

FWNL: Um Sistema de Processamento de Linguagem Natural para Configuração de Múltiplos Firewalls

José A. S. Goulart², Washington L. M. F. Junior², Kayuã O. Paim^{1,2},
Gabriel M. Lunardi², Diego L. Kreutz^{1,2}, Rodrigo B. Mansilha^{1,2}

¹ Programa de Pós-Graduação em Engenharia de Software (PPGES)

² Universidade Federal do Pampa (Unipampa)

{josegoulart.aluno, washingtonfagundes, kayuapaim.aluno,
gabriellunardi, kreutz, mansilha}@unipampa.edu.br

Resumo. Neste trabalho apresentamos um sistema baseado em processamento de linguagem natural, denominado FWNL, para fornecer configurações de firewall em FWLang e habilitar a integração automática ao FWUnify. O objetivo é desenvolver um pipeline com reconhecimento de entidades mencionadas para reduzir a curva de aprendizagem e mitigar erros operacionais na configuração de múltiplos firewalls. Implementamos um protótipo funcional do FWNL e demonstramos sua flexibilidade, a partir da vinculação de três ambientes de conversação: um baseado em linha de comando, outro baseado em serviço web e um terceiro baseado em um aplicativo de mensagens.

1. Introdução

A configuração de firewalls de múltiplas marcas e modelos requer o domínio prévio ou aprendizado de uma variedade de sintaxes e métodos de configuração para implementar corretamente as políticas de segurança desejadas. Para ajudar a superar esses desafios, foi proposta a ferramenta FWUnify [Fiorenza et al., 2021a] – uma solução para configuração integrada e automática de firewalls. Apesar dos avanços significativos, a FWUnify ainda apresenta oportunidade de melhoria no âmbito de interfaces com o usuário. Por exemplo, a FWUnify aceita como parâmetro de entrada um arquivo textual descrevendo uma intenção na linguagem FWLang [Fiorenza et al., 2021b]. Para aqueles administradores de sistemas acostumados com interfaces de linha de comando e descrição de configuração utilizando editores convencionais de texto, como é comum para alguns tipos de atividades relacionadas ao gerenciamento de firewalls [Botta et al., 2007, Haber and Bailey, 2007], a entrada da FWUnify pode ser considerada minimamente prática e intuitiva. No entanto, há um outro grupo de usuários modernos, ou ainda aqueles que gastam quatro ou menos horas semanais gerenciando firewalls, que prefere interfaces gráficas e *chatbots* [Voronkov et al., 2019].

Na prática, uma interface gráfica pode facilitar a criação, organização e visualização das políticas de segurança para os administradores de sistemas. Similarmente, um *chatbot*, utilizando intenções em linguagem natural, pode simplificar e agilizar o gerenciamento de políticas de segurança para gestores de tecnologia da informação. Entretanto, para simplificar, padronizar e agilizar o desenvolvimento de aplicações de gerenciamento, é importante que as interfaces utilizem uma mesma *linguagem universal*, como a FWLang [Fiorenza et al., 2021b].

Para oferecer opções de interfaces mais amigáveis e intuitivas, propomos e implementamos o FWNL¹, um sistema de interfaces (*frontend*) e serviços (*backend* – e.g., robô de conversação e componentes de integração) que permite intermediar a descrição de políticas de segurança em linguagem natural e a consequente tradução e aplicação das respectivas regras nos firewalls através da ferramenta FWUnify. Atualmente, o FWNL disponibiliza interfaces em linguagem natural a partir de três meios diferentes: via linha de comando, via navegador web ou via aplicativo de mensagens instantâneas Telegram. O manual² com os requisitos e as instruções para a instalação e execução estão disponíveis no mesmo repositório da ferramenta.

As contribuições técnicas deste trabalho são: (a) especificação de uma arquitetura de conversação utilizando linguagem natural para permitir a configuração integrada de firewalls em redes corporativas; (b) implementação de uma instância da arquitetura, denominada FWNL, modular e extensível; e (c) demonstração da modularização, através do acoplamento em três sistemas de interface humano-computador, e da eficácia do resultado aplicado em um ambiente real.

O restante deste trabalho está organizado como segue. Apresentamos a arquitetura e a implementação do FWNL nas seções 2 e 3, respectivamente. Em seguida, apresentamos alguns resultados de avaliações realizadas com o FWNL na Seção 4. Por fim, detalhamos a demonstração planejada para o Salão de Ferramentas e as considerações finais do trabalho na Seção 5.

