose Slone, Neil Houlsby, Xuehan Xiong, Zhen Yang, Elena Gribovskaya, Jonas Adler, Mateo Wirth, Lisa Lee, Music Li, Thais Kagohara, Jay Pavagadhi, Sophie Bridgers, Anna Bortsova, Sanjay Ghemawat, Zafarali Ahmed, Tianqi Liu, Richard Powell, Vijay Bolina, Mariko Iinuma, Polina Zablotskaia, James Besley, Da-Woon Chung, Timothy Dozat, Ramona Comanescu, Xiance Si, Jeremy Greer, Guolong Su, Martin Polacek, Raphaël Lopez Kaufman, Simon Tokumine, Hexiang Hu, Elena Buchatskaya, Yingjie Miao, Mohamed Elhawaty, Aditya Siddhant, Nenad Tomasev, Jinwei Xing, Christina Greer, Helen Miller, Shereen Ashraf, Aurko Roy, Zizhao Zhang, Ada Ma, Angelos Filos, Milos Besta, Rory Blevins, Ted Klimenko, Chih-Kuan Yeh, Soravit Changpinyo, Jiaqi Mu, Oscar Chang, Mantas Pajarskas, Carrie Muir, Vered Cohen, Charline Le Lan, Krishna Haridasan, Amit Marathe, Steven Hansen, Sholto Douglas, Rajkumar Samuel, Mingqiu Wang, Sophia Austin, Chang Lan, Jiepu Jiang, Justin Chiu, Jaime Alonso Lorenzo, Lars Lowe Sjösund, Sébastien Cevey, Zach Gleicher, Thi Avrahami, Anudhyan Boral, Hansa Srinivasan, Vittorio Selo, Rhys May, Konstantinos Aisopos, Léonard Hussenot, Livio Baldini Soares, Kate Baumli, Michael B. Chang, Adrià Recasens, Ben Caine, Alexander Pritzel, Filip Pavetic, Fabio Pardo, Anita Gergely, Justin Frye, Vinay Ramasesh, Dan Horgan, Kartikeya Badola, Nora Kassner, Subhrajit Roy, Ethan Dyer, Víctor Campos Campos, Alex Tomala, Yunhao Tang, Dalia El Badawy, Elspeth White, Basil Mustafa, Oran Lang, Abhishek Jindal, Sharad Vikram, Zhitao Gong, Sergi Caelles, Ross Hemsley, Gregory Thornton, Fangxiaoyu Feng, Wojciech Stokowiec, Ce Zheng, Phoebe Thacker, Çağlar Ünlü, Zhishuai Zhang, Mohammad Saleh, James Svensson, Max Bileschi, Piyush Patil, Ankesh Anand, Roman Ring, Katerina Tsihlas, Arpi Vezer, Marco Selvi, Toby Shevlane, Mikel Rodriguez, Tom Kwiatkowski, Samira Daruki, Keran Rong, Allan Dafoe, Nicholas FitzGerald, Keren Gu-Lemberg, Mina Khan, Lisa Anne Hendricks, Marie Pellat, Vladimir Feinberg, James Cobon-Kerr, Tara Sainath, Maribeth Rauh, Sayed Hadi Hashemi, Richard Ives, Yana Hasson, Eric Noland, Yuan Cao, Nathan Byrd, Le Hou, Qingze Wang, Thibault Sottiaux, Michela Paganini, Jean-Baptiste Lespiau, Alexandre Moufarek, Samer Hassan, Kaushik Shivakumar, Joost van Amersfoort, Amol Mandhane, Pratik Joshi, Anirudh Goyal, Matthew Tung, Andrew Brock, Hannah Sheahan, Vedant Misra, Cheng Li, Nemanja Rakićević, Mostafa Dehghani, Fangyu Liu, Sid Mittal, Junhyuk Oh, Seb Noury, Eren Sezener, Fantine Huot, Matthew Lamm, Nicola De Cao, Charlie Chen, Sidharth Mudgal, Romina Stella, Kevin Brooks, Gautam Vasudevan, Chenxi Liu, Mainak Chain, Nivedita Melinkeri, Aaron Cohen, Venus Wang, Kristie Seymore, Sergey Zubkov, Rahul Goel, Summer Yue, Sai Krishnakumaran, Brian Albert, Nate Hurley, Motoki Sano, Anhad Mohananey, Jonah Joughin, Egor Filonov, Tomasz Kępa, Yomna Eldawy, Jiawern Lim, Rahul Rishi, Shirin Badiezadegan, Taylor Bos, Jerry Chang, Sanil Jain, Sri Gayatri Sundara Padmanabhan, Subha Puttagunta, Kalpesh Krishna, Leslie Baker, Norbert Kalb, Vamsi Bedapudi, Adam Kurzrok, Shuntong Lei, Anthony Yu, Oren Litvin, Xiang Zhou, Zhichun Wu, Sam Sobell, Andrea Siciliano, Alan Papir, Robby Neale, Jonas Bragagnolo, Tej Toor, Tina Chen, Valentin Anklin, Feiran Wang, Richie Feng, Milad Gholami, Kevin Ling, Lijuan Liu, Jules Walter, Hamid Moghaddam, Arun Kishore,

