



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

DEPARTMENT OF COMPUTER SYSTEMS

**KLASIFIKACE ŠIFROVANÉHO PROVOZU V
POČÍTAČOVÝCH SÍTÍCH S VYUŽITÍM
TECHNOLIE FPGA**

THESIS TITLE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. LUKÁŠ NEVRKLA

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. TOMÁŠ MARTÍNEK, Ph.D.

BRNO 2026

Zadání diplomové práce



171232

Ústav: Ústav počítačových systémů (UPSY)
Student: **Nevrkla Lukáš, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Strojové učení
Název: **Klasifikace šifrovaného provozu v počítačových sítích s využitím technologie FPGA**
Kategorie: Návrh číslicových systémů
Akademický rok: 2025/26

Zadání:

1. Seznamte se s existujícími přístupy pro klasifikaci šifrovaného provozu v počítačových sítích. Zaměřte se na moderní metody založené na neuronových sítích.
2. Použijte existující nebo sestavte novou datovou sadu pro trénování a testování modelu strojového učení pro klasifikaci šifrovaného provozu.
3. Navrhněte model neuronové sítě pro klasifikaci šifrovaného provozu, který bude vhodně uzpůsoben pro implementaci v hardware, konkrétně v čípech s technologií FPGA.
4. Model natrénujte a ověřte přesnost jeho klasifikace s využitím připravené datové sady.
5. Vytvořený model následně ohodnoťte z pohledu implementace v hardware. Odhadněte potřebné množství výpočetních zdrojů, latenci a výkonnost klasifikace.
6. Diskutujte vlastnosti vytvořené implementace a možnosti dalšího pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Martínek Tomáš, doc. Ing., Ph.D.**

Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.

Datum zadání: 1.11.2025

Termín pro odevzdání: 20.5.2026

Datum schválení: 31.10.2025

Abstrakt

Protože šifrování síťového provozu znemožňuje některé úkony v kontextu správy sítí, vznikla oblast klasifikace šifrovaného provozu, která pomocí metod strojového učení analyzuje a klasifikuje provoz bez nutnosti jeho dešifrování. Specifickým případem je online klasifikace, během které jsou výsledky klasifikace využívány k aktivní manipulaci s jednotlivými síťovými toky (např. blokování, prioritizace či jiné mechanismy zajišťující kvalitu služeb). Nicméně provozování takových klasifikátorů na vysokorychlostních linkách klade vysoké nároky na latenci a propustnost celého řešení. Cílem této práce je navrhnout umělou neuronovou síť uzpůsobenou pro online klasifikaci provozu a akcelarovat ji v chytré síťové kartě vybavené FPGA. Konkrétně byla navržena rekurentní neuronová síť umožňující inkrementální klasifikaci síťových toků. Pro úsporu FPGA zdrojů byly navrženy optimalizace embedding tabulek a metody komprese stavové informace. Následně bylo využito vysoké konfigurovatelnosti FPGA pro implementaci akcelarátoru specializovaného pro tuto neuronovou síť a její parametry. Výsledné řešení překonává jedny z nejlepších dosavadních online klasifikátorů šifrovaného provozu (systémy N3IC, Kaleidoscope a FENIX) z hlediska přesnosti klasifikace, spotřeby FPGA zdrojů, latence, propustnosti i velikosti stavové informace.

Abstract

Network traffic encryption makes certain network management tasks infeasible, which has given rise to the field of encrypted traffic classification. This field employs machine learning methods to analyze and classify traffic without the need to decrypt it. A specific case is online classification, which uses the resulting decisions to actively manipulate individual flows (e.g., blocking, prioritization, or other mechanisms ensuring quality of service). However, deploying such classifiers on high-speed links poses a considerable challenge. The aim of this work is to design an artificial neural network tailored for online traffic classification and to accelerate it in a smart network interface card equipped with an FPGA. Specifically, a recurrent neural network enabling incremental classification of network flows was designed. To conserve FPGA resources, optimizations of embedding tables and methods for compressing state information were proposed. Subsequently, the high configurability of the FPGA was leveraged to implement an accelerator specialized for this neural network and its parameters. The resulting solution outperforms some of the best existing online classifiers of encrypted traffic (the N3IC, Kaleidoscope, and FENIX systems) in terms of classification accuracy, FPGA resource consumption, latency, throughput, and state information size.

Klíčová slova

Klasifikace šifrovaného provozu, rekurentní neuronové sítě, FPGA

Keywords

Encrypted traffic classification, recurrent neural networks, FPGA

Citace

NEVRKLA, Lukáš. *Klasifikace šifrovaného provozu v počítačových sítích s využitím*

technologie FPGA. Brno, 2026. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Tomáš Martínek, Ph.D.

Klasifikace šifrovaného provozu v počítačových sítích s využitím technologie FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana docenta Tomáše Martínka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Lukáš Nevřkla
18. května 2026

Poděkování

Tímto děkuji vedoucímu práce panu docentovi Tomášovi Martínkovi za pomoc, cenné rady a konzultace během vypracování této diplomové práce.

Obsah

1	Úvod	2
2	Neuronové sítě a jejich akcelerace na FPGA	4
2.1	Kvantizace a datové typy	4
2.2	Typická vrstva neuronové sítě	9
2.3	Kvantizace neuronových sítí	10
2.4	Způsoby výpočtu kvantizačních parametrů	15
2.5	FPGA akcelerátory neuronových sítí	17
3	Klasifikace šifrovaného provozu	19
3.1	Šifrování síťového provozu	19
3.2	Příznaky a datové sady	20
3.3	Offline klasifikace šifrovaného provozu	24
3.4	Online klasifikace šifrovaného provozu	26
4	Návrh neuronové sítě	30
4.1	Specifikace požadavků	30
4.2	Kritika současného stavu	31
4.3	Volba architektury	32
4.4	Volba experimentálního prostředí	33
4.5	Základní rekurentní modely	33
4.6	Předzpracování vstupních a výstupních příznaků	35
4.7	Vícevrstvé rekurentní buňky	41
4.8	Komprese stavu	45
4.9	Ohodnocení navržených modelů	48
5	Implementace FPGA akcelerátoru	52
5.1	Kvantizace navrženého modelu	52
5.2	FPGA implementace s využitím existujících nástrojů	55
5.3	Vlastní FPGA implementace	56
6	Závěr	60
	Literatura	62

Kapitola 1

Úvod

Šifrování síťového provozu sice chrání soukromí uživatelů, ale zároveň komplikuje monitorování a správu sítí. Úkony jako blokace nebezpečných služeb, prioritizace některých síťových toků nebo jiné metody zajišťování kvality služeb (QoS) jsou v důsledku šifrování provozu mnohem složitější.

Tradiční metody tyto úkony řešily pomocí analýzy obsahu paketů (tzv. *Deep Packet Inspection*). S příchodem šifrování obsahu to ale již není možné. V důsledku častých změn topologie internetu nejsou ani IP adresy spolehlivými indikátory konkrétních služeb. Navíc v případě CDN (*Content Delivery Networks*) může být na stejné IP adrese provozována široká škála služeb od různých poskytovatelů.

Bylo však zjištěno, že síťové toky vykazují charakteristické vzory, které lze rozpoznat i bez dešifrování obsahu paketů. Například pouhou analýzou velikostí paketů a jejich časových rozestupů lze rozpoznat konkrétní webovou aplikaci, která příslušný tok vytvořila. Vznikla tedy oblast nazvaná *klasifikace šifrovaného provozu*, která pomocí metod strojového učení dokáže ze zašifrovaných toků získávat informace klíčové pro správu sítí.

Specifickým případem je tzv. *online klasifikace*, při které jsou výsledky klasifikace použity v reálném čase pro přímou manipulaci s toky (např. blokace nebo prioritizace). Vytvoření klasifikátoru, který by bylo možné nasadit na vysokorychlostních linkách (např. 100 Gb/s a více), však naráží na striktní požadavky z pohledu propustnosti a latence. Jednou z možností, jak tyto požadavky splnit, je umístit klasifikační model přímo na chytrou síťovou kartu vybavenou FPGA.

Cílem této práce je navrhnout umělou neuronovou síť uzpůsobenou pro online klasifikaci provozu a akcelarovat ji v FPGA. Konkrétně byla navržena rekurentní neuronová síť, která umožňuje inkrementální klasifikaci přicházejících paketů. Parametry i aktivace navržené sítě byly úspěšně kvantizovány do 4 bitů za cenu minimálního poklesu přesnosti klasifikace (F1 skóre kleslo pouze o 0,61 % až o 2,5 %).

Navržený model byl následně implementován v FPGA. Vysoká konfigurovatelnost FPGA umožnila implementovat obvody specializované přímo pro konkrétní architekturu a dokonce i pro konkrétní parametry modelu. To umožnilo překonat existující metody online klasifikace (konkrétně systémy N3IC [33], Kaleidoscope [37] a FENIX [10]) z pohledu přesnosti klasifikace, propustnosti, spotřeby FPGA zdrojů i z pohledu latence. Například při srovnání se systémem FENIX měl nový model o 4,2 % vyšší F1 skóre klasifikace (93,4 %), spotřeboval 9× méně LUT a měl 3× nižší latenci (83 ns).

Existující způsoby akcelerace neuronových sítí na FPGA jsou nejprve popsány v kapitole 2. Následující kapitola 3 je věnována klasifikaci šifrovaného provozu, existujícím datovým sadám a přehledu existujících offline i online metod klasifikace. Návrh nové neuronové

sítě pro online klasifikaci provozu je popsán v kapitole 4. V závěrečné kapitole 5 je tato síť kvantizována, implementována v FPGA a její vlastnosti jsou porovnány s existujícími metodami online klasifikace.

Kapitola 2

Neuronové sítě a jejich akcelerace na FPGA

V kontextu neuronových sítí je možné akcelarovat buď trénování, nebo *inferenci* (nasazení již natrénovaného modelu). Obě fáze se liší v požadavcích na flexibilitu a latenci akcelérátoru. Během trénování je důležitá schopnost rychlého nasazení nových variant neuronových sítí, a tedy i flexibilita příslušného akcelérátoru. Během inference je model fixní a je důležitější vyšší propustnost a nižší latence.

GPU jsou z mnoha hledisek výhodné pro trénování i inferenci modelů strojového učení, neboť nabízí tisíce výpočetních jednotek schopných efektivního vykonávání tenzorových operací (*tenzorová jádra*) nebo speciálních matematických funkcí (*Special Function Units*). Dále nabízí dedikované jednotky pro efektivní práci s pamětí (*Load Store* jednotky) a několik vyrovnávacích pamětí (od obecných L1 a L2 cache až po specializované texturní paměti). Výrobci GPU karet také nabízejí rozsáhlou sadu efektivních knihoven a většina frameworků z oblasti strojového učení má pro GPU nativní podporu.

FPGA nabízejí oproti GPU dvě výhody, které jsou v některých aplikacích klíčové: (i) možnost navržení výpočetních jednotek optimalizovaných přímo pro konkrétní architekturu modelu, pro specifický způsob kvantizace nebo i pro konkrétní parametry modelu, (ii) dosažení velmi nízké latence v případech, kdy je výsledek klasifikace použitý přímo ve stejném FPGA čipu. Navíc oproti fixním ASIC čipům jsou FPGA rekonfigurovatelné, což umožňuje kdykoliv změnit parametry modelu nebo i jeho architekturu. Těchto vlastností je využito v následujících kapitolách této práce pro akceleraci online klasifikátorů šifrovaného provozu.

Obsah této kapitoly je zaměřený na obecné koncepty, které je možné využít pro efektivní akceleraci neuronových sítí na FPGA. V úvodní sekci 2.1 jsou představeny datové typy vhodné pro kvantizaci. Následuje popis typické vrstvy neuronové sítě (sekce 2.2) a různých úrovní kvantizace neuronových sítí (sekce 2.3). Způsob výpočtu kvantizačních parametrů je uveden v sekci 2.4. Poslední sekce 2.5 shrnuje existující FPGA akcelérátory neuronových sítí.

2.1 Kvantizace a datové typy

Kvantizace je způsob aproximace spojitě domény (např. reálná čísla $\mathbb{R}$) pomocí diskrétní množiny hodnot (tzv. *kvantizační hladiny*). Protože datové typy používané v digitálních systémech umožňují reprezentovat pouze konečnou množinu hodnot, lze jakékoliv vyjád-

ření neuronových sítí v těchto systémech považovat za kvantizované. Ve většině případů poskytují datové typy s plovoucí desetinnou čárkou (dále jen *FP* z angl. *Floating Point*) dostatečné množství úrovní pro přesnou aproximaci spojitě domény¹.

Z dosavadního výzkumu dokonce vyplývá, že pro řadu modelů je možné použít datové typy s mnohem menší bitovou šířkou než 32 bitů [11]. V literatuře je teprve tato konverze k úspornějším datovým typům označována jako *kvantizace neuronových sítí*.

Snížení bitové šířky operandů umožňuje redukovat spotřebu hardwarových zdrojů příslušných výpočetních elementů (viz tabulka 2.1). Například součin dvou 8 bitových celých čísel lze realizovat na přibližně 12× menší ploše čipu než v případě součinu dvou 32 bitových celých čísel.

Tabulka 2.1: Srovnání energetické náročnosti a plochy čipu operace sčítání a násobení pro různé bitové šířky operandů u technologie 45nm. Převzato z [11].

Datové typy operandů	Energie [<i>pJ</i>]	Plocha [μm^2]
Součet		
INT 8b	0,03	36
INT 16b	0,05	67
INT 32b	0,1	137
FP 16b	0,4	1360
FP 32b	0,9	4184
Součin		
INT 8b	0,2	282
INT 32b	3,1	3495
FP 16b	1,1	1640
FP 32b	3,7	7700

Při snížení bitové šířky také dochází k exponenciálnímu poklesu počtu kvantizačních hladin, a tím i ke zvýšení kvantizačních chyb. Aby tedy nedošlo k výrazné degradaci přesnosti modelu, je nutné zajistit efektivní využití dostupné bitové šířky. Proto je vhodné se zabývat datovými typy a způsoby, kterými tuto bitovou šířku využívají.