2. Arquitetura

A Figura 1 apresenta: (a) arquitetura do FWNL, (b) fluxograma geral de conversação, e (c) fluxograma de processamento de perguntas e respostas. Na Figura 1(a) é possível visualizar a contribuição deste trabalho e sua integração com o FWUnify. Especificamente, o FWNL é composto por um Chatbot (*i.e.*, robô de conversação) como principal componente de *backend*) e um conjunto inicial formado por três *frontends*: CLI, Messenger e Web. O FWNL conversa com o usuário e gera intenções de configurações de firewall na linguagem FWLang que podem ser aplicadas manualmente ou automaticamente, caso configurado, em alguma instância do FWUnify. O FWUnify, por sua vez, aplica as configurações nos firewalls do ambiente (*e.g.*, rede corporativa).

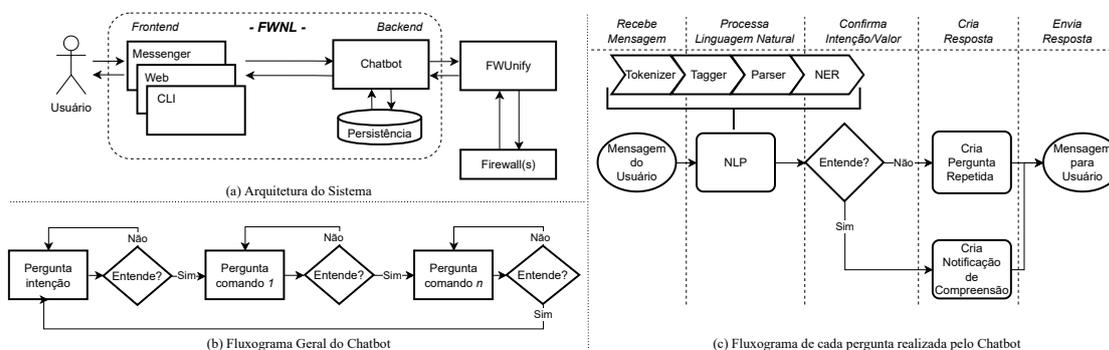


Figura 1. Arquitetura do FWNL.

¹<https://github.com/fw-tools/fwnl>

²<https://github.com/fw-tools/fwnl#readme>

Adicionalmente, o FWNL contém um componente de persistência que permite carregar dados sobre o sistema de firewalls (*e.g.*, endereços de *hosts*, listas de VLANs) para facilitar a conversação. O componente também permite armazenar histórico (se o usuário permitir) de conversas, intenções implementadas e impacto das intenções. Esse histórico poderá ser utilizado no futuro para treinar Chatbots como o FWNL.

O Usuário interage com algum *frontend* por meio de um esquema de perguntas e respostas, como ilustrado no fluxograma da Figura 1(b). Empregamos esse método como um primeiro passo em direção à uma conversação mais livre, *e.g.*, se o usuário desejar especificar uma regra com apenas uma mensagem. Por ora, o esquema de perguntas e respostas é guiado, *i.e.*, a construção das regras ocorre passo-a-passo. Essa estratégia é comumente empregada em robôs de conversação de auto-atendimento ou quando não há histórico disponível para gerar sistemas mais inteligentes que aprendem a cada seção de conversação, também conhecidos como agentes conversacionais generativos [Adamopoulou and Moussiades, 2020]. Além do processo de conversação, no futuro vislumbra-se acoplar também um processo de monitoramento, de modo que o Chatbot possa aprender além do que o usuário deseja, o que ele realmente precisa.

A FWLang, a linguagem do FWUnify, é baseada no conceito de redes baseadas em intenções ou IBNs (*Intent-Based*). Uma intenção da FWLang permite representar políticas utilizadas em firewalls modernos, como lista de controle de acesso (*access control list* - ACL), filtros de URL e *traffic shaping*. Inicialmente, o Chatbot procura descobrir qual a *intenção* do usuário considerando um catálogo disponível. Dependendo da intenção do usuário, o Chatbot pergunta sobre cada *comando* relacionado à intenção de interesse do usuário. Para cada comando, existe uma série de *valores* aceitáveis. Por exemplo, a pergunta sobre destino pode aceitar endereços IP (versões 4 e 6), nomes válidos de *hosts* ou intervalos de endereços. Caso a resposta não seja coerente com nenhum valor possível para o comando, a pergunta é repetida. Caso contrário, o Chatbot pergunta sobre o próximo comando (se ainda houver algum comando a ser respondido) ou pergunta sobre uma nova intenção.