Jakub Adamek, Tyler Mercado, Jonathan Mallinson, Siddhinita Wandekar, Stephen Cagle, Eran Ofek, Guillermo Garrido, Clemens Lombriser, Maksim Mukha, Botu Sun, Hafeezul Rahman Mohammad, Josip Matak, Yadi Qian, Vikas Peswani, Pawel Janus, Quan Yuan, Leif Schelin, Oana David, Ankur Garg, Yifan He, Oleksii Duzhyi, Anton Älgmyr, Timothée Lottaz, Qi Li, Vikas Yadav, Luyao Xu, Alex Chinien, Rakesh Shivanna, Aleksandr Chuklin, Josie Li, Carrie Spadine, Travis Wolfe, Kareem Mohamed, Subhabrata Das, Zihang Dai, Kyle He, Daniel von Dincklage, Shyam Upadhyay, Akanksha Maurya, Luyan Chi, Sebastian Krause, Khalid Salama, Pam G Rabinovitch, Pavan Kumar Reddy M, Aarush Selvan, Mikhail Dektiarev, Golnaz Ghiasi, Erdem Guven, Himanshu Gupta, Boyi Liu, Deepak Sharma, Idan Heimlich Shtacher, Shachi Paul, Oscar Akerlund, François-Xavier Aubet, Terry Huang, Chen Zhu, Eric Zhu, Elico Teixeira, Matthew Fritze, Francesco Bertolini, Liana-Eleonora Marinescu, Martin Bölle, Dominik Paulus, Khyatti Gupta, Tejasi Latkar, Max Chang, Jason Sanders, Roopa Wilson, Xuewei Wu, Yi-Xuan Tan, Lam Nguyen Thiet, Tulsee Doshi, Sid Lall, Swaroop Mishra, Wanming Chen, Thang Luong, Seth Benjamin, Jasmine Lee, Ewa Andrejczuk, Dominik Rabiej, Vipul Ranjan, Krzysztof Styrc, Pengcheng Yin, Jon Simon, Malcolm Rose Harriott, Mudit Bansal, Alexei Robsky, Geoff Bacon, David Greene, Daniil Mirylenka, Chen Zhou, Obaid Sarvana, Abhimanyu Goyal, Samuel Andermatt, Patrick Siegler, Ben Horn, Assaf Israel, Francesco Pongetti, Chih-Wei "Louis" Chen, Marco Selvatici, Pedro Silva, Kathie Wang, Jackson Tolins, Kelvin Guu, Roey Yogev, Xiaochen Cai, Alessandro Agostini, Maulik Shah, Hung Nguyen, Noah Ó Donnaile, Sébastien Pereira, Linda Friso, Adam Stambler, Adam Kurzrok, Chenkai Kuang, Yan Romanikhin, Mark Geller, ZJ Yan, Kane Jang, Cheng-Chun Lee, Wojciech Fica, Eric Malmi, Qijun Tan, Dan Banica, Daniel Balle, Ryan Pham, Yanping Huang, Diana Avram, Hongzhi Shi, Jasjot Singh, Chris Hidey, Niharika Ahuja, Pranab Saxena, Dan Dooley, Srividya Pranavi Potharaju, Eileen O'Neill, Anand Gokulchandran, Ryan Foley, Kai Zhao, Mike Dusenberry, Yuan Liu, Pulkit Mehta, Ragha Kotikalapudi, Chalence Safranek-Shrader, Andrew Goodman, Joshua Kessinger, Eran Globen, Prateek Kolhar, Chris Gorgolewski, Ali Ibrahim, Yang Song, Ali Eichenbaum, Thomas Brovelli, Sahitya Potluri, Preethi Lahoti, Cip Baetu, Ali Ghorbani, Charles Chen, Andy Crawford, Shalini Pal, Mukund Sridhar, Petru Gurita, Asier Mujika, Igor Petrovski, Pierre-Louis Cedoz, Chenmei Li, Shiyuan Chen, Niccolò Dal Santo, Siddharth Goyal, Jitesh Punjabi, Karthik Kappaganthu, Chester Kwak, Pallavi LV, Sarmishta Velury, Himadri Choudhury, Jamie Hall, Premal Shah, Ricardo Figueira, Matt Thomas, Minjie Lu, Ting Zhou, Chintu Kumar, Thomas Jurdi, Sharat Chikkerur, Yenai Ma, Adams Yu, Soo Kwak, Victor Ähdel, Sujeevan Rajayogam, Travis Choma, Fei Liu, Aditya Barua, Colin Ji, Ji Ho Park, Vincent Hellendoorn, Alex Bailey, Taylan Bilal, Huanjie Zhou, Mehrdad Khatir, Charles Sutton, Wojciech