Datové typy s plovoucí desetinnou čárkou jsou výhodné během trénování neuronových sítí, při kterém se rozsahy hodnot často mění. Konkrétní rozsah je totiž zvolen dynamicky tak, aby bylo možné danou hodnotu reprezentovat s co nejvyšší přesností. Výpočetní elementy musí spravovat tzv. *exponent*, který určuje aktuálně zvolený rozsah, a tzv. *mantisu*, která určuje pozici hodnoty uvnitř tohoto rozsahu.

Ukazuje se, že pro neuronové sítě je výhodnější zvolit jiný poměr mezi bitovou šířkou exponentu a šířkou mantisy než v případě datových typů ze standardu IEEE 754. Zatímco standardní typ *FP16* z IEEE 754 dedikuje 5 bitů pro exponent, datový typ *BF16* (*Brain Float*), specializovaný pro neuronové sítě, dedikuje 8 bitů pro exponent. V některých případech jsou pro trénování i inferenci neuronových sítí používány i menší bitové šířky jako jsou 8 bitové typy *FP8* [2], nebo 4 bitové typy *FP4* [5].

Maximální reprezentovatelný rozsah je u *FP8* a *FP4* typů velmi omezený. Např. *FP4* umožňuje reprezentovat maximálně hodnoty v rozsahu ± 6 [7]. Proto bývají používány do-

¹ *FP32* ze standardu IEEE 754 má teoreticky 2^{32} různých hladin, ale některé z nich reprezentují speciální nečíselné stavy (např. *NaN* a *inf*). Ale i po odečtení speciálních nečíselných stavů má *FP32* stále přes 4 miliardy různých hladin.

datečné techniky jako např. *microscaling*, při kterém skupina *FP8* nebo *FP4* hodnot sdílí dodatečný exponent s větší bitovou šířkou. Tento exponent je opět vypočten dynamicky tak, aby bylo možné hodnoty reprezentovat s co nejvyšší přesností. Protože ale přísluší celé skupině hodnot (ne každé hodnotě zvlášť), je dodatečná režie nízká. Nejnovější GPU karty již nabízejí hardwarovou podporu pro tyto speciální datové typy.

Všechny FP typy přinášejí v důsledku správy exponentu režii navíc. Např. pro sečtení nebo odečtení dvou hodnot je nutné nejprve zarovnat jejich exponenty pomocí odpovídajícího posunu mantisy. Takové zarovnání zvyšuje množství dodatečných hardwarových zdrojů a latenci příslušných operací. Jak ukazuje předchozí tabulka 2.1, sčítání ve formátu FP32 vyžaduje přibližně 30× větší plochu čipu než sčítání celých čísel při stejné bitové šířce.

Protože jsou rozsahy hodnot během inference stabilnější než během trénování, je možné použít datový typ s pevnou desetinnou čárkou (dále jen *FX* z angl. *Fixed-Point*). V tomto případě je exponent E během inference fixní celočíselná hodnota, čímž je možné ušetřit netriviální množství hardwarových zdrojů. Akcelerátor tedy pracuje pouze s celočíselným posunem q_x uvnitř rozsahu zvoleného během návrhu akcelerátoru.

Pro konverzi spojitě hodnoty do kvantizované reprezentace je použit vztah 2.1 (tzv. kvantizace) a pro opačnou konverzi je použit vztah 2.2 (tzv. dekvantizace)². Funkce $\text{round}(x)$ označuje libovolnou metodu zaokrouhlování a funkce $\text{clip}(x, x_{\min}, x_{\max})$ označuje ořez hodnoty x do rozsahu $\langle x_{\min}, x_{\max} \rangle$. Symbol q_x je v následujících částech práce používán pro označení kvantizované varianty libovolné proměnné x . Symbol b_x je použit pro označení bitové šířky proměnné x . Všechny dodatečné parametry, které jsou potřebné pro správnou interpretaci daného datového typu, budou dále označovány jako tzv. *kvantizační parametry*. V případě FX je jediným kvantizačním parametrem exponent E .

$$q_x = \mathcal{Q}_x(x) = \text{clip}(\text{round}(2^E \cdot x), q_{x_{\min}}, q_{x_{\max}}) \quad (2.1)$$

$$x \approx \mathcal{D}_x(q_x) = 2^{-E} \cdot q_x \quad (2.2)$$

Obecnějším přístupem je tzv. *celočíselná asymetrická (afinní) kvantizace*. Pro kvantizaci a dekvantizaci hodnoty x jsou v tomto případě použity obecnější vztahy 2.3 a 2.4. Zatímco u FX byl multiplikativní faktor omezen pouze na mocniny dvou (2^E), u afinní kvantizace je použit libovolný spojitý škálovací faktor S . Navíc je přidán posun Z , který umožňuje kvantizační rozsah posunout mimo nulový bod. Tímto zobecněním je možné kvantizační rozsah zvolit přesněji pro konkrétní veličinu, a tím i zajistit efektivnější využití dostupné bitové šířky. Je vhodné poznamenat, že škálovací faktor S bývá v praxi vyjadřován pomocí FP typů, zatímco posun Z je možné vyjádřit celočíselným datovým typem. V případě, kdy je člen Z nulový, se jedná o tzv. *symetrickou kvantizaci*, pomocí které lze reprezentovat pouze rozsahy souměrné kolem nuly.

$$q_x = \mathcal{Q}_x(x) = \text{clip}\left(\text{round}\left(\frac{x}{S} + Z\right), q_{x_{\min}}, q_{x_{\max}}\right) \quad (2.3)$$

$$x \approx \mathcal{D}_x(q_x) = S \cdot (q_x - Z) \quad (2.4)$$

Pro výpočet škálovacího faktoru S a posunu Z je nejprve nutné zvolit vhodný kvantizační rozsah $\langle x_{\min}, x_{\max} \rangle$ a bitovou šířku b_x . Konkrétní způsoby, kterými je tento rozsah

²V případě dekvantizace je využit symbol $\approx$ místo $=$, protože během kvantizace byly aplikovány operace *round* a *clip*, které způsobily vznik kvantizačních chyb.

zvolen, jsou popsány v sekci 2.4. Poté lze škálovací faktor S vypočíst podle vztahu 2.5 a posun Z podle vztahu 2.6 [11].

$$S = \frac{x_{max} - x_{min}}{q_{x_{max}} - q_{x_{min}}} = \frac{x_{max} - x_{min}}{2^{b_x} - 1} \quad (2.5)$$

$$Z = \text{clip} \left(\text{round} \left(q_{x_{min}} - \frac{x_{min}}{S} \right), q_{x_{min}}, q_{x_{max}} \right) \quad (2.6)$$

Extrémním příkladem redukce bitové šířky je binární kvantizace [11]. V tomto případě je použit pouze jednobitový datový typ, který umožňuje reprezentovat pouze informaci o znaménku (tedy hodnoty -1 a $+1$). Násobení dvou binárních hodnot lze realizovat pouze pomocí operace XNOR (viz tabulka 2.2). K sečtení binárních hodnot je použita operace *popcount*, která sečte počet jedniček v binárním vektoru. Lineární vrstvy binárních neuronových sítí lze tedy vyhodnotit pouze pomocí jednoduchých operací XNOR a *popcount*.

Alternativou je ternární kvantizace, která navíc umožňuje reprezentovat i hodnotu 0. Pro ternární kvantizaci je používán dvoubitový datový typ, přičemž jeden stav není využit. Dále je vhodné poznamenat, že binární, ternární i FX datové typy lze chápat jako podmnožinu celočíselné asymetrické kvantizace. Přehled zmíněných datových typů je uveden v tabulce 2.3.

Tabulka 2.2: Logická tabulka pro XNOR obvod reprezentující násobení dvou binárních vstupů.

x_1	x_2	$x_1 \cdot x_2$	q_{x_1}	q_{x_2}	XNOR(q_{x_1}, q_{x_2})
1	1	1	1	1	1
1	-1	-1	1	0	0
-1	1	-1	0	1	0
-1	-1	1	0	0	1

Doposud zmíněné datové typy je možné zařadit do kategorie *uniformních datových typů*, neboť všechny kvantizační hladiny jsou rozloženy rovnoměrně v rámci zvoleného rozsahu. Efektivní využití dostupné bitové šířky spočívá na správné volbě tohoto rozsahu. Jeho rozšířením dojde k oddálení kvantizačních úrovní, což povede ke zvětšení kvantizačních chyb. Naopak při zúžení rozsahu může dojít k ořezu některých hodnot.

Existují však i práce využívající tzv. *neuniformní kvantizaci*, při které je možné kvantizační hladiny rozmístit nerovnoměrně. Umístění hladin lze přesněji přizpůsobit konkrétnímu rozdělení hodnot, což umožňuje dále snížit kvantizační chyby.

U některých metod neuniformní kvantizace je rozmístění kvantizačních hladin fixní. Např. práce [25] využívá exponenciální rozestupy hladin, což znamená, že výpočetní elementy pracují s příslušnými hodnotami v logaritmické doméně (viz rovnice 2.7). To umožňuje násobení nahradit prostým součtem, avšak původní operace sčítání se tím stávají problematické. Jiné metody neuniformní kvantizace přímo trénují umístění kvantizačních hladin spolu s ostatními parametry modelu (např. práce *nuLSQ* [13]).

$$q_x = \mathcal{Q}_x(\log_2(x)) \quad (2.7)$$

Zatímco akcelerace neuniformně kvantizovaných modelů může být na GPU problematická, v případě FPGA je možné pro tento účel navrhnout specifické výpočetní elementy. Nicméně většina nástrojů pro kvantizaci neuronových sítí a jejich implementaci v FPGA tyto datové typy zatím nepodporuje.

Tabulka 2.3: Přehled základních datových typů používaných pro kvantizaci neuronových sítí. Sloupec *Formát* uvádí hodnoty, ze kterých se daný datový typ skládá. Datové typy s plovoucí desetinnou čárkou (FP) jsou složeny ze znaménka S , exponentu E a mantisy M . Hodnota uvedená za S, E, M označuje počet bitů dedikovaných pro příslušnou část. V případě ostatních datových typů je použita jediná celočíselná hodnota q_x . Pro správnou interpretaci některých datových typů je nutná dodatečná informace, která je uvedena ve sloupci *Parametry* (tzv. *kvantizační parametry*). Sloupec *Dekvantizace* uvádí způsob, kterým je možné kvantizovanou hodnotu interpretovat ve spojitě doméně.

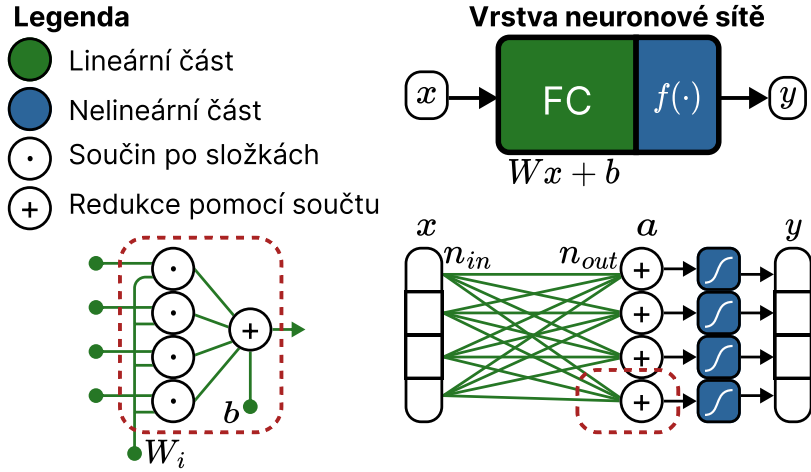
Datový typ	Formát	Parametry	Dekvantizace
Plovoucí řadová čárka	S, E, M		$(-1)^S \left(1 + \frac{M}{2^{b_M}}\right) \cdot 2^{E-\text{bias}}$ kde: $\text{bias} = 2^{b_E-1} - 1$
FP32 (IEEE 754)	$S1, E8, M23$		
FP16 (IEEE 754)	$S1, E5, M10$		
BF16 (Brain float)	$S1, E8, M7$		
FP8 E4M3	$S1, E4, M3$		
FP8 E5M2	$S1, E5, M2$		
MXFP4 / NVFP4	$S1, E2, M1$		
Pevná řadová čárka	q_x	E	$2^{-E} \cdot q_x$
Celočíselný typ			
Symetrický	q_x	S	$S \cdot q_x$
Asymetrický	q_x	S, Z	$S \cdot (q_x - Z)$
Speciální typy			
Binární	$q_x \in \{0, 1\}$		$(-1)^{q_x}$
Ternární	$q_x \in \{0, 1, 2\}$		$q_x - 1$

2.2 Typická vrstva neuronové sítě

Typická vrstva neuronové sítě se skládá z lineární části a z nelineární aktivační funkce $f(a)$ (viz schématické znázornění na obrázku 2.1). Lineární vrstva obsahuje ve většině případů všechny učitelné parametry modelu (matice vah W a vektor prahů b ³) a zároveň představuje výpočetně nejnáročnější část modelu. V této práci bude použita převážně tzv. plně propojená vrstva (*FC* z angl. *fully-connected layer*), která je definována vztahem 2.8. Člen n_{in} reprezentuje počet vstupů a n_{out} počet výstupů.

$$a = Wx + b$$

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n_{out}} \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n_{in}} \\ w_{21} & w_{22} & \cdots & w_{2n_{in}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_{out}1} & w_{n_{out}2} & \cdots & w_{n_{out}n_{in}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_{in}} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n_{out}} \end{pmatrix} \quad (2.8)$$



Obrázek 2.1: Schématické znázornění typické vrstvy neuronové sítě. Zelená barva označuje lineární část a modrá barva nelineární aktivační funkci. Proměnná x značí vstupní vektor a proměnná y označuje výstupní vektor. Proměnná a představuje výstup lineární části (tzv. *akumulátor*). Na obrázku je konkrétně znázorněna plně propojená lineární vrstva.