Cada mensagem do Usuário é processada de acordo com a Figura 1(c). A etapa NLP (*Natural Language Processing*) compreende um *pipeline* composto por diversas etapas. Visamos facilitar a expansão desse *pipeline* para agregar mais inteligência ao FWNL. Nesse sentido, consideramos alguns componentes essenciais do processamento de linguagem natural [Manning and Schutze, 1999, Manning, 2008]:

- Tokenizer: dada uma sequência de caracteres, o Tokenizer separa a sentença em partes denominadas *tokens*, ou termos, descartando caracteres semanticamente inúteis (*e.g.*, pontuação). Por exemplo, a sentença “Eu quero definir uma ACL.” teria os seguintes *tokens*: “Eu”, “quero”, “definir”, “uma” e “ACL”.
- Tagger: identifica a qual classe gramatical pertence cada *token*. Esse processo é também conhecido como identificação das partes do discurso (POS - *Part-Of-Speech*).
- Parser: a sentença, quando separada em *tokens*, perde a semântica. Para evitar que isso ocorra, o Parser induz uma árvore de dependência sintática entre os *tokens*.
- NER (*Named Entity Recognition*): o reconhecimento de entidades nomeadas é um ramo do processamento de linguagem natural que permite identificar entidades (*e.g.*, locais, países, nomes de pessoas, moedas) que sejam comuns na língua e, com isso, melhorar o entendimento das sentenças [Nadeau and Sekine, 2007].

3. Implementação

Para concretizar o FWNL implementamos uma versão funcional e a disponibilizamos publicamente. Essa implementação foi escrita em linguagem Python 3.

3.1. Backend

Como mencionado anteriormente, a *pipeline* NLP é o principal componente do FWNL e foi concebido para ser apto a expansão e de fácil integração com novos componentes. Para atingir esses objetivos, aproveitamos a biblioteca Python spaCy³, que reúne uma série de *pipelines* NLP treinados para diversas linguagens naturais. Entre os *pipelines* disponíveis estendemos uma instância que concretiza o esquema apresentado na Figura 1(c) na língua inglesa. Especificamente, nós aproveitamos os componentes Tokenizer, Tagger, Parser da maneira como são disponibilizados e substituímos o componente NER, que é pré-treinado para uso geral. Nós estendemos o NER de propósito geral para incluir aspectos específicos, e ser capaz de rotular entidades relacionadas a configurações de *firewall* (e.g., IPv4, IPv6, protocolos de rede, valores de vazão). Como ainda não dispomos de histórico de conversação para treinar uma NER, construímos um NER baseado em uma série de expressões regulares. As regras podem estar escritas diretamente em expressão e regulares ou na Linguagem de Domínio Específico (DSL) denominada RITA⁴, que é de nível mais alto de abstração. Nesta versão não implementamos a camada de persistência.

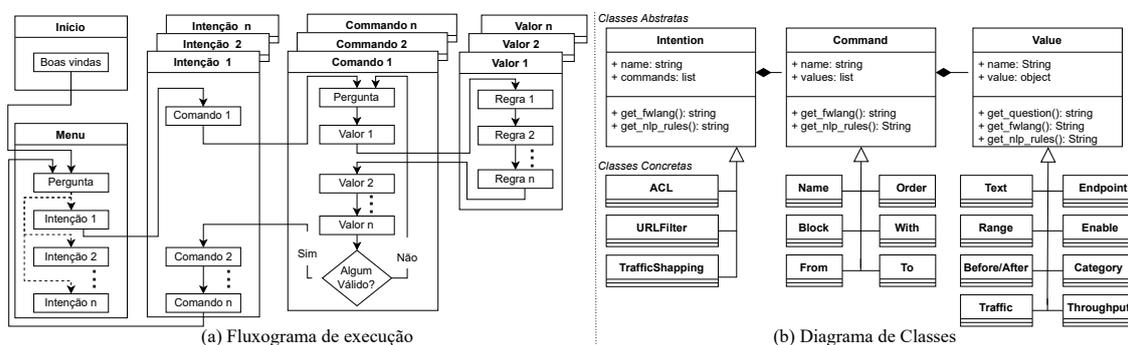


Figura 2. Organização do FWNL implementado.