Rzadkowski, Fiona Macintosh, Konstantin Shagin, Paul Medina, Chen Liang, Jinjing Zhou, Pararth Shah, Yingying Bi, Attila Dankovics, Shipra Banga, Sabine Lehmann, Marissa Bredesen, Zifan Lin, John Eric Hoffmann, Jonathan Lai, Raynald Chung, Kai Yang, Nihal Balani, Arthur Bražinskas, Andrei Sozanschi, Matthew Hayes, Héctor Fernández Alcalde, Peter Makarov, Will Chen, Antonio Stella, Liselotte Snijders, Michael Mandl, Ante Kärrman, Paweł Nowak, Xinyi Wu, Alex Dyck, Krishnan Vaidyanathan, Raghavender R, Jessica Mallet, Mitch Rudominer, Eric Johnston, Sushil Mittal, Akhil Udathu, Janara Christensen, Vishal Verma, Zach Irving, Andreas Santucci, Gamaleldin Elsayed, Elnaz Davoodi, Marin Georgiev, Ian Tenney, Nan Hua, Geoffrey Cideron, Edouard Leurent, Mahmoud Alnahlawi, Ionut Georgescu, Nan Wei, Ivy Zheng, Dylan Scandinaro, Heinrich Jiang, Jasper Snoek, Mukund Sundararajan, Xuezhi Wang, Zack Ontiveros, Itay Karo, Jeremy Cole, Vinu Rajashekhar, Lara Tumeh, Eyal Ben-David, Rishub Jain, Jonathan Uesato, Romina Datta, Oskar Bunyan, Shimu Wu, John Zhang, Piotr Stanczyk, Ye Zhang, David Steiner, Subhajit Naskar, Michael Azzam, Matthew Johnson, Adam Paszke, Chung-Cheng Chiu, Jaume Sanchez Elias, Afroz Mohiuddin, Faizan Muhammad, Jin Miao, Andrew Lee, Nino Vieillard, Jane Park, Jiageng Zhang, Jeff Stanway, Drew Garmon, Abhijit Karmarkar, Zhe Dong, Jong Lee, Aviral Kumar, Luowei Zhou, Jonathan Evens, William Isaac, Geoffrey Irving, Edward Loper, Michael Fink, Isha Arkatkar, Nanxin Chen, Izhak Shafran, Ivan Petrychenko, Zhe Chen, Johnson Jia, Anselm Levskaya, Zhenkai Zhu, Peter Grabowski, Yu Mao, Alberto Magni, Kaisheng Yao, Javier Snaider, Norman Casagrande, Evan Palmer, Paul Suganthan, Alfonso Castaño, Irene Giannoumis, Wooyeol Kim, Mikołaj Rybiński, Ashwin Sreevatsa, Jennifer Prendki, David Soergel, Adrian Goedeckemeyer, Willi Gierke, Mohsen Jafari, Meenu Gaba, Jeremy Wiesner, Diana Gage Wright, Yawen Wei, Harsha Vashisht, Yana Kulizhskaya, Jay Hoover, Maigo Le, Lu Li, Chimezie Iwuanyanwu, Lu Liu, Kevin Ramirez, Andrey Khorlin, Albert Cui, Tian LIN, Marcus Wu, Ricardo Aguilar, Keith Pallo, Abhishek Chakladar, Ginger Perng, Elena Allica Abellan, Mingyang Zhang, Ishita Dasgupta, Nate Kushman, Ivo Penchev, Alena Repina, Xihui Wu, Tom van der Weide, Priya Ponnapalli, Caroline Kaplan, Jiri Simsa, Shuangfeng Li, Olivier Dousse, Fan Yang, Jeff Piper, Nathan Ie, Rama Pasumarthi, Nathan Lintz, Anitha Vijayakumar, Daniel Andor, Pedro Valenzuela, Minnie Lui, Cosmin Paduraru, Daiyi Peng, Katherine Lee, Shuyuan Zhang, Somer Greene, Duc Dung Nguyen, Paula Kurylowicz, Cassidy Hardin, Lucas Dixon, Lili Janzer, Kiam Choo, Ziqiang Feng, Biao Zhang, Achintya Singhal, Dayou Du, Dan McKinnon, Natasha Antropova, Tolga Bolukbasi, Orgad Keller, David Reid, Daniel Finchelstein, Maria Abi Raad, Remi Crocker, Peter Hawkins, Robert Dadashi, Colin Gaffney, Ken Franko, Anna Bulanova, Rémi Leblond, Shirley Chung, Harry Askham, Luis C. Cobo, Kelvin Xu, Felix Fischer, Jun Xu, Christina Sorokin, Chris Al-