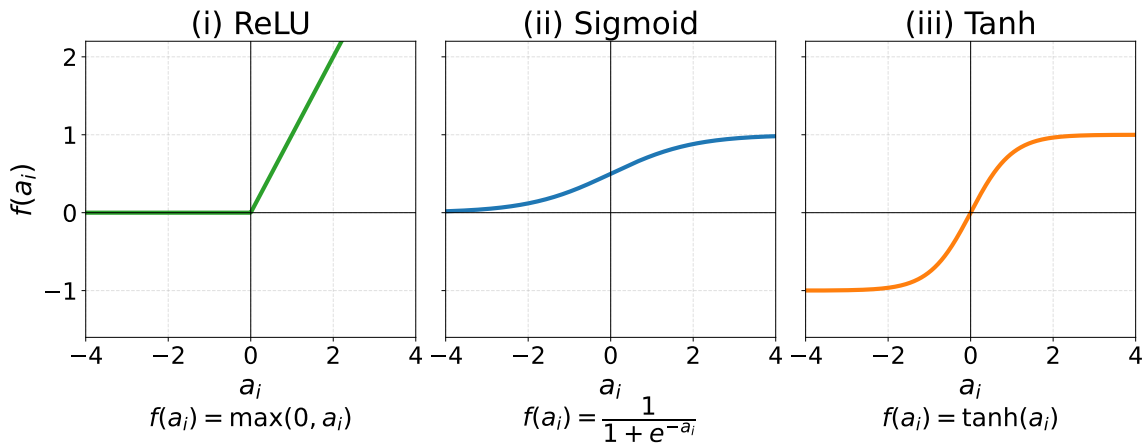
Každý propoj *FC* vrstvy reprezentuje součin jednoho vstupu a jedné váhy ($w_{ij} \cdot x_j$). Výstupem této vrstvy je tzv. *akumulátor* a , jehož jednotlivé složky a_i jsou získány sečtením příslušných součinů a prahu b_i . Aby se zabránilo přetečení akumulátoru, je jeho bitová šířka větší než bitová šířka vstupů a vah. Pokud bitovou šířku vstupu označíme b_x a bitovou šířku vah b_w , pak lze bitovou šířku výstupního akumulátoru určit pomocí vztahu 2.9. Člen $b_x + b_w$ odpovídá bitové šířce výsledku násobení jednoho vstupu a jedné váhy. Člen $\lceil \log_2(n_{in}) \rceil$ zabráňuje přetečení během akumulace n_{in} součinů. Poslední člen $+1$ je přidán kvůli přičtení prahu b .

$$b_a = b_x + b_w + \lceil \log_2(n_{in}) \rceil + 1 \quad (2.9)$$

³Pokud jsou prahy b nenulové, pak se správně jedná o tzv. *afinní* transformaci. Nicméně pro shodu s běžnou terminologií v literatuře bude i zde použito označení lineární transformace i v případě nenulových prahů b .

Nelineární část $f(a)$ lze ve většině případů vyjádřit pomocí funkcí f aplikovaných na jednotlivé složky akumulátoru a_i nezávisle: $y_i = f(a_i)$. Jejich výpočet tedy lze snadno paralelizovat. Příklady těchto funkcí jsou zobrazeny na obrázku 2.2. Výjimkou je tzv. *softmax* funkce, která slouží k normalizaci vstupního vektoru (viz rovnice 2.10). Jelikož je součet všech výstupních hodnot softmax funkce roven právě jedné, lze tyto hodnoty interpretovat jako pravděpodobnosti. Akcelerace této funkce je výrazně složitější než v případě ostatních nelineárních funkcí.

$$\text{softmax}(a_i) = \frac{e^{a_i}}{\sum_{j=1}^C e^{a_j}} \quad \text{pro } i = 1, 2, \dots, C \quad (2.10)$$



Obrázek 2.2: Základní nelineární aktivační funkce $f(a_i)$ používané v neuronových sítích: (i) *ReLU*, (ii) *sigmoid* a (iii) *tanh*.

2.3 Kvantizace neuronových sítí

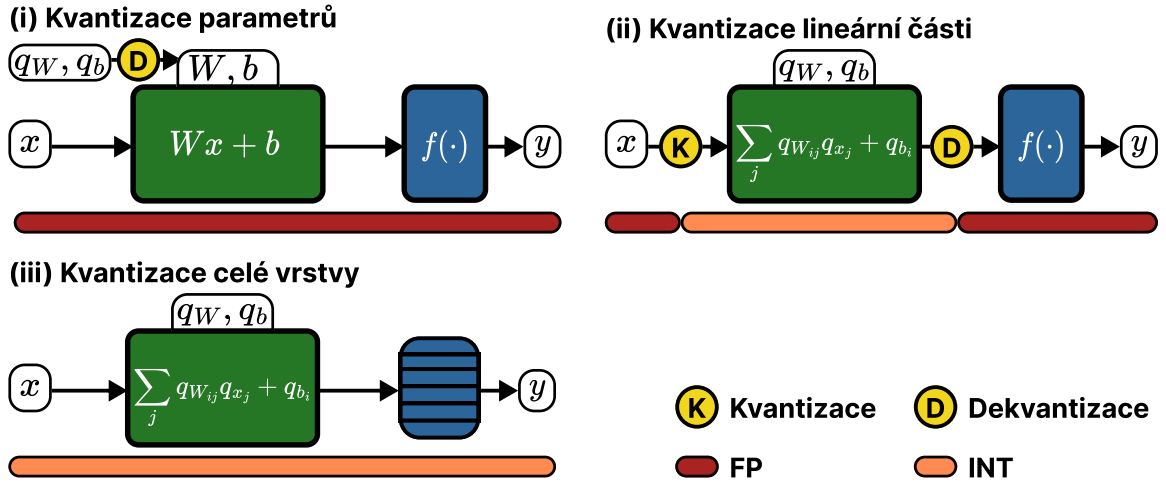
V neuronové síti je možné kvantizovat dva druhy hodnot: parametry a aktivace⁴. Parametry jsou během inference konstantní, ale aktivace závisí na vstupních datech. Kvantizace aktivací je tedy komplikovanější. V literatuře se objevují 3 stupně kvantizace neuronových sítí: (i) kvantizace pouze parametrů, (ii) kvantizace parametrů i aktivací, ale pouze v lineárních vrstvách a (iii) kvantizace všech parametrů i aktivací v celém modelu (viz obrázek 2.3).

Stupeň (i) je z pohledu implementace nejjednodušší a nezpůsobí výraznou degradaci přesnosti modelu. Tento směr lze chápat jako formu komprese modelu. Je výhodný v případech, kdy je akcelerace neuronové sítě limitována propustností paměti, nebo pokud se původní model nevejde do paměti akcelérátoru.

Kvantizační stupeň (ii)

V běžných neuronových sítích představují lineární vrstvy výpočetně nejnáročnější část. Proto je cílem stupně (ii) kvantizovat celé lineární vrstvy a jejich inferenci realizovat pouze pomocí kvantizovaných datových typů (zejména pomocí celočíselné aritmetiky). Aby bylo

⁴Pojem aktivace se v literatuře používá pro označení buď výstupů z nelineárních aktivačních funkcí, nebo všech mezivýsledků, které vznikly během výpočtu neuronové sítě. V této práci je použita druhá varianta (pojem aktivace tedy označuje všechny mezivýsledky).



Obrázek 2.3: Srovnání 3 stupňů kvantizace neuronových sítí. V případě (i) jsou kvantizovány pouze parametry a vyhodnocení modelu probíhá v datových typech s plovoucí desetinnou čárkou. U varianty (ii) jsou pouze lineární vrstvy vyhodnocovány pomocí kvantizovaných datových typů. V případě (iii) je celý model vyhodnocován v kvantizované reprezentaci. Stupeň (iii) vyžaduje zavést určité aproximace nelineárních aktivačních funkcí. Může se jednat např. o vyhledávací tabulku, která je znázorněna v obrázku.

možné porozumět návrhovým rozhodnutím, která jsou použita pro (ii) stupeň kvantizace, je vhodné představit matematickou podstatu kvantizace lineárních vrstev.

Předpokládejme plně propojenou lineární vrstvu $a = Wx + b$ a uniformní celočíselnou kvantizaci. Každou spojitou hodnotu v je možné pomocí kvantizačních parametrů S_v a Z_v (škálovací faktor a posun) rozložit následovně: $v \approx S_v \cdot (q_v - Z_v)$ (viz vzorec pro dekvantizaci 2.4). Cílem je přeuspořádat spojitě škálovací faktory tak, aby bylo možné během inference pracovat pouze s celočíselnými variantami původních hodnot. Je vhodné poznamenat, že posun Z_v je již celočíselná hodnota.

Uvažujme nyní součin jedné vstupní hodnoty x_i a jedné váhy W_{ij} . Při vytknutí škálovacích faktorů S_{x_i} a $S_{W_{ij}}$ je možné tento součin realizovat pouze pomocí celočíselné aritmetiky (viz $q_{x_i} \cdot q_{W_{ij}}$ v rovnici 2.11). Nově vzniklý škálovací faktor $(S_{x_i} \cdot S_{W_{ij}})$ je stále spojitý, nicméně pokud je očekáván pouze celočíselný výstup, lze tento dekvantizační člen vypustit.

Kvůli nenulovým posunům $Z_{x_i}, Z_{W_{ij}}$ vznikají po roznásobení dodatečné členy, které mohou komplikovat návrh akcelérátoru. Člen $-Z_{x_i}q_{W_{ij}} + Z_{x_i}Z_{W_{ij}}$ není závislý na vstupním vektoru x a může být připočítán k prahům b lineární vrstvy. Problematický může být člen $-q_{x_i}Z_{W_{ij}}$, který je závislý na vstupu a vyžaduje dodatečné výpočetní zdroje. Ve většině případů je dostačující symetrická kvantizace, pro kterou je možné součin vyjádřit jednodušší rovnicí 2.12 [39].

$$\begin{aligned}
 x_i \cdot W_{ij} &\approx \underbrace{(S_{x_i} \cdot (q_{x_i} - Z_{x_i}))}_{x_i} \cdot \underbrace{(S_{W_{ij}} \cdot (q_{W_{ij}} - Z_{W_{ij}}))}_{W_{ij}} \\
 &= \underbrace{(S_{x_i} \cdot S_{W_{ij}})}_{\text{float}} \cdot \underbrace{(q_{x_i}q_{W_{ij}} - q_{x_i}Z_{W_{ij}} - Z_{x_i}q_{W_{ij}} + Z_{x_i}Z_{W_{ij}})}_{\text{int}} \quad (2.11)
 \end{aligned}$$

$$\begin{aligned}
x_i \cdot W_{ij} &\approx \underbrace{(S_{x_i} \cdot q_{x_i})}_{x_i} \cdot \underbrace{(S_{W_{ij}} \cdot q_{W_{ij}})}_{W_{ij}} \\
&= \underbrace{(S_{x_i} \cdot S_{W_{ij}})}_{\text{float}} \cdot \underbrace{(q_{x_i} q_{W_{ij}})}_{\text{int}}
\end{aligned} \tag{2.12}$$

Pro získání konečného výstupu lineární vrstvy je nutné sečíst všechny mezivýsledné součiny $x_i \cdot W_{ij}$ a práh b . Vytknutí škálovacích faktorů je v případě součtu možné pouze, pokud jsou škálovací faktory všech operandů shodné (viz rovnice 2.13). Aby tedy bylo možné vyhodnotit lineární vrstvu pouze pomocí celočíselné aritmetiky, musí určité skupiny veličin sdílet stejné škálovací faktory. Jemnost, se kterou jsou škálovací faktory sdíleny, se v literatuře označuje jako *granularita kvantizace*. Vyšší granularita znamená, že více veličin sdílí stejný kvantizační rozsah. Pokud mají tyto veličiny rozdílné rozsahy hodnot, pak bude nutné rozšířit kvantizační rozsah a tím i snížit přesnost uvnitř tohoto rozsahu.

$$a + b \approx \underbrace{(S \cdot q_a)}_a + \underbrace{(S \cdot q_b)}_b = S \cdot (q_a + q_b) \tag{2.13}$$

Nejjemnější granularita, při které je možné škálovací faktory vytknout, je popsána omezeními v rovnici 2.14. Celý vstupní vektor x sdílí stejný škálovací faktor S_x , celý řádek matice vah W_i sdílí stejný škálovací faktor S_{W_i} a škálovací faktor prahů b_i musí být roven $S_{W_i} \cdot S_x$. V literatuře je často používána hrubší úroveň granularity, ve které má celá matice vah jeden sdílený faktor S_W (tzv. *per-tensor* granularita) [39].

$$\begin{aligned}
a_i &= \sum_j W_{ij} x_j + b_i \\
&\approx \sum_j \underbrace{(S_{W_{ij}} \cdot q_{W_{ij}})}_{W_{ij}} \cdot \underbrace{(S_x \cdot q_{x_j})}_{x_j} + \underbrace{(S_{b_i} \cdot q_{b_i})}_{b_i}
\end{aligned} \tag{2.14}$$

pro omezení: $S_{W_{ij}} = S_{W_i} \forall j, S_{x_j} = S_x \forall j, S_{b_i} = S_{W_i} \cdot S_x$

$$= \underbrace{(S_{W_i} \cdot S_x)}_{\text{float}} \cdot \underbrace{\left(\sum_j q_{W_{ij}} q_{x_j} + q_{b_i} \right)}_{\text{int}}$$

Výše uvedeným omezením se lze vyhnout použitím FX datových typů. Škálovací faktory jsou v tomto případě omezeny pouze na mocniny dvou $S_v = 2^{E_v}$, a proto lze přechod mezi různými kvantizačními rozsahy realizovat jen pomocí bitových posunů. Díky tomu je možné lineární vrstvu vyhodnotit pomocí celočíselné aritmetiky i v případě, kdy má každá hodnota vlastní kvantizační rozsah. Toho využívá např. nástroj *HGQ* (*High Granularity Quantization* [35]), který umožňuje nastavit vlastní kvantizační rozsahy a vlastní bitové šířky pro každou dílčí váhu i aktivaci. Omezení škálovacích faktorů na mocniny dvou v případě FX typů však může vést k horšímu využití dostupné bitové šířky. Navíc v případě sčítání dvou hodnot s výrazně odlišnými exponenty E musí dojít k rozšíření bitové šířky jednoho z operandů. To může vyžadovat dodatečné výpočetní zdroje.