Para prover reutilização e reaproveitamento de código, e facilitar a manutenibilidade e extensibilidade do FWNL, organizamos o Chatbot (*backend*) conforme a Figura 2. A Figura 2(a) detalha o processo seguido pelo Chatbot, que inicia com alguma mensagem de boas vindas e segue para o menu principal. Nesse menu deve ser selecionado alguma das *intenções* implementadas. O Chatbot itera sobre as possíveis intenções do Usuário e confirma com ele se a mais similar é a intenção desejada. Considerando uma intenção selecionada, cada um dos seus *comandos* é processado sequencialmente. A resposta de um comando é processada considerando cada uma das regras de *valores* pertencentes ao comando. Ao final, escolhe-se como resposta ao comando o primeiro valor que houver *match* de regra. Futuramente, pretendemos aperfeiçoar essa decisão, por exemplo considerando graus de proximidade para aceitar e sugerir correções para erros de digitação.

³<https://spacy.io/>

⁴<https://github.com/zaibacu/rita-dsl>

A Figura 2(b) apresenta a estrutura de classes da nossa implementação do FWNL. Atualmente o FWNL contém o mapeamento de três intenções que concretizam a classe *Intention: ACL* (*i.e.*, controle de acesso), *URLFilter* (*i.e.*, filtro de URL), e *TrafficShaping* (*i.e.*, modelagem de tráfego). Como a Figura 2(b) omite as relações de composição entre classes por fins de simplificação, apresentamos a seguir alguns exemplos para esclarecer. A lista de comandos da classe *ACL* inclui, entre outros comandos, o *From* e *To*, que modelam a origem e o destino da configuração, respectivamente. Ambos comandos aceitam valores *Endpoint* (*i.e.*, ponto final de comunicação) e *Range* (*i.e.*, intervalo de endereços). A classe *Endpoint* aceita regras de formação IPv4, IPv6 e uma lista de nomes de hosts cadastrados (*e.g.*, servidor, laboratório, secretaria).

3.2. Frontend

Para permitir a interação entre Usuário e Chatbot, e demonstrar o uso da API, implementamos três *interfaces*: *CLI*, *Web* e *Messenger*. Essas interfaces são exemplificadas na Figura 3 e discutidas em seguida.

```

FwBot Terminal: Good evening! I'm here to help you in setting up firewalls.
User: i want to configure an ACL
FwBot Terminal: Do you want to make ACL (i.e., Access Control List)?
User: yes
FwBot Terminal: Great. So, let me help you with ACL.
Let me ask you some questions...
FwBot Terminal: Regarding Name.
What's its value? Hint: Text
User: test-acl
FwBot Terminal: I got it: text('test-acl')
FwBot Terminal: Regarding From.
What's its value? Hint: Endpoint|Range
User: 10.0.0.0/24
FwBot Terminal: I got it: range('10.0.0.0/24')
FwBot Terminal: Regarding To.
What's its value? Hint: Endpoint|Range
User: 138.197.63.21
FwBot Terminal: I got it: endpoint('138.197.63.21')
FwBot Terminal: Regarding Block.
What's its value? Hint: Protocol
User: https
FwBot Terminal: I got it: traffic('https')
FwBot Terminal: Regarding Order.
What's its value? Hint: Before|After
User: before
FwBot Terminal: I got it: before('all')
FwBot Terminal: Here's your final configuration:
FwBot Terminal:
define intent acl:
name      text('test-acl')
from      range('10.0.0.0/24')
to        endpoint('138.197.63.21')
block     traffic('https')
order     before('all')

```

(a) CLI



(b) Web



(c) Messenger

Figura 3. Interfaces implementadas para comunicação com o Chatbot.

A primeira interface implementada (Figura 3(a)) é a linha de comando. Ela é importante para exemplificar a utilização do Chatbot, para fins de desenvolvimento e testes e também para atender aqueles administradores de sistemas que preferem interfaces de linha de comando. A segunda interface implementada (Figura 3(b)) é um ambiente de conversação Web. Ela é importante para abranger um público mais amplo possível pois requer apenas um navegador moderno. Na interface *Web* usamos as bibliotecas *Flask*, *Gunicorn* e *Dart Sass*. A terceira interface é através do aplicativo de mensagens *Telegram*⁵. Ela foi implementada visando um ambiente de conversação acessível com auxílio de opções clicáveis. Nessa interface usamos a biblioteca *python-telegram-bot*.