berti, Chu-Cheng Lin, Colin Evans, Alek Dimitriev, Hannah Forbes, Dylan Banarse, Zora Tung, Mark Omernick, Colton Bishop, Rachel Sterneck, Rohan Jain, Jiawei Xia, Ehsan Amid, Francesco Piccinno, Xingyu Wang, Praseem Banzal, Daniel J. Mankowitz, Alex Polozov, Victoria Krakovna, Sasha Brown, MohammadHossein Bateni, Dennis Duan, Vlad Firoiu, Meghana Thotakuri, Tom Natan, Matthieu Geist, Ser tan Girgin, Hui Li, Jiayu Ye, Ofir Roval, Reiko Tojo, Michael Kwong, James Lee-Thorp, Christopher Yew, Danila Sinopalnikov, Sabela Ramos, John Mellor, Abhishek Sharma, Kathy Wu, David Miller, Nicolas Sonnerat, Denis Vnukov, Rory Greig, Jennifer Beattie, Emily Caveness, Libin Bai, Julian Eisenschlos, Alex Korchemniy, Tomy Tsai, Mimi Jasarevic, Weize Kong, Phuong Dao, Zeyu Zheng, Frederick Liu, Fan Yang, Rui Zhu, Tian Huey Teh, Jason Sanmiya, Evgeny Gladchenko, Nejc Trdin, Daniel Toyama, Evan Rosen, Sasan Tavakkol, Linting Xue, Chen Elkind, Oliver Woodman, John Carpenter, George Papamakarios, Rupert Kemp, Sushant Kafle, Tanya Grunina, Rishika Sinha, Alice Talbert, Diane Wu, Denese Owusu-Afriyie, Cosmo Du, Chloe Thornton, Jordi Pont-Tuset, Pradyumna Narayana, Jing Li, Saaber Fatehi, John Wieting, Omar Ajmeri, Benigno Uria, Yeongil Ko, Laura Knight, Amélie Héliou, Ning Niu, Shane Gu, Chenxi Pang, Yeqing Li, Nir Levine, Ariel Stolovich, Rebeca Santamaria-Fernandez, Sonam Goenka, Wenny Yustalim, Robin Strudel, Ali Elqursh, Charlie Deck, Hyo Lee, Zonglin Li, Kyle Levin, Raphael Hoffmann, Dan Holtmann-Rice, Olivier Bachem, Sho Arora, Christy Koh, Soheil Hassas Yeganeh, Siim Põder, Mukarram Tariq, Yanhua Sun, Lucian Ionita, Mojtaba Seyedhosseini, Pouya Tafti, Zhiyu Liu, Anmol Gulati, Jasmine Liu, Xinyu Ye, Bart Chrzaszcz, Lily Wang, Nikhil Sethi, Tianrun Li, Ben Brown, Shreya Singh, Wei Fan, Aaron Parisi, Joe Stanton, Vinod Koverkathu, Christopher A. Choquette-Choo, Yunjie Li, TJ Lu, Abe Ittycheriah, Prakash Shroff, Mani Varadarajan, Sanaz Bahargam, Rob Willoughby, David Gaddy, Guillaume Desjardins, Marco Cornero, Brona Robenek, Bhavishya Mittal, Ben Albrecht, Ashish Shenoy, Fedor Moiseev, Henrik Jacobsson, Alireza Ghaffarkhah, Morgane Rivière, Alanna Walton, Clément Crepy, Alicia Parrish, Zongwei Zhou, Clement Farabet, Carey Radebaugh, Praveen Srinivasan, Claudia van der Salm, Andreas Fidjeland, Salvatore Scellato, Eri Latorre-Chimoto, Hanna Klimczak-Plucińska, David Bridson, Dario de Cesare, Tom Hudson, Piermaria Mendolicchio, Lexi Walker, Alex Morris, Matthew Mauger, Alexey Guseynov, Alison Reid, Seth Odoom, Lucia Loher, Victor Cotruta, Madhavi Yenugula, Dominik Grewe, Anastasia Petrushkina, Tom Duerig, Antonio Sanchez, Steve Yadlowsky, Amy Shen, Amir Globerson, Lynette Webb, Sahil Dua, Dong Li, Surya Bhupatiraju, Dan Hurt, Haroon Qureshi, Ananth Agarwal, Tomer Shani, Matan Eyal, Anuj Khare, Shreyas Rammohan Belle, Lei Wang, Chetan Tekur, Mihir Sanjay Kale, Jinliang Wei, Ruoxin Sang, Brennan Saeta, Tyler Liechty, Yi