Lineární operace je tedy možné za určitých podmínek realizovat pomocí celočíselné aritmetiky. Nedílnou součástí většiny vrstev je i nelineární aktivační funkce f . Právě skrze tyto nelinearity není možné škálovací faktory vytknout: $f(S \cdot q) \neq S \cdot f(q)$ [1, Definice 3.2]. Proto jsou nelineární aktivační funkce v případě kvantizačního stupně (ii) vyhodnocovány pomocí FP datových typů. Před lineární vrstvy jsou vloženy kvantizační operace a za lineární vrstvy jsou vloženy dekvantizační operace.

Kvantizační stupeň (iii)

U zařízení s omezenou podporou FP typů (zejména FPGA nebo mikrokontroléry) je používán (iii) stupeň kvantizace, který umožňuje celý model vyhodnotit pouze pomocí celočíselných operací. V takovém případě je ale nutné zavést určité aproximace nelineárních aktivačních funkcí, které umožní jejich implementaci bez použití FP aritmetiky. V FPGA je možné nelineární aktivační funkce realizovat následujícími způsoby: (i) použitím jednoduchých aktivačních funkcí (např. *ReLU*, *hard-tanh* nebo *hard-sigmoid*), (ii) pomocí předpočítaných tabulek (použito v nástroji *HLS4ML* [30]) nebo (iii) pomocí tzv. *prahovací* funkce (angl. *thresholding function*).

Řada modelů již používá jednoduché aktivační funkce. Např. *ReLU*, které lze implementovat pouze pomocí jednoho komparátoru a jednoho multiplexeru (viz obrázek 2.4(i)). Jiné neuronové sítě, jako např. rekurentní síť *GRU* nebo *LSTM*, používají složitější aktivační funkce *sigmoid* nebo *tanh*. Tyto funkce je možné aproximovat tzv. hard variantami (*hard-sigmoid* nebo *hard-tanh*), které je možné implementovat podobně jako *ReLU*. Nicméně, jedná se pouze o hrubou aproximaci původních funkcí, která může způsobit výraznou degradaci přesnosti modelu.

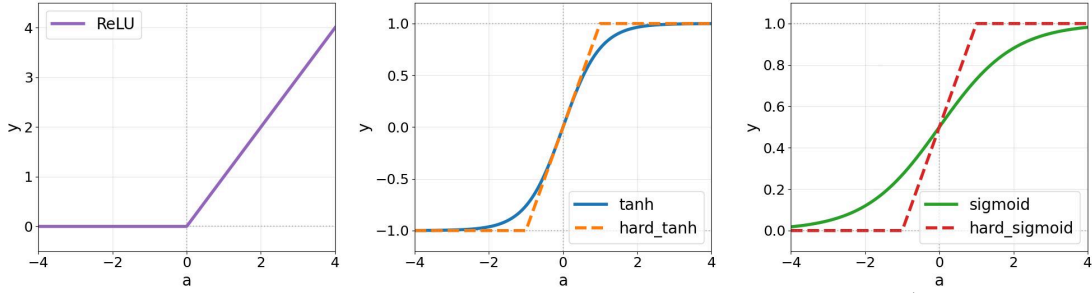
Přístup (ii) je založený na předpočítaných tabulkách, pomocí kterých lze implementovat i výrazně složitější nelineární funkce (viz obrázek 2.4(ii)). Nicméně velikost těchto tabulek roste exponenciálně s bitovou šířkou vstupů do nelineární funkce. To je nevýhodné, neboť vstupy těchto funkcí jsou akumulátory lineárních vrstev, které mají výrazně větší bitové šířky než parametry modelu. Proto nástroj *HLS4ML* umožňuje ořezat několik nejméně významných bitů akumulátorů, a tím i redukovat velikost předpočítaných tabulek. To ale povede ke snížení přesnosti aproximace.

Prahovací funkce $T(a, t)$ použitá v přístupu (iii) je definována rovnicí 2.15 [36]. Funkce porovnává vstupní akumulátor a s $n - 1$ prahy $t_1 < t_2 < \dots < t_{n-1}$, které rozdělují vstupní kvantizační rozsah do n intervalů. Každý z těchto intervalů odpovídá právě jedné výstupní kvantizační úrovni (viz obrázek 2.4(iii)). Kvantizovaný výstup $q_y \in \{0, 1, \dots, n-1\}$ odpovídá počtu prahů menších než akumulátor a .

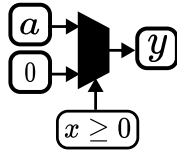
$$T(a, t) = \begin{cases} 0 & \text{pro } a \leq t_1 \\ 1 & \text{pro } t_1 < a \leq t_2 \\ \dots & \dots \\ n-2 & \text{pro } t_{n-2} < a \leq t_{n-1} \\ n-1 & \text{pro } t_{n-1} < a \end{cases} \quad (2.15)$$

Vhodným nastavením prahů t je možné realizovat monotonní lineární i nelineární funkce. Musí se ale vždy jednat o funkce, které jsou na vstupní vektor aplikované po složkách. Prahovací funkce tedy nemůže nahradit např. celou plně propojenou lineární vrstvu. Konkrétně lze prahovací funkci použít pro: (i) aproximaci nelineárních aktivačních funkcí, (ii) rekvantizaci do nového kvantizačního rozsahu a (iii) pohlcení spojitých škálovacích faktorů.

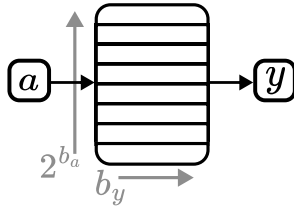
(i) Jednoduché aktivační funkce



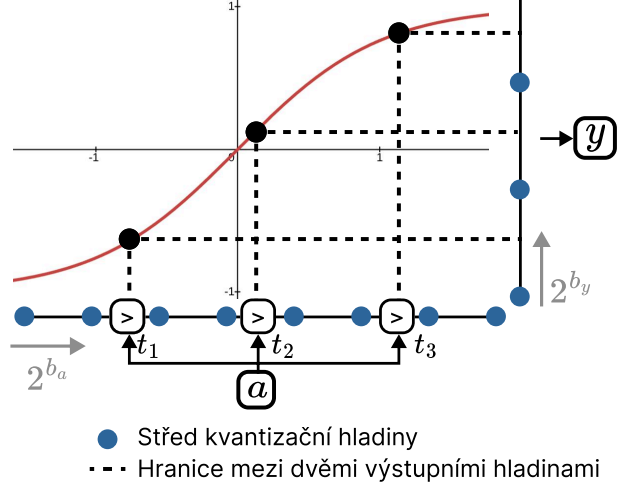
$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad \text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases} \quad \text{HardSigmoid}(x) = \begin{cases} 0 & \text{if } x < -1 \\ \frac{x+1}{2} & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$



(ii) Předpočítaná tabulka



(iii) Thresholding funkce



Obrázek 2.4: Základní způsoby implementace nelineárních aktivačních funkcí $y = f(a)$ na FPGA. V případě metody (i) jsou použity pouze funkce, které je možné vyhodnotit pomocí několika komparátorů a multiplexerů (např. *ReLU*, *hard-tanh* nebo *hard-sigmoid*). Metoda (ii) využívá předpočítané tabulky a metoda (iii) využívá tzv. prahovací funkci definovanou rovnicí 2.15.

Nástroj FINN používá prahovací funkci pro všechny výše zmíněné účely v rámci tzv. *streamlining* transformace [36]. Streamlining transformace umožňuje transformovat uniformně kvantizované neuronové sítě na výpočetní graf obsahující pouze celočíselné operace. Všechny nelineární aktivační funkce jsou nahrazeny prahovacími funkcemi. Všechny zbylé škálovací faktory S nebo posuny Z jsou pohlceny do prahů těchto prahovacích funkcí. Konkrétně se může jednat o škálovací faktor $S_{W_i} \cdot S_x$, který zbyl v rovnici 2.14, nebo o parametry *batch* normalizace. Protože prahovací funkce může realizovat i rekvantizaci, mohou být kvantizační parametry následující vrstvy nezávislé na aktuální vrstvě.

Pro určení prahů t je nutné určit vstupní a výstupní kvantizační parametry a funkci $f : \mathbb{R} \rightarrow \mathbb{R}$, kterou má prahovací funkce realizovat. Pokud je f identita ($f(a) = a$), pak prahovací funkce realizuje pouze rekvantizaci. Předpokládejme kvantizaci vstupu a definovanou pomocí funkce $\mathcal{Q}_a$ a dekvantizaci výstupu y definovanou pomocí funkce $\mathcal{D}_y$. Pak je možné kvantizovanou verzi prahu t_i určit pomocí vzorce 2.16. Člen $i - 0.5$ představuje hranici

mezi dvěma výstupními kvantizačními hladinami ($i - 1$ a i). Člen $\mathcal{D}_y(i - 0.5)$ reprezentuje dekvantizovanou hodnotu této hranice. Práh t_i je určen projekcí hranice pomocí inverzní funkce f^{-1} . Kvantizovaná varianta prahu je nakonec získána pomocí funkce $\mathcal{Q}_a$.

$$q_{t_i} = \mathcal{Q}_a(f^{-1}(\mathcal{D}_y(i - 0.5))) \quad (2.16)$$

Je vhodné poznamenat, že počet prahů t je exponenciální vzhledem k bitové šířce výstupu y . Jelikož však tento výstup y často slouží jako vstup následující vrstvy, je v mnoha případech žádoucí jeho bitovou šířku omezit. Vyhodnocení lineárních vrstev na větší bitové šířce by totiž vyžadovalo více hardwarových zdrojů. Jelikož počet prahů nezávisí na bitové šířce akumulátoru, mohou být prahovací funkce vhodnější než předpočítané tabulky z přístupu (ii).

2.4 Způsoby výpočtu kvantizačních parametrů

Předtím, než bude možné neuronovou síť kvantizovat, je nutné určit potřebné kvantizační parametry (např. škálovací faktory S a posuny Z). Podle toho, zda výpočet kvantizačních parametrů zasahuje do trénovací fáze, je možné rozlišit: *Post-Training Quantization* (PTQ) a *Quantization-Aware Training* (QAT). PTQ je realizováno až po natrénování modelu, zatímco QAT spojuje kvantizaci s trénováním.

Během PTQ se z parametrů a aktivací natrénovaného modelu počítají takové statistiky, podle kterých je možné určit potřebné kvantizační parametry. Pro kvantizaci parametrů lze tyto statistiky získat přímo. Pro kvantizaci aktivací je ale nutné model vyhodnotit na reprezentativní datové sadě a získat z něj požadované mezivýsledky (aktivace). V literatuře se tento proces nazývá *kalibrace*.

V případě uniformní kvantizace je pro určení škálovacího faktoru S a posunu Z nutné nejprve zvolit kvantizační rozsah $\langle x_{min}, x_{max} \rangle$. Tento rozsah lze určit například podle minima a maxima ze sledovaných hodnot. Tyto statistiky ale mohou být citlivé na odlehlé hodnoty, které mohou způsobit zbytečné rozšíření kvantizačního rozsahu. Proto je vhodnější kvantizační rozsah určit podle percentilů sledovaných hodnot. Existují také pokročilejší metody, které na základě histogramů ze sledovaných hodnot optimalizují kvantizační parametry tak, aby minimalizovaly zvolenou metriku (např. KL divergenci) [11].

Zatímco metoda PTQ pracuje s již natrénovaným modelem, metoda QAT simuluje kvantizaci již během trénování. QAT současně optimalizuje parametry kvantizace i parametry modelu. Díky tomu mohou být nalezeny parametry neuronové sítě, které jsou z hlediska kvantizace výhodnější. Za zmínku stojí, že kvantizační chyby vznikající během QAT mohou sloužit i jako forma regularizace, díky které může být přesnost kvantizovaného modelu v některých případech dokonce vyšší než přesnost původního modelu [3].

Během trénovací fáze QAT metody je neuronová síť vyhodnocována v FP typech a kvantizace jsou pouze simulovány pomocí tzv. *simulované kvantizace* (*fake quantization*). Přímé použití kvantizovaných datových typů by totiž způsobilo numerické chyby a nestabilitu trénování. Na určitá místa v neuronové síti jsou vloženy operátory simulované kvantizace, které zajistí vznik stejných kvantizačních chyb jako během inference. V případě asymetrické celočíselné kvantizace definované v sekci 2.1 je operátor simulované kvantizace určen rovnicí 2.17. Jedná se o kvantizaci následovanou okamžitou dekvantizací. Aby tedy bylo možné během QAT simulovat kvantizaci aktivací, je nutné zasáhnout do implementace modelu a vložit do něj tyto operátory.

$$q_x = \mathcal{D}_x(\mathcal{Q}_x(x)) = S \cdot \left(\text{clip} \left(\text{round} \left(\frac{x}{S} + Z \right), q_{x_{\min}}, q_{x_{\max}} \right) - Z \right) \quad (2.17)$$

Protože kvantizace obsahuje nediferencovatelné operace (viz *round* a *clip* v rovnici 2.17), není možné skrze operátory simulované kvantizace propagovat gradienty nutné pro trénování neuronové sítě. QAT metody proto pouze aproximují skutečné derivace těchto operátorů. Často používanou aproximací je tzv. *Straight-Through Estimator* (STE). Při dopředném výpočtu jsou operátory simulované kvantizace aplikovány beze změny, ale při zpětném průchodu jsou úplně přeskočeny. I přesto, že vynecháním těchto operací vzniká určitá chyba v úpravě parametrů, STE dosahuje v řadě případů dostatečně dobrých výsledků [11].

Samotná kvantizace je realizována nástroji, které rozšiřují frameworky pro trénování neuronových sítí (např. *PyTorch* nebo *TensorFlow*). Srovnání konkrétních nástrojů zobrazuje tabulka 2.4. V tabulce jsou zahrnuty pouze nástroje, které podporují kvantizaci parametrů i aktivací. Nástroje jsou rozlišeny podle podpory metod PTQ, QAT a podle podporovaných datových typů a jejich bitových šířek.