⁵<https://telegram.org/>

4. Avaliação

Para testar a proposta, executamos um cenário em ambiente virtualizado contendo uma instância do FWNL acoplado ao FWUnify configurado para gerenciar um firewall IPTables. Os sistemas foram executados em uma topologia de rede virtualizada composta por duas máquinas virtuais (VM). Iniciamos e monitoramos um processo de comunicação entre as duas VMs, e em seguida usamos o FWNL para bloquear a comunicação e, por fim, averiguamos o impacto no processo de comunicação. Abaixo, apresentamos os resultados na Figura 4 e detalhamos esse processo em seguida.

```
File Edit View Terminal Tabs Help
@debian:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination

@debian:~$
```

(a) Regras do IPTables antes.

```
File Edit View Terminal Tabs Help
@debian:~$ ping 192.168.56.4 -w 5 -c 5
PING 192.168.56.4 (192.168.56.4) 56(84) bytes of data:
64 bytes from 192.168.56.4: icmp_seq=1 ttl=64 time=3.27 ms
64 bytes from 192.168.56.4: icmp_seq=2 ttl=64 time=2.79 ms
64 bytes from 192.168.56.4: icmp_seq=3 ttl=64 time=4.66 ms
64 bytes from 192.168.56.4: icmp_seq=4 ttl=64 time=4.27 ms
64 bytes from 192.168.56.4: icmp_seq=5 ttl=64 time=3.52 ms

--- 192.168.56.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 64ms
rtt min/avg/max/mdev = 2.789/3.701/4.657/0.680 ms
@debian:~$
```

(b) Saída do comando *ping* antes.

1	FwBot	Good evening! I'm here to help you in setting up firewalls.	13	User	192.168.56.4
2	User	I want to configure an ACL	14	FwBot	I got it: endpoint('192.168.56.4')
3	FwBot	Do you want to make ACL (i.e., Access Control List)?	15	FwBot	Regarding Block. What's its value? Hint: Protocol
4	User	yes	16	User	https
5	FwBot	Great. So, let me help you with ACL. Let me ask you some questions...	17	FwBot	I got it: traffic('https')
6	FwBot	Regarding Name. What's its value? Hint: Text	18	FwBot	Regarding Order. What's its value? Hint: Before—After
7	User	test-acl	19	User	before
8	FwBot	I got it: text('test-acl')	20	FwBot	I got it: before('all-intents')
9	FwBot	Regarding From. What's its value? Hint: Endpoint—Range	21	FwBot	Here's your final configuration:
10	User	0.0.0.0	22	FwBot	define intent acl:
11	FwBot	I got it: endpoint('0.0.0.0')			name text('test-acl')
12	FwBot	Regarding To. What's its value? Hint: Endpoint—Range			from endpoint('0.0.0.0')
					to endpoint('192.168.56.4')
					block traffic('https')
					order before('all-intents')

(c) Transcrição da conversa.

```
File Edit View Terminal Tabs Help
@debian:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination

DROP     tcp  --  default                192.168.56.4
        tcp dpt:https

@debian:~$
```

(d) Regras do IPTables após.

```
File Edit View Terminal Tabs Help
@debian:~$ ping 192.168.56.4 -w 5 -c 5
PING 192.168.56.4 (192.168.56.4) 56(84) bytes of data:
ping: sendmsg: Operation not permitted

--- 192.168.56.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 108ms
@debian:~$
```

(e) Saída do comando *ping* após.

Figura 4. Resultado da aplicação de uma intenção ACL ao firewall IPTables.

Nós seguimos os seguintes passos: (a) usando VirtualBox (6.1.36, hospedado em um ambiente Windows rodando em uma máquina AMD Ryzen) instanciamos duas máquinas virtuais (VM) com sistema Debian 11; (b) numa VM, executamos o FWNL, o FWUnify e os micro serviços necessários para seu funcionamento, conforme descrito na

documentação da ferramenta⁶; (c) verificamos na Figura 4(a) que as políticas do IPTables da primeira VM estavam vazias e predefinidas como *ACCEPT*, e na Figura 4(b) que a segunda VM é alcançável a partir da primeira; (d) então, conforme transcrição mostrada na Figura 4(c), nós conversamos com o Chatbot (linhas 1-21) para criar uma intenção ACL (linha 22); (e) em seguida, nós copiamos a resposta final do Chatbot (linha 22) para um arquivo e o carregamos no FWUnify; (f) verificamos na Figura 4(d) que as políticas do IPTables da primeira VM contém as configurações adicionadas pela FWUnify, e na Figura 4(e) que a segunda VM não é mais alcançável a partir da primeira.