Sun, Yao Zhao, Stephan Lee, Pandu Nayak, Doug Fritz, Manish Reddy Vuyyuru, John Aslanides, Nidhi Vyas, Martin Wicke, Xiao Ma, Evgenii Eltyshev, Nina Martin, Hardie Cate, James Manyika, Keyvan Amiri, Yelin Kim, Xi Xiong, Kai Kang, Florian Luisier, Nilesh Tripuraneni, David Madras, Mandy Guo, Austin Waters, Oliver Wang, Joshua Ainslie, Jason Baldridge, Han Zhang, Garima Pruthi, Jakob Bauer, Feng Yang, Riham Mansour, Jason Gelman, Yang Xu, George Polovets, Ji Liu, Honglong Cai, Warren Chen, XiangHai Sheng, Emily Xue, Sherjil Ozair, Christof Angermueller, Xiaowei Li, Anoop Sinha, Weiren Wang, Julia Wiesinger, Emmanouil Koukoumidis, Yuan Tian, Anand Iyer, Madhu Gurumurthy, Mark Goldenson, Parashar Shah, MK Blake, Hongkun Yu, Anthony Urbanowicz, Jennimaria Palomaki, Chrisantha Fernando, Ken Durden, Harsh Mehta, Nikola Momchev, Elahe Rahimtoroghi, Maria Georgaki, Amit Raul, Sebastian Ruder, Morgan Redshaw, Jinhyuk Lee, Denny Zhou, Komal Jalan, Dinghua Li, Blake Hechtman, Parker Schuh, Milad Nasr, Kieran Milan, Vladimir Mikulik, Juliana Franco, Tim Green, Nam Nguyen, Joe Kelley, Aroma Mahendru, Andrea Hu, Joshua Howland, Ben Vargas, Jeffrey Hui, Kshitij Bansal, Vikram Rao, Rakesh Ghiya, Emma Wang, Ke Ye, Jean Michel Sarr, Melanie Moranski Preston, Madeleine Elish, Steve Li, Aakash Kaku, Jigar Gupta, Ice Pasupat, Da-Cheng Juan, Milan Someswar, Tejvi M., Xinyun Chen, Aida Amini, Alex Fabrikant, Eric Chu, Xuanyi Dong, Amruta Muthal, Senaka Buthpitiya, Sarthak Jauhari, Nan Hua, Urvashi Khandelwal, Ayal Hitron, Jie Ren, Larissa Rinaldi, Shahar Drath, Avigail Dabush, Nan-Jiang Jiang, Harshal Godhia, Uli Sachs, Anthony Chen, Yicheng Fan, Hagai Taitelbaum, Hila Noga, Zhuyun Dai, James Wang, Chen Liang, Jenny Hamer, Chun-Sung Ferng, Chenel Elkind, Aviel Atias, Paulina Lee, Vít Listík, Mathias Carlen, Jan van de Kerkhof, Marcin Pikus, Krunoslav Zaher, Paul Müller, Sasha Zykova, Richard Stefanec, Vitaly Gatsko, Christoph Hirnschall, Ashwin Sethi, Xingyu Federico Xu, Chetan Ahuja, Beth Tsai, Anca Stefanoiu, Bo Feng, Keshav Dhandhania, Manish Katyal, Akshay Gupta, Atharva Parulekar, Divya Pitta, Jing Zhao, Vivaan Bhatia, Yashodha Bhavnani, Omar Alhadlaq, Xiaolin Li, Peter Danenberg, Dennis Tu, Alex Pine, Vera Filippova, Abhipso Ghosh, Ben Limonchik, Bhargava Urala, Chaitanya Krishna Lanka, Derik Clive, Yi Sun, Edward Li, Hao Wu, Kevin Hongtongsak, Ianna Li, Kalind Thakkar, Kuanysh Omarov, Kushal Majmundar, Michael Alverson, Michael Kucharski, Mohak Patel, Mudit Jain, Maksim Zabelin, Paolo Pelagatti, Rohan Kohli, Saurabh Kumar, Joseph Kim, Swetha Sankar, Vineet Shah, Lakshmi Ramachandruni, Xiangkai Zeng, Ben Bariach, Laura Weidinger, Amar Subramanya, Sissie Hsiao, Demis Hassabis, Koray Kavukcuoglu, Adam Sadovsky, Quoc Le, Trevor Strohman, Yonghui Wu, Slav Petrov, Jeffrey Dean, and Oriol Vinyals. 2023. Gemini: A family of highly capable multimodal models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton

Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. LLaMA 2: Open foundation and fine-tuned chat models.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023a. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *Proceedings of the 12th International Conference on Learning Representations*.

Jiali Zeng, Yongjing Yin, Yufan Jiang, Shuangzhi Wu, and Yunbo Cao. 2022. Contrastive learning with prompt-derived virtual semantic prototypes for unsupervised sentence embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2022*.

A Testing Set Collection Detail

To deduplicate, we utilized ChatGPT (OpenAI, 2022) to transform all command lines into concise descriptions that encapsulate the purpose and intention. Subsequently, we transformed these brief descriptions into embeddings using GTE-Large (Li et al., 2023), which achieved SOTA performance on the MTEB leaderboard (Muennighoff et al., 2022) among models of similar size. We then applied DBSCAN (Ester et al., 1996) for clustering the embeddings. Through this approach, each cluster contains command lines with highly similar semantics based on their explanations. Finally, we extracted two command lines from each cluster, resulting in a testing set comprising 2,807 command lines in total.

B Initial-Seed Collection Detail

To gather high-quality initial seeds, we first extracted all command lines executed in DARPA Transparent Computing (DARPA, 2019), totaling 142,886 unique command lines. We then applied a heuristic filtering process to eliminate command lines that are semantically similar and differ only slightly, such as variations in log file suffixes. Ultimately, we curated 722 command lines as part of the initial seeds.

To further extend the initial seeds, we formulated 796 command lines based on the descriptions and corresponding syntax found in Windows Commands[3]. Additionally, we parsed all example command lines from SS64[4], totaling 497 command lines, and collected an additional 46 command lines from GitHub. Together, these contributions amounted to 2,061 high-quality and diverse command lines, which were integrated to form our initial command-line seeds.

C Windows-Commands Coverage Rate Detail

While many Windows commands are listed separately, several utilize identical executable files, like 'reg add' and 'reg copy'. To present a unified perspective, commands with shared executable were further grouped into one, resulting in 306 unique Windows commands.

For common file name extensions, we first parsed all extensions from a clean Windows 10

[3] Windows Command-line reference A-Z
[4] SS64 Windows CMDs

- "C:\Windows\System32\bitsadmin.exe" /transfer 59697582645 /priority foreground http://example.com/example1234 "C:\Users\Public\Videos\V123456789\log32.dll" - powershell -command "Start-BitsTransfer -Source 'http://malicious.com/malicious1234' -Destination 'C:\Users\Public\Videos\V99999999\log32.dll' -Priority High"
- "cmd" /c "net use \\REMOTEDIR /user:Administrator password /persistent:no" - python -c "import os; os.system('net use \\REMOTEDIR /user:Administrator password /persistent:no')"
- reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall" /f "Chrome" /s - Get-ItemProperty HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall* | Select-Object -Property DisplayName, UninstallString | Where-Object {$_.DisplayName -like '*Chrome*'} | Format-Table -AutoSize
- schtasks /create /tn "TaskName" /tr "C:\Path\to\program.exe" /sc daily /st 00:00 - cronjob schedule daily 00:00 /path/to/program

Table 9: The similar command-line pairs in CmdDataset. Similar command lines are not merely similar on the lexical level but also in terms of their intrinsic purpose and semantic meaning.