Tabulka 2.4: Srovnání populárních nástrojů pro kvantizaci neuronových sítí. Nástroje jsou rozděleny podle frameworku, pro který jsou primárně určeny (PyTorch nebo TensorFlow). Znak • označuje podporu dané funkce. Sloupce *PTQ* a *QAT* označují podporu *Post-Training Quantization* a *Quantization-Aware Training*. Sloupec *Práh* označuje podporu kvantizace prahů lineárních vrstev. Nástroje označené sloupcem *Op.* nabízejí operátor simulované kvantizace, pomocí kterého lze kvantizovat libovolné části modelu (parametry i aktivace). Sloupce *FP*, *FX*, *INT* a *BIN* označují podporované datové typy a použitelné bitové šířky. Znak *X* označuje libovolnou bitovou šířku. Sloupec *BIN* označuje speciální podporu pro binární nebo ternární kvantizaci. Hodnota 1 ve sloupci *BIN* označuje podporu pro binární kvantizaci, zatímco hodnota 2 označuje podporu pro ternární kvantizaci. Informace byly čerpány převážně z [17] a z příslušných dokumentací jednotlivých nástrojů.

Nástroj	PTQ	QAT	Práh	Op.	FP	FX	INT	BIN
PyTorch								
Brevitas	•	•	•	•	1-8	X	X	1,2
TorchAO	•	•	–	–	4,6,8	–	X	–
ModelOpt (NVIDIA)	•	•	•	•	4,8	–	X	–
Quark (AMD)	•	•	•	•	4,6,8,16	–	4, 8	–
optimum-quanto	•	•	–	–	8	–	8	–
TensorFlow (Keras)								
TensorFlow Lite	•	•	•	•	16	–	8	–
QKeras	–	•	•	•	–	–	X	1,2
Larq	•	•	–	•	–	–	X	1,2
HGQ	–	•	•	•	–	X	–	–
PyTorch i TensorFlow								
NNI (Microsoft)	•	•	–	–	–	–	X	1
AIMET (Qualcomm)	•	•	•	–	16	–	4,8	–
NNCF (Intel)	•	•	•	–	4,8	–	4,8	–
Vitis AI (AMD)	•	•	•	•	16	–	8	–

Je vhodné poznamenat, že některé nástroje nepodporují kvantizaci prahů b lineárních vrstev. V případě GPU akcelerace má totiž přičtení prahu v FP datových typech minimální dopad na výkonnost. Naopak v případě FPGA je kvantizace prahů často nezbytná. Navíc některé nástroje neumožňují umístit operátor simulované kvantizace na libovolné místo v modelu (viz sloupec *Op.* v tabulce 2.4). V takovém případě nemusí být možné kvantizovat složitější architektury modelů (např. rekurentní síť *GRU* nebo *LSTM*).

Pouze nástroje *Brevitas*, *Larq*, *HGQ* a *Vitis AI* jsou zaměřené na FPGA akceleraci neuronových sítí. Nástroj *Brevitas* poskytuje pravděpodobně nejvyšší úroveň konfigurovatelnosti uniformní kvantizace. Kromě předdefinovaných vrstev a operátorů simulované kvantizace je možné sestavit i vlastní kvantizační operátor. Nástroj *Larq* je specializovaný na binární a ternární kvantizaci. Nástroj *HGQ* nabízí nejjemnější granularitu kvantizace, neboť každá hodnota může mít svůj vlastní kvantizační rozsah. Avšak kvůli tomu podporuje pouze FX datové typy.

2.5 FPGA akcelerátory neuronových sítí

Akcelerátory neuronových sítí lze rozdělit do dvou hlavních kategorií: (i) univerzální akcelerátory řízené instrukcemi a (ii) akcelerátory specializované pro jednu konkrétní neuronovou síť. V případě kategorie (i) je neuronová síť zkompileována do posloupnosti instrukcí, které jsou postupně vykonávány sadou univerzálních výpočetních elementů. Tato řešení mají tzv. *temporální architekturu*, neboť jednotlivé vrstvy neuronové sítě se v čase střídají na stejných výpočetních elementech. Tyto akcelerátory jsou podobné GPU, ale díky tomu, že se fyzicky nacházejí přímo v FPGA vedle uživatelské aplikace, mohou dosáhnout nižší latence. Navíc oproti GPU umožňují konfigurovat počet výpočetních elementů a tím i úroveň paralelismu. Nejčastěji se jedná o komerční nástroje (např. *Vitis AI* od AMD).

Řešení kategorie (ii) mají nejčastěji tzv. *prostorovou architekturu* (*spatial architecture*). Celá neuronová síť je realizována jako zřetězená linka, ve které je každý stupeň specializovaný na konkrétní vrstvu. Díky zřetězení je možné dosáhnout vyšší propustnosti než v případě temporální architektury.

Spotřebu hardwarových zdrojů je možné regulovat pomocí úrovně paralelismu každého stupně linky (v literatuře se používá termín *folding* nebo *reuse* faktor). Při reuse faktoru větším než jedna budou jednotlivé části vrstev zpracovávány postupně na stejných výpočetních elementech. Tím dojde ke zvýšení inicializačního intervalu zřetězené linky, zvýšení latence a snížení propustnosti. Navíc nebude možné výpočetní elementy specializovat pro jeden konkrétní výpočet. Také může být nezbytné mezi jednotlivé stupně vložit fronty, které dále zvýší latenci a využití hardwarových zdrojů. Pro akceleraci větších neuronových sítí je ale tato úroveň paralelismu nezbytná.

Návrh zřetězené linky akcelerující neuronové sítě je komplikovanější než překlad modelu do posloupnosti instrukcí pro univerzální akcelerátor. I přesto existují nástroje, které umožňují tuto linku navrhnout automatizovaně. Nejčastěji se ale jedná o výzkumné nástroje (např. *FINN* od AMD), které podporují pouze některé ze standardních typů neuronových sítí.

Prvním krokem automatické konverze neuronové sítě na zřetězenou linku je získání výpočetního grafu modelu. Nástroje pro trénování neuronových sítí (např. *PyTorch*) umožňují tento výpočetní graf zachytit automatizovaně a uložit jej do datového formátu ONNX. Při použití QAT (např. pomocí nástroje *Brevitas*) je vhodnější formát *QONNX* (*Quantized ONNX*), který umožňuje do výpočetního grafu uložit i operátory simulované kvantizace

(viz sekce 2.4). Tyto operátory obsahují informace o kvantizačních parametrech v konkrétní části modelu (např. škálovací faktor, bitová šířka, atd.).

Klíčovou fází celé konverze jsou transformace a optimalizace výpočetního grafu. Tyto transformace mohou zahrnovat například: (i) odvození chybějících kvantizačních parametrů, (ii) streamlining transformaci (viz sekce 2.3), (iii) odstranění nepotřebných operací, (iv) algebraické zjednodušení výpočetního grafu, (v) vložení registrů nebo front, (vi) kalibrace hloubky front, (vii) minimalizace bitových šířek a (viii) konverze uzlů na hardwarové bloky.

Poté nástroje vygenerují popis akcelérátoru v jazyce *HLS* (*High-Level Synthesis*) nebo na RTL úrovni (např. *VHDL* nebo *Verilog*). Hlavní výhoda HLS spočívá v automatickém rozmístění registrů do výpočetního grafu. Nakonec lze použít standardní nástroje (např. *Vivado* nebo *Quartus*), které z tohoto popisu vygenerují binární konfiguraci pro FPGA.

Tabulka 2.5 zobrazuje srovnání existujících nástrojů pro návrh FPGA akcelérátorů. Řešení jsou rozdělena podle architektury (*temporální* nebo *prostorová*), podle výstupu (HLS nebo RTL) a podle podporovaných datových typů. Řada nástrojů umožňuje implementovat pouze jednoduché nelineární aktivační funkce (např. ReLU). Nástroje, které podporují složitější aktivační funkce, mají ve sloupci *Nelineární funkce* uveden (místo znaku „–“) konkrétní způsob jejich implementace.

V případě temporální architektury jsou nelineární aktivační funkce vyhodnocovány pomocí FP datových typů, nebo jsou podporovány pouze jednoduché aktivační funkce. Na základě sekce 2.3 je možné tato řešení zařadit do (ii) úrovně kvantizace neuronových sítí.

FINN je jediný nástroj, který podporuje *streamlining* transformaci (viz sekce 2.3), a který implementuje nelineární aktivační funkce pomocí prahovacích funkcí. Ostatní nástroje s prostorovou architekturou jsou omezeny pouze na FX typy. Díky omezení škálovacích faktorů na mocniny dvou je možné konverze mezi různými kvantizačními rozsahy realizovat pouze pomocí bitových posunů. Ale jak již bylo zmíněno, omezení škálovacích faktorů může snížit využití dostupné bitové šířky.

Tabulka 2.5: Srovnání nástrojů pro návrh FPGA akcelérátorů neuronových sítí. Nástroje jsou rozděleny podle architektury (*temporální* nebo *prostorová*), podle výstupu (HLS nebo RTL), podle podporovaných datových typů a podle způsobu implementace nelineárních funkcí. Pokud sloupec *Nelineární funkce* obsahuje znak „–“, pak daný nástroj podporuje pouze jednoduché nelineární funkce (např. ReLU). Veškeré informace byly čerpány z dokumentací jednotlivých nástrojů.

Nástroj	Výstup	Datové typy	Nelineární funkce
Temporální architektura			
Vitis AI (AMD)	RTL	FP-16, INT-8	–
Matlab DL Toolbox	RTL	FP-32, INT-8	FP-32
FPGA AI Suite (Intel)	HLS / RTL	FP-16, INT-8	FP-16
cgra4ml	RTL	INT-X	–
Prostorová architektura			
FINN (AMD)	HLS / RTL	INT-X	Prahovací f.
HLS4ML	HLS	FX-X	LUT
tinyHLS	HLS	FX-X	–

Kapitola 3

Klasifikace šifrovaného provozu

Cílem klasifikace šifrovaného provozu (angl. *Encrypted Traffic Classification* nebo zkráceně ETC) je zařazení síťových toků¹ do předem zvolených kategorií. Přestože je většina síťové komunikace šifrovaná, existují metody strojového učení, které v síťovém provozu dokáží rozpoznat charakteristické vzory a identifikovat konkrétní typ komunikace. Cílem není prolomit samotné šifrování a zjistit přesný obsah paketů, ale pouze odhadnout, o který typ provozu se jedná.

V literatuře bývá provoz klasifikován do následujících kategorií: (i) hrubé kategorie služeb (webová pošta, VoIP, přenos souborů apod.), (ii) konkrétní webové aplikace (YouTube, WhatsApp, Spotify apod.), (iii) konkrétní akce v rámci dané aplikace (odeslání zprávy, nahrání souboru apod.), (iv) konkrétní bezpečnostní hrozby nebo anomálie (DDoS útok, malware apod.), (v) typ šifrovacího protokolu (TLS, QUIC, WireGuard, Tor apod.), (vi) konkrétní operační systém nebo typ zařízení a (vii) konkrétní webová stránka (tzv. *website fingerprinting*). Výsledek klasifikace může být použit pro monitorování sítě, blokaci konkrétních služeb, zajištění kvality služeb (QoS) nebo včasné odhalení bezpečnostních hrozeb.

V sekci 3.1 jsou popsány základní principy populárních šifrovacích protokolů. Sekce 3.2 obsahuje popis vstupních příznaků a datových sad používaných v ETC. Samotné způsoby klasifikace jsou následně rozděleny na *offline* a *online* přístupy. Zatímco offline metody popsané v sekci 3.3 se zaměřují na analýzu již ukončeného síťového provozu, online metody popsané v sekci 3.4 využívají výsledek klasifikace v reálném čase. Zbytek práce bude zaměřen převážně na metody online klasifikace.

3.1 Šifrování síťového provozu

Na síťové komunikaci se podílí celá řada zařízení a systémů (síťové přepínače, směrovače, servery, různé operační systémy apod.). Proto bylo vytvořeno několik abstrakcí, které usnadňují pochopení celého procesu. Například model TCP/IP definuje abstrakci do následujících vrstev: (i) fyzická (L1) a linková (L2) vrstva, zajišťující komunikaci na úrovni konkrétního fyzického média, (ii) síťová (L3) vrstva, která spravuje adresování a směrování v rámci celé sítě, (iii) transportní (L4) vrstva, zodpovědná za doručení dat až do konkrétní aplikace a (iv) aplikační (L5 až L7) vrstva, definující formát a význam přenášených dat.

Aby zařízení a systémy pracující na daných vrstvách mohly plnit své funkce, přidávají k přenášeným datům dodatečné informace (tzv. *protokolové hlavičky*). I tyto hlavičky mo-

¹Síťovým tokem se v této práci rozumí tzv. *bidirectional flow*, který je identifikován následující pěticí údajů: zdrojová a cílová IP adresa, zdrojový a cílový port a protokol.

hou obsahovat citlivé informace, které je vhodné šifrovat. Podle toho, na které vrstvě je šifrování realizováno, se odvíjí i množství hlaviček, které jsou chráněny. Šifrování linkové vrstvy (např. *MACsec*) chrání data síťové, transportní i aplikační vrstvy, zatímco šifrování na aplikační vrstvě (např. *PGP* nebo *SSH*) chrání pouze data aplikační vrstvy. Nicméně šifrování na nižších vrstvách má kratší dosah. Např. šifrování na úrovni linkové vrstvy je platné pouze na konkrétním fyzickém médiu. Jakmile data dorazí k prvnímu síťovému přepínači, dojde k jejich dešifrování. Bezpečnost přenosu na další zařízení pak závisí na implementaci zbytku síťové infrastruktury.

Nejrozšířenější je šifrování na úrovni transportní vrstvy, které zabezpečuje komunikaci od koncového zařízení až k cílovému serveru. Populárními protokoly v této kategorii jsou TLS (*Transport Layer Security*) a QUIC (*Quick UDP Internet Connections*). Oba protokoly mají společnou počáteční fázi (tzv. handshake fáze), která je založená na asymetrické kryptografii. Cíle této fáze jsou následující: zvolení šifrovacích algoritmů a jejich parametrů, autentizace jedné nebo obou stran komunikace pomocí certifikátů a ustanovení společné tajné informace, ze které bude následně odvozen klíč pro symetrickou kryptografii. Jelikož je asymetrická kryptografie výpočetně náročnější než symetrická, je pro přenos samotných aplikačních dat použita symetrická kryptografie.