Portanto, foi possível apurar que a intenção ACL construída pelo Chatbot é compatível com a ferramenta FWUnify. Testes futuros devem ser feitos para averiguar o comportamento do ChatBot com outros tipos de regras para firewalls (*e.g.*, Traffic Shaping, URL Filter). Posto isso, fica claro que há espaço para melhorias, principalmente quanto a integração com o ambiente gerenciado pelo FWUnify, visto que nossa implementação do Chatbot se encarrega apenas de gerar uma configuração válida mas não ativar tal ou, quando necessário, remover regras existentes. Outros resultados e discussões acerca da exatidão do processo de tradução e da eficácia de políticas geradas, para firewalls Cisco, IPTables e OpenFlow, estão disponíveis em [Fiorenza et al., 2021b].

5. Considerações Finais

Demonstração

A demonstração do FWNL será realizada em um ambiente contendo duas máquinas virtuais e um firewall IPTables. O ambiente será executado em um dispositivo próprio dos autores. As funcionalidades da ferramenta serão apresentadas através dos seguintes passos: (a) apresentação da política de segurança ACL utilizando a linguagem FWlang; (b) demonstração do processo de conversação com FWNL para aplicar uma ACL usando o FWunify; (c) apresentação dos comandos gerados automaticamente pela ferramenta no IPTables; e (d) demonstração do funcionamento das regras aplicadas no IPTables.

Conclusão

Neste trabalho, apresentamos o FWNL: um sistema de processamento de linguagem natural para configuração de múltiplos firewalls. Apresentamos uma arquitetura modular que é demonstrada por meio de uma implementação funcional composta por três *front-ends* (CLI, Web e Messenger); e um *back-end* baseado em perguntas e respostas que gera intenção em formato FWLang e que pode ser aplicado em múltiplos firewalls usando FWUnify.

Vislumbramos diversas oportunidades de desenvolvimento e pesquisa. No campo de desenvolvimento, pretendemos integrar FWNL com o FWUnify e incluir regras para os componentes restantes da linguagem FWLang. Além disso, pretendemos implementar a camada de persistência para incluir facilidades para configuração local, como listas de IPs, hosts. A camada de persistência será fundamental para o principal objetivo de pesquisa: desenvolver métodos de aprendizado por reforço como forma de melhorar o processo conversacional [Uc-Cetina et al., 2022]. Inicialmente, por meio de *feedback* explícito dos usuários e, posteriormente, identificando padrões de conversação e intenções de alto nível

⁶<https://github.com/fw-tools/fwunify#readme>

com auxílio de técnicas de aprendizado profundo, tais como representações distribuídas de palavras e frases [Mikolov et al., 2013] e transformadores bidirecionais para compreensão da linguagem (*i.e.*, BERT) [Devlin et al., 2019]. Ainda, pretendemos disponibilizar opções de conversação em outros idiomas sendo, o português, a primeira prioridade.

Agradecimentos

Este trabalho foi realizado com apoio da Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), por meio dos editais ARD 10/2020 e PROBIC.

Referências

- Adamopoulou, E. and Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006.
- Botta, D., Werlinger, R., Gagné, A., Beznosov, K., Iverson, L., Fels, S., and Fisher, B. (2007). Towards understanding it security professionals and their tools. In *Proceedings of the 3rd SOUPS*, page 100–111. ACM.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (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*, pages 4171–4186. Association for Computational Linguistics.
- Fiorenza, M., Kreutz, D., Mansilha, R., Macedo, D., Feitosa, E., and Immich, R. (2021a). FWunify: uma ferramenta para simplificar a configuração de múltiplos firewalls. In *Anais Estendidos do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. SBC.
- Fiorenza, M., Kreutz, D., Mansilha, R., Macedo, D., Feitosa, E., and Immich, R. (2021b). Representação e aplicação de políticas de segurança em firewall de redes híbridas. In *XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.
- Haber, E. M. and Bailey, J. (2007). Design guidelines for system administration tools developed through ethnographic field studies. In *Proceedings of the 2007 Symposium on CHIMIT*, page 1–es. ACM.
- Manning, C. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.
- Manning, C. D. (2008). *Introduction to information retrieval*. Syngress Publishing,.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13*, page 3111–3119, Red Hook, NY, USA. Curran Associates Inc.
- Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26.
- Uc-Cetina, V., Navarro-Guerrero, N., and Martin-Gonzalez (2022). Survey on reinforcement learning for language processing. *Artificial Intelligence Review*.
- Voronkov, A., Martucci, L. A., and Lindskog, S. (2019). System administrators prefer command line interfaces, don't they? an exploratory study of firewall interfaces. In *Fifteenth SOUPS*. USENIX Association.