Here are 12 Windows command line examples for referencing:
1. **{sampled command line seed 1}**
2. **{sampled command line seed 2}**
...
12. **{sampled command line seed 12}**

Your job is to synthesize 4 new Windows command lines. Please adhere to the following synthesizing guidelines:
- Ensure diverse command lines in appearance, argument value, purpose, result, and length, particularly making sure the generated command lines differ significantly from the reference command lines in every aspect.
- Prioritize practicality in generated commands, ideally those executed or executable. For example, please give me real argument value, filename, IP address, and username.
- Include Windows native commands, commands from installed applications or packages (for entertainment, work, artistic, or daily purposes), commands usually adopted by IT, commands corresponding with mitre att&ck techniques, or even some commonly used attack command lines. The more uncommon the command line, the better.,
- Do not always generate short command lines only. Be creative to synthesize all kind of command lines.

Give me your generated command lines only without any explanation or anything else. Separate each generated command line with "\n" and add a prefix "<CMD>" before each generated cmd.

Figure 6: The prompt used for generating a single command line. 12 exemplary command lines are randomly sampled from total command-line seeds for in-context demonstration.

```
Your task is to generate a similar Windows command line for each entry in the
following command line list.
In this task, 'similar' means that the command lines share the same purpose, or
intention, rather than merely having a similar appearance.

Consequently, the generated command lines may differ significantly in argument
values, format, and order from the original command line, or even from a different
executable file, as long as they serve a similar purpose or intention.
{query command line}

Be creative to make the command lines appear distinctly different while adhering
to the defined 'similar' criteria. For instance, you might employ obfuscation
techniques, randomly rearrange the order of arguments, change the way to call the
exe file, or substitute the executable file with a similar one.
Please provide only the generated similar command lines without any explanation,
prefixed with "<CMD>", and separate each command line with "\n".
```

Figure 7: The prompt used for generating similar command lines. The query command line is randomly sampled from the total command-line seeds.

Command Line	Explanation	Labels
`net use Z: \\192.168.1.1\SharedFolder /user:administrator Passw0rd! | findstr /i connected`	This command line is used to map a network drive to the letter Z, connect to a shared folder on a specific IP address using administrator credentials, and then search for the keyword "connected" in the output.	Positive
`schtasks /create /sc weekly /d MON,TUE,WED,THU,FRI /tn "WeeklyBackup" /tr "C:\Scripts\backup.bat" /st 18:00`	This command line creates a scheduled task to run a backup script every weekday at 6:00 PM.	Positive
`tasklist /fi "IMAGENAME eq notepad.exe" /fo list | find "1234"`	This command line is used to list all running processes with the name "notepad.exe" and then search for a specific process ID "1234" within the list.	Positive
`findstr /s /i /m "hello world" "world take care" C:\Users\* \*.pdf`	Search for the phrase "hello world" in all PDF files located in the C:\Users directory and its subdirectories, ignoring case and only displaying the file names that contain the phrase.	Neutral

Table 10: Example of command lines and their corresponding explanations generated by GPT-3.5-Turbo (OpenAI, 2022). The rightmost column denotes the labels (e.g., Positive, Neutral, or Negative) assigned by the expert.

virtual machine, and removing those with special characters and frequencies lower than 0.05%. This process yielded a total of 75 common extensions. We then identified how many of these extensions are included in our training set.

D Evaluation Metrics Detail

MRR@K is a key metric in information retrieval and recommendation systems. It calculates the average reciprocal rank of the first relevant item within a list of ranked results, focusing on the top K positions. This method helps us gauge how well the most relevant item ranks among the top 10 with the highest predicted scores. The Top@K metric is similar to MRR@K but differs in that it awards a score if the ground truth is within the top K ranks, without the rank-dependent decay seen in MRR.

E Hyperparameters and Training Process of CmdCaliper Detail

We trained CmdCaliper for 2 epochs using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.00002 and a batch size of 64. For the temperature parameter τ in Equation 1, we set it to 0.05. CmdCaliper was trained on three distinct model scales: small, base, and large. These models were initialized from the GTE-small (Li et al., 2023), GTE-base, and GTE-large, respectively.

We randomly selected 1,000 similar pairs from the training set to form a validation set. Evaluations on the validation set were conducted every 50 training steps. The checkpoint that demonstrated optimal performance on the validation set was then used for subsequent evaluations on the testing set.

F Semantic-Based Malicious Command-Line Detection Detail

Initially, we define the pre-collected set of malicious command lines as the 'malicious gene pool'. Given a new command line, we leverage a command-line embedding model to obtain their embedding vectors, and compute the semantic similarity between each command line within the malicious gene pool. If one of the similarities exceeds a pre-defined threshold, we classify the new command line as malicious. This approach enables us to detect malicious command lines from a semantic perspective, even when attackers attempt to obfuscate command lines to evade pattern-based detection.