Protokoly TLS a QUIC se odlišují v použitém transportním protokolu. Zatímco TLS používá TCP, QUIC je založen na jednodušším a rychlejším protokolu UDP. Mechanismy, které protokol TCP přináší (spolehlivý přenos, ochrana proti zahlcení sítě atd.), jsou v režii QUIC protokolu. Díky tomu je ještě více informací transportní vrstvy chráněno šifrováním (např. sekvenční a potvrzovací čísla nebo příznaky o ukončení nebo obnovení spojení).

Počáteční paket TLS a QUIC spojení (tzv. *Client Hello*) je v základní variantě nešifrovaný. To může útočníkovi odhalit podporované verze šifrovacích algoritmů, ale také cílovou doménu, o kterou klient žádá server (tzv. *Server Name Indication* nebo zkráceně SNI). Řada webů je totiž distribuována pomocí CDN (*Content Delivery Network*), které hostují více domén na stejné IP adrese. V takovém případě je SNI pole klíčové pro zvolení konkrétní webové aplikace, se kterou má klient zájem komunikovat.

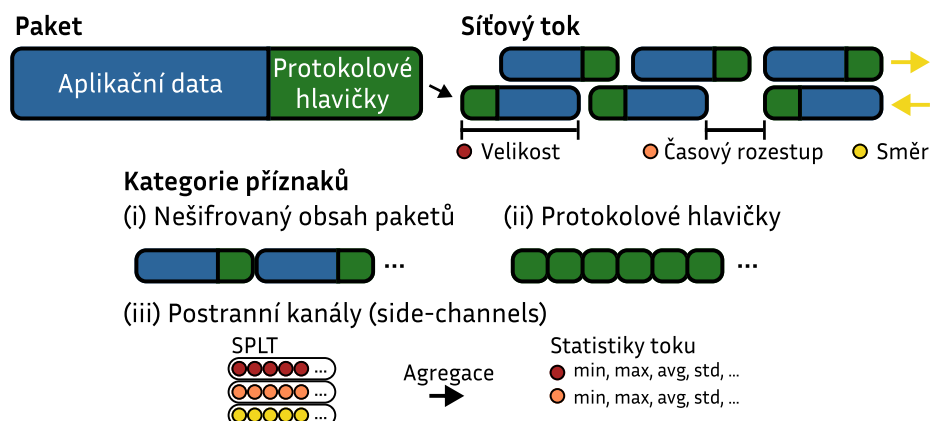
S cílem skrytí informací o cílové doméně (SNI pole), byly TLS a QUIC protokoly rozšířeny i o možnost šifrování prvního paketu Handshake fáze (tzv. *Encrypted Client Hello* nebo zkráceně ECH) [29]. V takovém případě klient zašifruje počáteční paket pomocí dočasného klíče odvozeného z veřejného klíče serveru. Při použití ECH zůstávají viditelné IP adresy, porty a vnější SNI. Tyto údaje budou ale stejné pro všechny domény hostované na stejném CDN, proto nelze s jistotou určit, které domény klient navštěvuje. Přesto existují v oblasti klasifikace šifrovaného provozu způsoby, pomocí kterých lze cílovou doménu nebo webovou aplikaci odhadnout i v případě použití ECH [23].

3.2 Příznaky a datové sady

Datové příznaky, které bývají v literatuře využity pro ETC, je možné rozdělit do tří hlavních kategorií: (i) nešifrovaný obsah paketů, (ii) protokolové hlavičky a (iii) informace z postranních kanálů (tzv. *side-channel* informace). Kategorie příznaků jsou znázorněny na obrázku 3.1.

První kategorií příznaků bylo možné využít v době, kdy byla velká část síťové komunikace nešifrovaná. Konkrétně se jednalo o metody nazývané *Deep Packet Inspection* (DPI), které analyzovaly samotný obsah aplikačních dat.

Příznaky kategorie (ii) jsou nejčastěji tvořeny protokolovými hlavičkami z handshake fáze protokolu TLS nebo QUIC. Konkrétně jsou použity např. v systému Kaleidoscope [37]



Obrázek 3.1: Schématické znázornění paketu, síťového toku a tří hlavních kategorií příznaků využívaných pro klasifikaci šifrovaného provozu. Kategorie (i) je použitelná pouze pro nešifrovaný provoz. Kategorie (ii) využívá nešifrované protokolové hlavičky (převážně z handshake fáze TLS nebo QUIC protokolu). Kategorie (iii) využívá informace z postranních kanálů. Příznaky SPLT jsou složeny ze sekvencí velikostí paketů, časových rozestupů mezi pakety a směru paketů v rámci toku. Statistiky toku obsahují pouze statistické charakteristiky z těchto sekvencí.

nebo v modelu DISTILLER [27]. Nicméně použitelnost těchto příznaků začíná být omezo-
vána již zmíněným rozšířením ECH, které šifruje i počáteční paket celého spojení.

U příznaků kategorie (ii) je také nutné pečlivě odstranit všechna datová pole, která by
mohla způsobit přetrénování modelu na konkrétní datovou sadu. Jedná se o údaje, které
jsou platné pouze pro konkrétní spojení (např. TCP window size, IP header checksum,
TCP sequence number) nebo pro konkrétní zapojení síťové infrastruktury (např. MAC
adresy, IP adresy, porty apod.) [38]. Důsledky zařazení takových polí do vstupních příznaků
dobře ilustruje práce [23], která mimo jiné uvádí přesnost klasifikace modelu založeného
na exaktním porovnávání IP adres. Zatímco na trénovacích datech dosáhl model téměř
100% přesnosti, na testovacích datech vzdálených o více než týden klesla přesnost na téměř
0 % v důsledku změn IP adres některých serverů.

Třetí kategorie příznaků využívá informace, které nejsou přímo získány z obsahu komu-
nikace, ale spíše z jejího průběhu. Jedná se např. o časové řady SPLT (*Sequence of Packet
Lengths and Times*) [21], které pro každý paket evidují jeho velikost, časový rozestup od
předchozího paketu a jeho směr v aktuálním toku (příchozí nebo odchozí). Tyto příznaky
odrážejí nejen komunikaci mezi klientem a serverem, ale i interakci uživatele s jeho apli-
kací. Např. při načtení webové stránky vznikne tok s odlišnými velikostmi paketů a časovými
prodlevami než v případě hlasového hovoru.

Vzhledem k rostoucí propustnosti síťové infrastruktury je zachycení, uložení a zpraco-
vání rozsáhlých datových sad stále náročnější. Proto SPLT příznaky neobsahují informace
o všech paketech celého toku, ale jsou omezeny pouze na několik desítek paketů (často
prvních 20 nebo 30 paketů). Práce [21] uvádí, že více než 15 % úspěšně ustanovených TLS
spojení má více než 30 paketů, zatímco u QUIC protokolu je to více než 40 %. Přesto
práce [24] ukazuje, že pouze pomocí SPLT příznaků omezených na prvních 30 paketů je
možné klasifikovat TLS spojení s přesností 97,04 %. Ve zmíněné práci bylo také zjištěno, že
nejdůležitější částí SPLT příznaků jsou právě velikosti paketů. Při použití pouze sekvence
velikostí paketů došlo k poklesu přesnosti klasifikace jen o 0,5 %.

Jednou z alternativ SPLT příznaků jsou SBLT příznaky (*Sequence of Burst Lengths and Times*) [21], které pakety seskupují do tzv. burstů. Seskupování paketů sice vede ke ztrátě části informace, ale zároveň umožňuje využít více paketů toku bez nárůstu velikosti příznaků.

Další alternativou je agregace SPLT časových řad do *statistik toku*. Jedná se o statistické charakteristiky (např. minima, maxima, průměry, směrodatné odchylky, histogramy), které jsou vypočteny z velikostí nebo časových rozestupů paketů. Tyto statistiky sice mohou využít informace z celého toku, ale zároveň způsobí ztrátu časových závislostí mezi pakety.

Jedním z hlavních problémů všech zmíněných datových příznaků je tzv. *datový drift*. Jedná se o fenomén, kdy se samotná distribuce dat mění v čase vlivem změn v síťových protokolech, aplikacích, infrastruktuře a samotném chování uživatelů. To vede k postupnému snižování přesnosti klasifikátoru.

V literatuře neexistuje dokonalé řešení datového driftu. Jednou z možností je přetrénovat model na nových datech, čímž ale může dojít ke ztrátě již naučených znalostí. V literatuře se proto objevují následující alternativní metody: (i) *replay-based* přístupy, jejichž cílem je k novým datům přidat vhodně zvolené vzorky z původní datové sady, (ii) destilace znalostí (tzv. *knowledge distillation*), při které je starý model použit pro trénování nového modelu a (iii) inkrementální rozšiřování architektury modelu (např. model *I2RNN* [34]).

Před srovnáním existujících datových sad je vhodné zmínit článek [19], který poukazuje na nekorektní využívání datových sad v některých dosavadních publikacích. Autoři zjistili, že běžně používané datové sady pro klasifikaci síťového provozu obsahují velké množství toků s duplicitními příznaky (např. stejné sekvence velikostí a časových rozestupů paketů). Kvůli tomu vzniknou dva problémy.

První problém nastává, pokud toky s duplicitními příznaky přísluší ke stejné klasifikační třídě. Pokud je datová sada rozdělena na trénovací a testovací část náhodně, je pravděpodobné, že se duplicitní toky objeví v obou částech. V takovém případě dojde k nadhodnocení schopnosti generalizace modelu. Model je totiž testován na datech, která se již objevila během jeho tréninku. Pokud se však duplicitní toky objevují v různých třídách, nastává situace, kdy dané vzorky není možné správně klasifikovat (toky nelze s těmito příznaky rozlišit). Proto byli autoři práce [19] schopni pro dané datové sady vypočíst maximálně dosažitelnou přesnost klasifikace (viz tabulka 3.1).

Jednoduché řešení nadhodnocené generalizace, které článek [19] navrhuje, je chronologické rozdělení datové sady na trénovací a testovací část. Takovéto dělení lépe simuluje reálné nasazení modelu, neboť je model trénován na datech, která chronologicky předcházejí testovacím datům. Zároveň lze na takto rozdělené datové sadě pozorovat již zmíněný datový drift. Nicméně, jak poukazuje článek [19], řada prací stále používá náhodné rozdělení datové sady. Navíc řada datových sad a příslušných nástrojů nenabízí možnost chronologického rozdělení.

Tabulka 3.1 uvádí přehled populárních datových sad pro klasifikaci šifrovaného síťového provozu. U každé datové sady je uveden počet toků, počet tříd, možnost chronologického rozdělení datové sady, typ příznaků, které jsou v datové sadě k dispozici, nejlepší známý výsledek klasifikace (sloupec *SOTA* z angl. *State-of-the-Art*) a maximálně dosažitelná přesnost klasifikace (sloupec *Max. A*). Pokud je pro datovou sadu dostupný původní PCAP soubor, je možné provést chronologické rozdělení sady podle časových razítek paketů. Také je možné z tohoto souboru získat kterýkoliv ze zmíněných typů příznaků. Některé datové sady jsou omezeny pouze na SPLT příznaky, případně jejich podobné varianty.

Pro hodnocení úspěšnosti klasifikace jsou nejčastěji využívány následující metriky: přesnost klasifikace (angl. *accuracy* nebo zkráceně *A*) a *F1* skóre (F_1). Nejlépe interpretovatelná

Tabulka 3.1: Srovnání datových sad podle počtu toků, počtu klasifikovatelných tříd, možnosti chronologického rozdělení, typu příznaků, nejlepšího známého výsledku (SOTA) a maximální dosažitelné přesnosti. Symbol • označuje možnost chronologického rozdělení datové sady. Příznaky SPLT označují časové řady obsahující velikosti, časové rozestupy a směry paketů. PCAP označuje datové sady, pro které jsou k dispozici původní PCAP soubory. Ve sloupci SOTA je uvedena buď přesnost klasifikace ($\mathbb{A}$) nebo F1 skóre ($\mathbb{F}_1$). Maximální přesnost je teoretická horní hranice, které lze dosáhnout kvůli výskytu toků s duplicitními příznaky u rozdílných tříd. Informace byly čerpány z prací [19] a [22]. U datové sady ISCXVPN2016-T byly ve sloupci SOTA použity výsledky z práce [10].

Datová sada	P. toků	P. tříd	Chrono.	Příznaky	SOTA	Max. $\mathbb{A}$
CESNET-QUIC22	153M	102	•	SPLT	80,87 % $\mathbb{A}$	99,57 %
CESNET-TLS22	141M	191	•	SPLT	98,25 % $\mathbb{A}$	99,41 %
EdgeIoTset	73M	15	•	PCAP	99,82 % $\mathbb{A}$	99,85 %
AppClassNet	10M	500	—	SPLT	92,11 % $\mathbb{A}$	99,93 %
UNSW-NB15	2,5M	10	•	PCAP	99,17 % $\mathbb{A}$	99,17 %
CIC-IDS-2017	2M	8	•	PCAP	99,84 % $\mathbb{A}$	99,97 %
CIC-DoHBrw	1,5M	2	•	PCAP	98,66 % $\mathbb{A}$	98,79 %
MIRAGE19	122k	20	—	SPLT	85,30 % $\mathbb{F}_1$	96,36 %
MIRAGE22	59k	9	—	SPLT	98,67 % $\mathbb{F}_1$	96,90 %
ISCXVPN2016-T	10,5k	7	•	PCAP	97,43 % $\mathbb{A}$	98,87 %
UTMOBILENET21	9,5k	17	—	SPLT	88,73 % $\mathbb{F}_1$	93,46 %
UCDAVIS19	7k	5	—	SPLT	98,63 % $\mathbb{A}$	100,00 %

je přesnost, která je definována jako poměr správně klasifikovaných vzorků oproti počtu všech klasifikací. Pokud mají jednotlivé třídy datové sady výrazně odlišné počty vzorků, je vhodnější použít F1 skóre. Tato metrika totiž penalizuje modely, které dosahují vysoké přesnosti na nejpočetnějších třídách, ale selhávají na těch méně početných.

CESNET-QUIC22 a *CESNET-TLS22* jsou v aktuálním srovnání největší datové sady z pohledu počtu toků. Důležité je, že obsahují záznam skutečného síťového provozu, který byl zachycen v reálném prostředí. U každého síťového toku je uvedena příslušná webová aplikace (např. YouTube, Facebook, Instagram apod.), která byla zjištěna z SNI pole. *CESNET-QUIC22* obsahuje síťové toky s šifrováním QUIC, zatímco *CESNET-TLS22* obsahuje toky s šifrováním TLS.

Autoři článku [23] našli významný datový drift mezi prvním a druhým týdnem datové sady CESNET-QUIC22. Po podrobnějším prozkoumání bylo zjištěno, že došlo ke změně certifikátů služeb Google, což způsobilo významný pokles přesnosti klasifikace těchto služeb. Jelikož změna certifikátů ovlivnila převážně prvních 8 paketů toku (fáze handshake), podařilo se autorům částečně zmírnit dopad datového driftu zamaskováním těchto paketů během tréninku.

Datová sada *ISCXVPN2016* obsahuje záznamy toků s VPN tunelem a bez VPN tunelu. Toky jsou navíc rozděleny do kategorií podle hrubého typu provozu (web, email, chat, streaming, přesun souborů, VoIP a P2P). Datová sada byla zachycena v laboratorním prostředí, nicméně s příslušnými aplikacemi interagovali skuteční uživatelé. Podrobnější popis zbylých datových sad je uveden v práci [19].

3.3 Offline klasifikace šifrovaného provozu

Klasifikátory šifrovaného provozu lze rozdělit do dvou hlavních kategorií: (i) metody klasického strojového učení a (ii) metody hlubokého učení. První kategorie metod může přinést vyšší interpretovatelnost naučených modelů a také nižší výpočetní náročnost trénování a inference. Nicméně tyto metody jsou závislé na precizní přípravě vstupních příznaků, což limituje jejich škálovatelnost. Práce [31] řadí do této kategorie metody, jako jsou: (i) rozhodovací stromy (*Decision Trees* a *Random Forests*), (ii) *gradient boosting* metody (*XGBoost* nebo *LightGBM*), (iii) *Support Vector Machines* (SVM) a (iv) pravděpodobnostní modely (např. *Gaussian Mixture Model* nebo *Hidden Markov Model*).

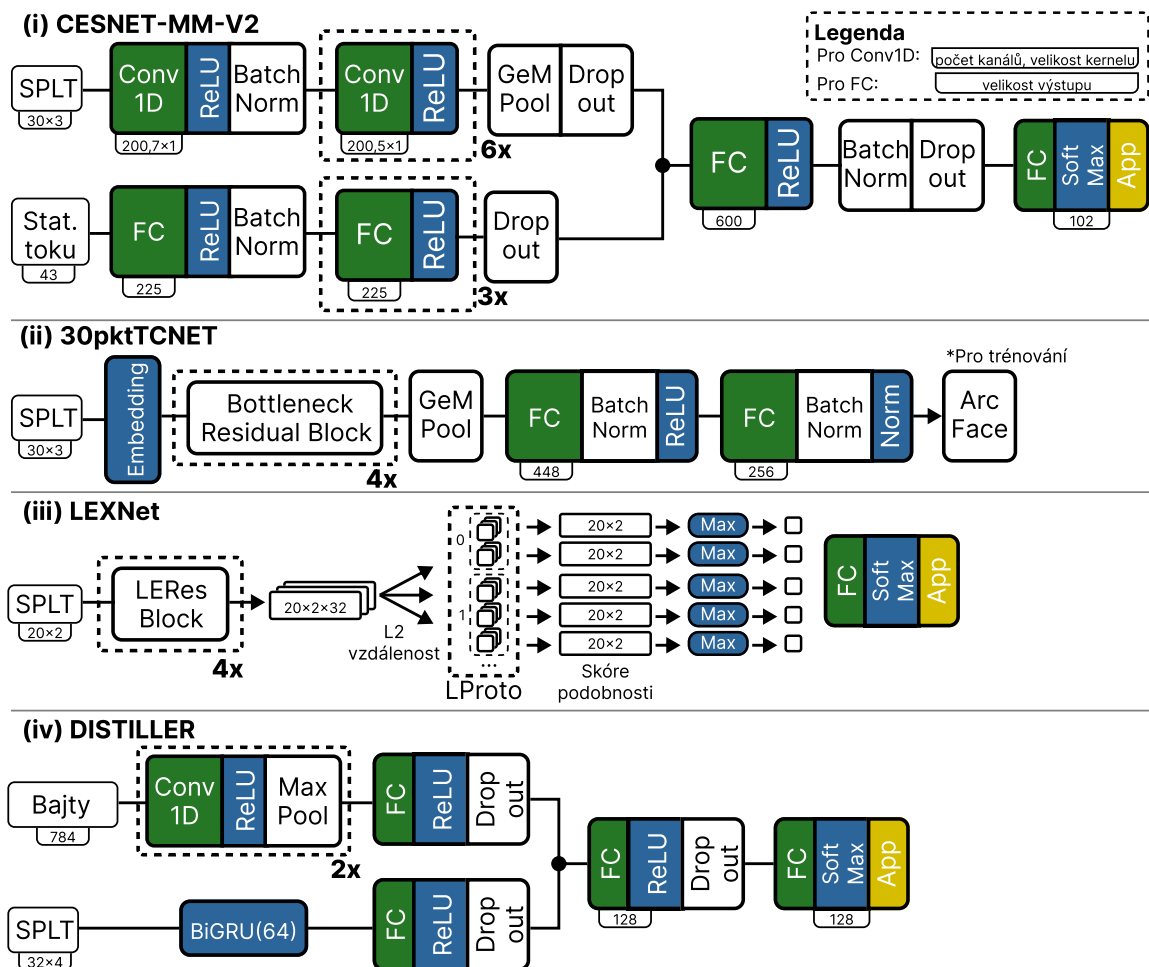
Druhým přístupem jsou metody hlubokého učení, které umožňují automatizovaně extrahovat komplexnější příznaky. Do této kategorie spadají zejména neuronové sítě, kterých již byla v oblasti ETC navržena široká škála. Práce [31] zmiňuje následující kategorie: (i) plně propojené neuronové sítě (*MLP*), (ii) konvoluční neuronové sítě (*CNN*), (iii) rekurentní neuronové sítě (*RNN*), (iv) grafové neuronové sítě (*GNN*), (v) modely s transformer architekturou a (vi) generativní modely (*AE*, *VAE*, *GAN* apod.).

V oblasti ETC bylo navrženo velké množství různých modelů. Proto se tato práce zaměřuje pouze na ty nejvýznamnější, které dosáhly nejlepších výsledků na datových sadách popsanych v sekci 3.2. Výsledek srovnání je zobrazen v tabulce 3.2. U každé datové sady je uveden název modelu, jeho architektura, typ vstupních příznaků, počet parametrů a dosažená přesnost klasifikace ($\mathbb{A}$) nebo F1 skóre ($\mathbb{F}_1$). Jednotlivá řešení jsou popsána níže.

Tabulka 3.2: Srovnání nejlepších známých výsledků klasifikace šifrovaného provozu na populárních datových sadách. U každé datové sady je uveden název modelu, jeho architektura, typ vstupních příznaků, počet parametrů a dosažená přesnost klasifikace ($\mathbb{A}$) nebo F1 skóre ($\mathbb{F}_1$). Informace, které nebylo možné dohledat, jsou označeny „–“. SPLT označuje časové řady obsahující velikosti, časové rozestupy a směry paketů, *stat.* označuje statistiky toku a *raw* označuje surové bajty z prvních paketů.

Datová sada	Model	Arch.	Vstup	Param.	$\mathbb{A}$ / $\mathbb{F}_1$
CESNET-QUIC22	LightGBM	LightGBM	SPLT+stat.	–	80,87 % $\mathbb{A}$
	CESNET-MM-V2	CNN+MLP	SPLT+stat.	2.3M	82,74 % $\mathbb{F}_1$
CESNET-TLS22	30pktTCNET	CNN	SPLT	1M	98,25 % $\mathbb{A}$
	LEXNet	CNN	SPLT	119k	97,20 % $\mathbb{A}$
AppClassNet	30pktTCNET	CNN	SPLT	1M	92,11 % $\mathbb{A}$
MIRAGE19	30pktTCNET	CNN	SPLT	1M	85,30 % $\mathbb{F}_1$
MIRAGE22	30pktTCNET	CNN	SPLT	1M	98,67 % $\mathbb{F}_1$
ISCXVPN2016*	DISTILLER	CNN+GRU	SPLT+raw	–	81,71 % $\mathbb{A}$
	30pktTCNET	CNN	SPLT	1M	80,19 % $\mathbb{A}$
ISCXVPN2016	FENIX	CNN	SPLT	36k	97,43 % $\mathbb{A}$
UTMOBILENET21	30pktTCNET	CNN	SPLT	1M	88,73 % $\mathbb{F}_1$

Na datové sadě CESNET-QUIC22 dosáhl nejvyšší přesnosti ($\mathbb{A}$) model *LightGBM*, nicméně z pohledu F1 skóre ($\mathbb{F}_1$) jej překonala neuronová síť *CESNET-MM-V2* [23]. Vstupem obou modelů jsou SPLT příznaky a statistiky toku (jedná se tedy o multimodální modely). Architektura neuronové sítě je znázorněna na obrázku 3.2(i). SPLT příznaky jsou zpracovány pomocí konvolučních vrstev, zatímco statistiky toku jsou zpracovány pomocí plně propojených vrstev. Obě modalitty jsou zkombinovány pomocí dalších plně propojených vrstev.



Obrázek 3.2: Srovnání čtyř neuronových sítí pro klasifikaci šifrovaného provozu: (i) *CESNET-MM-V2* [23], (ii) *30pktTCNET* [22], (iii) *LEXNet* [8] a (iv) *DISTILLER* [27]. Zelené bloky označují lineární vrstvy, modré bloky označují nelineární aktivační funkce a bílé bloky označují speciální operace. Blok FC označuje plně propojenou vrstvu (*Fully Connected layer*), zatímco blok Conv1D označuje 1D konvoluční vrstvu. U lineárních vrstev je uveden jejich rozměr v počtu výstupních příznaků (pro FC blok) nebo v počtu kanálů a velikosti jádra (pro Conv1D blok).

Vstupem modelu *30pktTCNET* [22] jsou pouze SPLT příznaky, které jsou nejprve předzpracovány embedding vrstvou. Jádro modelu tvoří konvoluční bloky inspirované architekturou *ResNet* (konkrétně *Bottleneck Residual Block*). Výstupem modelu je embedding vektor namísto pravděpodobnosti tříd jako u ostatních modelů.

Autoři model předtrénovali na datové sadě CESNET-QUIC22 pomocí ztrátové funkce *ArcFace*. Inference je realizována pomocí tzv. *database ranking*. Aktuální výstupní embedding vektor je porovnán s embedding vektory z předpřipravené databáze. Výsledná třída je následně zjištěna podle tříd několika nejbližších nalezených vektorů. Autoři také úspěšně využili tento natrénovaný model na nových datových sadách pomocí metody *transfer learning*.

Tím dokonce překonali dosavadní nejlepší řešení na datových sadách: CESNET-TLS22, MIRAGE19, MIRAGE22, AppClassNet a UTMOBILENET21².

Také je vhodné zmínit model *LEXNet* [8], který dosáhl podobných výsledků jako model *30pktTCNET*, ale s $8,4\times$ méně parametry. Jeho vstupní příznaky obsahují pouze velikosti a směry prvních 20 paketů toku (*30pktTCNET* a *CESNET-MM-V2* používají příznaky z prvních 30 paketů). *LEXNet* je složený z optimalizovaných konvolučních bloků a z matice tzv. *prototypů* (viz obrázek 3.2(iii)). Právě díky této matici nabízí částečnou vysvětlitelnost výsledků klasifikace. Detailní popis architektury a trénování modelu *LEXNet* je uveden v práci [8].

Srovnání výsledků na datové sadě ISCXVPN2016 je o něco složitější, protože různé práce použily různá předzpracování. Práce [27] v této datové sadě našla 65 % toků obsahujících pouze jeden UDP paket, který byl vygenerován aplikací *BlueStacks* během vytváření datové sady. Proto některé práce tyto problematické toky z datové sady odstranily. Takto upravená datová sada je v tabulce 3.2 označena symbolem *. Protože řada modelů z následující sekce 3.4 byla testována na neupravené datové sadě ISCXVPN2016, je v tabulce 3.2 uveden i jeden z těchto výsledků.

Na předzpracované datové sadě *ISCXVPN2016** dosáhl nejlepšího výsledku model *DISTILLER* [27], který využívá jak SPLT příznaky (z prvních 32 paketů), tak i surové bajty paketů³. V rámci SPLT příznaků byla kromě velikostí, časových rozestupů a směrů paketů použita také velikost TCP okna. SPLT příznaky jsou následně zpracovány pomocí rekurentní sítě BiGRU (*Bidirectional GRU*), zatímco surové bajty jsou zpracovány pomocí konvolučních vrstev (viz obrázek 3.2(iv)). Obě modalitty jsou zkombinovány pomocí dvou plně propojených vrstev. Pokud budeme uvažovat pouze řešení, která používají jen SPLT příznaky, pak nejlepšího výsledku dosáhl opět model *30pktTCNET*.

3.4 Online klasifikace šifrovaného provozu

Řada aplikací ETC (např. blokáce provozu nebo prioritizace toků) vyžaduje ke své funkci přímou interakci se síťovým provozem. V takových případech je latence klasifikátoru kritická. Akcelerace modelů na GPU nebo CPU je z tohoto pohledu nevýhodná (práce [4] a [10] uvádějí latenci klasifikace v řádu několika milisekund). Kvůli vysoké latenci klasifikátoru může dojít např. k propuštění škodlivých paketů ještě před jejich zablokováním.

S cílem snížit latenci klasifikace vznikla v oblasti ETC nová řešení (tzv. *online* klasifikátory). Tato nová řešení používají pro akceleraci klasifikátorů: (i) programovatelné síťové přepínače (*programmable switch*) [40], (ii) chytré síťové karty (SmartNIC) s technologií FPGA [33] nebo (iii) kombinaci předchozích přístupů [10].