In this experiment, we utilize the open-source dataset: atomic-red-team (Canary), which encompasses a variety of command lines corresponding to numerous MITRE ATT&CK techniques (e.g., Abuse Elevation Control Mechanism or Browser Session Hijacking). The dataset consists of a set of techniques, denoted as $\{t_1, t_2, \ldots, t_n\}$. Within each technique t_i, we obtain a set of malicious command lines, denoted as $\mathbb{L}_i$, containing M_i entries: $\mathbb{L}_i = \{c_1^i, c_2^i, \ldots, c_{M_i}^i\}$. By unioning all these malicious command line sets, we form a total command line set $\mathbb{A}$, which consists of 1,523 malicious command lines across various techniques, represented as $\mathbb{A} = \bigcup_{i=1}^{n} \mathbb{L}_i$. To construct the malicious gene pool $\mathbb{P}_i$ for each technique t_i, we select the first $r\%$ of the command lines from each technique, defined as $\mathbb{P}_i = \{c_1^i, c_2^i, \ldots, c^i_{\lceil \frac{r}{100} \times M_i \rceil}\}$. The remaining command lines form the incoming command line set $\mathbb{O}_i$, denoted as: $\mathbb{O}_i = \{c^i_{\lceil \frac{r}{100} \times M_i \rceil + 1}, \ldots, c^i_{M_i}\}$.

For malicious gene pool construction, we exclude techniques with fewer than 9 malicious command lines, resulting in a total of 55 distinct malicious gene pools corresponding to different techniques. For the evaluation of each technique t_i, we first exclude the command lines in the malicious gene pool to form a candidate command line set $\mathbb{C}_i$, denoted as $\mathbb{C}_i = \mathbb{A} \setminus \mathbb{P}_i$. We treat the incoming command line set $\mathbb{O}_i$ as the positive set, while the negative set $\mathbb{G}_i$ is formed by excluding all command lines corresponding to the technique t_i, denoted as $\mathbb{G}_i = \mathbb{A} \setminus \mathbb{L}_i$.

Given a command line c_j^i from the candidate command line set $\mathbb{C}_i$ and an embedding model E, the embedding vector e_j^i can be computed by $e_j^i = E(c_j^i)$. Subsequently, the malicious score $s_{c_j^i}$ of the command line c_j^i for the technique t_i is determined by calculating its maximum similarity with each command line p_k^i in the malicious gene pool $\mathbb{P}_i$:

$$s^i_{c^i_j} = \max_{p_k^i \in \mathbb{P}_i} S(e_j^i, E(p_k^i)) \tag{2}$$

where $S(\cdot)$ is the similarity function (e.g., cosine similarity function). Note that if the command line c_j belongs to the positive set $\mathbb{O}_i$ of the technique t_i, then the malicious score should exceed the pre-defined threshold τ for correct classification; otherwise, the score should be below the threshold if the command line is in the negative set $\mathbb{G}_i$.

We iteratively evaluated all 55 techniques and

concatenated all malicious scores to compute the area under the curve (AUC). This evaluation metric aligns with real-world application scenarios where a static threshold is usually applied across all techniques.

G Classification Dataset Synthesis Detail

These commands to synthesize the command-line classification dataset were chosen based on their ability to accept a wide range of randomly generated strings as arguments, provided that the corresponding file exists. The classifier's task is to identify the corresponding Windows command, regardless of the argument's length or complexity. For example, the command lines "find 'fewj2po3kdlewfmpemrgborktig fe34krop4k5ogjs9rkgewfefw34f'" and "find 'test'" should both be correctly categorized under the 'find' command. This highlights the importance of a robust command-line embedding model in encoding the command information within its embeddings, as it plays a crucial role in determining the purpose of each command line.

In this experiment, we used the pattern "<command> '<argument value>'" to randomly generate 7,000 command lines for each command. Of these, 3,500 were assigned to the training set and the remaining 3,500 to the testing set. The arguments for each command line were formed by concatenating seven random strings, made up of ASCII letters and digits, with lengths ranging from 1 to 20 characters, separated by spaces. To increase the difficulty of classification and simulate the obfuscation techniques that attackers might use in real-world scenarios to evade detection, we also randomly incorporated seven different commands into the argument values. For example, a synthesized command line might read: "certutil.exe 'msiexec tr9QI1L find C print oGod 5K 7Okf4 2ZcVT9 rundll32 sc query mNjIL robocopy q5'", where only the first command, 'certutil', is valid.

We randomly selected 20% of the training set to serve as a validation set in the search for optimal hyperparameters. Subsequently, we utilized the obtained optimal hyperparameters to train a logistic regression classifier for performance evaluation on the testing set.