Programovatelné síťové přepínače mohou dosahovat propustnosti v řádu několika Tb/s, avšak zavádějí striktní omezení na implementovaná řešení. Mezi hlavní omezení patří například limit výpočetních stupňů zřetězené linky, množství dostupné paměti nebo absence operací s plovoucí desetinnou čárkou. Problematický je také *read-modify-write* přístup k registrům, který je často možný pouze u jednoduchých atomických operací. Naopak FPGA nabízejí větší flexibilitu implementace, avšak propustnost jejich síťového rozhraní dosahuje řádově stovek Gb/s. Kvůli odlišné propustnosti a flexibilitě obou řešení se liší i cílová apli-

²Je možné, že model *30pktTCNET* dosáhl nejlepších výsledků i na datové sadě CESNET-QUIC22. Nicméně autoři tento model nevyhodnotili na celé datové sadě, ale pouze na její části. Výsledky uvedené v práci [22] tedy nejsou přímo srovnatelné s výsledky modelů *LightGBM* a *CESNET-MM-V2*.

³Autoři konkrétně použili prvních 784 bajtů z těla transportní vrstvy. V této části se převážně nacházejí hlavičky protokolů TLS nebo QUIC a začátek aplikačních dat.

kace. Například v datových centrech mohou programovatelné přepínače řešit jednoduché úlohy na hlavních spojích, zatímco FPGA mohou zpracovávat komplexní úlohy u koncových serverů.

Systém N3IC [33] z roku 2022 je jedním z prvních řešení online klasifikace mimo CPU nebo GPU. N3IC akceleruje binární neuronovou síť přímo na FPGA *Virtex-7*. Autoři konkrétně použili třívrstvou plně propojenou síť (MLP) s binárními aktivacemi a váhami. Díky binární kvantizaci je možné maticové násobení nahradit hradly XOR a operacemi *pop-count* (viz sekce 2.1). Vstupem sítě jsou statistiky toku (celkový počet paketů a bajtů toku, průměrná velikost paketů apod.). Tyto statistiky jsou po přijetí každého nového paketu aktualizovány, konvertovány do binárního vektoru (256 b) a nakonec zpracovány binární neuronovou sítí.

Systém BoS (*Brain-on-Switch*) [40] z roku 2024 realizuje hybridní systém, který kombinuje programovatelný síťový přepínač *Barefoot Tofino 1* a GPU. Autorům se podařilo na síťovém přepínači implementovat rekurentní síť GRU, která klasifikuje většinu toků s velmi nízkou latencí. Toky, které nemají dostatečně vysokou jistotu klasifikace (max. 5 % všech toků), jsou klasifikovány větším modelem YaTC [41] na GPU. Vstupem GRU modelu jsou SPLT příznaky, zatímco vstupem YaTC modelu jsou počáteční bajty prvních 5 paketů toku. Latence modelu v síťovém přepínači je zanedbatelná (závisí na počtu použitých stupňů zřetěžené linky a frekvenci přepínače). Problematická je latence klasifikace na GPU, která se pohybuje v řádu sekund (max. 6 sekund).

Za zmínku stojí způsob, kterým byl model GRU kvantizován. Autoři totiž kvantizovali pouze aktivace modelu na binární hodnoty, zatímco parametry ponechali nezměněné. Binární kvantizace redukuje počet možných vstupních kombinací v každé vrstvě natolik, že výstupy lze předem předpočítat a uložit do vyhledávací tabulky. Každá vrstva modelu je pak jednoduše implementována vyhledávací tabulkou. Nicméně tento přístup je možný, pouze pokud má skrytý stav GRU sítě jednotky hodnot (autoři konkrétně použili 9 hodnot).

Systém *Kaleidoscope* [37] z roku 2024 je implementován v FPGA *Virtex UltraScale+*. Autoři implementovali obecný nástroj pro kvantizaci a implementaci neuronových sítí v FPGA. Pro kvantizaci aktivací i parametrů použili 8bitovou FX kvantizaci (konkrétně 1 bit pro znaménko, 2 bity pro celou část a 5 bitů pro desetinnou část). Pro implementaci modelů v FPGA použili temporální architekturu (viz sekce 2.5).

Zajímavostí je, že autoři navrhli dva odlišné klasifikátory pro různé typy síťového provozu. Pro zpracování krátkých toků s nízkým objemem přenesených dat (tzv. myší toky) použili menší plně propojenou síť (MLP) a pro zpracování dlouhých toků s větším objemem přenesených dat (tzv. sloní toky) použili přesnější konvoluční model (CNN).

Systém *Kaleidoscope* neuchovává žádnou stavovou informaci napříč pakety toku. První paket každého toku je vždy klasifikován MLP modelem, zatímco CNN model je použit pouze při obdržení 16. paketu daného toku. V obou případech je vstupem vždy prvních 64 bajtů z daných paketů.

Systém *FENIX* [10] z roku 2025 kombinuje programovatelný síťový přepínač *Tofino 2* a FPGA *Xilinx Zynq UltraScale+*. Pakety jsou předzpracovány na síťovém přepínači, na kterém se nachází i malý rozhodovací strom pro rychlou klasifikaci nových toků. Síťový přepínač ukládá SPLT příznaky pro posledních 9 paketů každého toku a na základě Token Bucket algoritmu je odesílá do FPGA. Zde jsou tyto toky klasifikovány pomocí přesnější konvoluční nebo rekurentní sítě (Vanilla RNN). Parametry i aktivace obou modelů jsou kvantizovány na 8 bitů a modely jsou implementovány pomocí prostorové architektury (viz sekce 2.5). Parametry modelu jsou dynamicky načteny z hostujícího serveru.

Srovnání existujících řešení

Srovnání výše uvedených řešení je uvedeno v tabulce 3.3. U každého řešení je uveden způsob jeho akcelerace, architektura, vstupní příznaky, bitová šířka kvantizace vah a aktivací, přesnost klasifikace nebo F1 skóre, latence, propustnost a počet parametrů modelu. Všechny metody byly ohodnoceny na stejné datové sadě ISCXVPN2016⁴. V tabulce je uvedena maximální propustnost síťového rozhraní, při které byl systém fyzicky otestován. Využití hardwarových zdrojů FPGA je uvedeno až v sekci 5.3, ve které jsou jednotlivá řešení srovnána s vlastní implementací.

Tabulka 3.3: Porovnání vybraných řešení online klasifikace šifrovaného síťového provozu na datové sadě ISCX-VPN-2016 [12]. Sloupce *S*, *F* a *G* označují, zda je systém akcelerován pomocí síťového přepínače (S), FPGA (F) nebo GPU (G). Následující sloupce obsahují architekturu modelu, typ vstupních příznaků, bitovou šířku kvantizace vah (*W*) a aktivací (*A*), přesnost klasifikace (*A*) nebo F1 skóre (F_1), latenci (*ns*), propustnost (*Gb/s*) a počet parametrů modelu. U řešení BoS je uvedena architektura modelu, která je akcelerována na síťovém přepínači. V případě systému FENIX je uveden model akcelerovaný v FPGA. Příznaky *Stat.* označují statistiky toku, *SPLT* označuje časové řady obsahující velikosti, časové rozestupy a směry paketů a *Raw* označuje surové bajty paketů. U systému Kaleidoscope je uvedena přesnost (*A*), jelikož autoři neuvádějí F1 skóre. U ostatních prací je uvedeno F1 skóre (F_1) z práce [10]. Uvedená latence představuje dobu zpracování jednoho paketu. Protože systém FENIX zpracovává 9 paketů současně, je v tabulce uvedena hodnota 9×255 ns (autoři konkrétně uvádějí celkovou latenci 2300 ns). Počet parametrů byl vypočten na základě informací a zdrojových kódů, které jednotlivé práce poskytují.

Práce	S	F	G	Arch.	Vstup	W	A	A / F_1	[ns]	Gb/s	Param.
N3IC	–	•	–	MLP	Stat.	1	1	73,8 % F_1	1 000	40 Gb/s	41 600
BoS	•	–	•	GRU	SPLT	*	1	86,3 % F_1	–	40 Gb/s	36 131
Kaleid.	–	•	–	MLP	Raw	8	8	92,9 % <i>A</i>	256	100 Gb/s	4 258
Kaleid.	–	•	–	CNN	Raw	8	8	96,1 % <i>A</i>	2 700	100 Gb/s	–
FENIX	•	•	–	RNN	SPLT	8	8	87,3 % F_1	9×255	800 Gb/s	36 325
FENIX	•	•	–	CNN	SPLT	8	8	89,2 % F_1	9×255	800 Gb/s	36 303

Nízké F1 skóre systému N3IC je způsobeno extrémní kvantizací parametrů i aktivací a také použitím statistik toku jako vstupních příznaků. Protože jsou tyto statistiky agregovány přes jednotlivé pakety toku, dochází ke ztrátě časových vzorů, které mohou být pro přesnou klasifikaci klíčové.

Specifický způsob kvantizace u systému BoS umožnil celý model implementovat přímo na síťovém přepínači. Systém byl sice prakticky otestován na propustnosti 40 Gb/s, ale v simulaci dosáhl propustnosti až 1,6 Tb/s. BoS dosáhl o 12,5 % vyššího F1 skóre než systém N3IC díky použití: (i) časových řad SPLT, které zachycují časové závislosti mezi pakety a (ii) přesnějšího modelu na GPU. Právě tento druhý model akcelerovaný na GPU může být kvůli své vysoké latenci a omezené propustnosti limitujícím faktorem celého systému.

Systém Kaleidoscope také kombinuje dva různě velké modely, avšak oba jsou akcelerovány přímo v FPGA. Menší model umožňuje rychlou klasifikaci většiny síťových toků, zatímco větší model je použit pro zpracování dlouhotrvajících toků s větším objemem pře-

⁴Systém N3IC sice nebyl v původní práci [33] vyhodnocen na datové sadě ISCXVPN2016, ale autoři systému FENIX toto vyhodnocení provedli ve svém článku [10].

nášených dat. Autoři však neuvádějí F1 skóre a neposkytují podrobnější informace o implementaci dílčích modelů.

Nakonec systém FENIX kombinuje předzpracování toků na síťovém přepínači a akceleraci modelů v FPGA. Díky použití mírnější 8 bitové kvantizace dosáhl model FENIX vyššího F1 skóre (89,2 %) než systémy BoS a N3IC. Použití síťového přepínače umožnilo dosáhnout propustnosti až 800 Gb/s, avšak hlavní limitací celého řešení zůstává propustnost a latence klasifikace v FPGA. Při vysokém počtu příchozích paketů nemusí být možné klasifikovat všechny síťové toky včas.

Kapitola 4

Návrh neuronové sítě

Cílem této kapitoly je nalezení takové neuronové sítě pro online klasifikaci šifrovaného provozu, která bude vhodným kandidátem pro implementaci v FPGA. Samotný způsob implementace v FPGA je uveden v následující kapitole 5.

Aktuální kapitola začíná stanovením požadavků (sekce 4.1), kritikou současného stavu online klasifikace šifrovaného provozu (sekce 4.2) a zdůvodněním volby konkrétní architektury neuronové sítě (sekce 4.3). Experimentální prostředí společné pro všechny navržené modely je popsáno v sekci 4.4. Zbývající části kapitoly se postupně věnují samotnému návrhu neuronové sítě. Nejprve byly ověřeny základní rekurentní modely (sekce 4.5), následuje ověření dodatečných vstupních a výstupních transformací (sekce 4.6), ověření vícevrstvých variant rekurentních sítí (sekce 4.7) a komprese stavové informace (sekce 4.8). Kapitulu uzavírá srovnání výsledného modelu s existujícími offline i online klasifikátory (sekce 4.9).

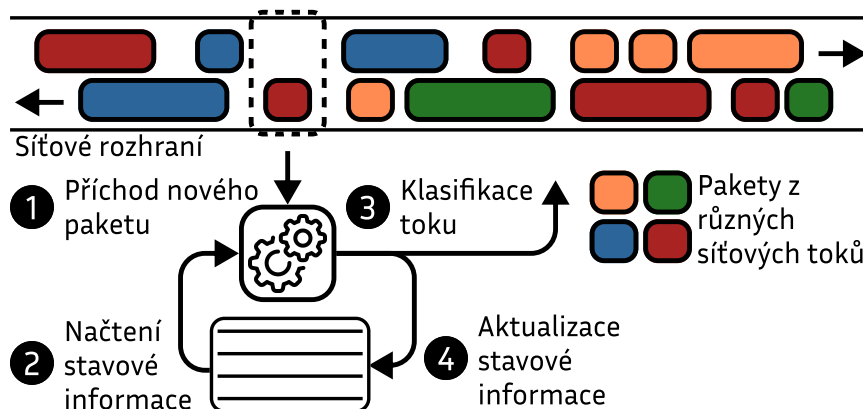
4.1 Specifikace požadavků

Online klasifikace síťových toků na vysokorychlostních linkách (100 Gb/s a více) klade vysoké nároky na propustnost a latenci klasifikačního modelu. Aby bylo možné provést dostatečný počet klasifikací, musí mít navržená neuronová síť nízkou výpočetní složitost.

Výpočetně nejnáročnější části typických modelů jsou lineární vrstvy. Jejich složitost závisí na počtu obsažených parametrů, způsobu využití těchto parametrů a dalších vlastnostech jejich implementace (např. způsob kvantizace). V této kapitole bude výpočetní složitost modelů reflektována převážně počtem parametrů.

Další komplikací je skutečnost, že pakety různých toků se na síťovém rozhraní prolínají a přicházejí s předem neznámými časovými rozestupy (viz obrázek 4.1). Aby klasifikátor mohl zohlednit historii konkrétního toku, je nutné pro každý aktivní tok udržovat dodatečnou stavovou informaci. Jelikož mohou být současně aktivní tisíce až miliony toků, je velikost této stavové informace klíčová.

Užitečnou vlastností je tzv. *inkrementální klasifikace*. Klasifikátor disponující touto vlastností je schopen zpřesnit dosavadní predikci s příchodem každého nového paketu, a to bez nutnosti opakované analýzy předchozích paketů. Výpočetní složitost takového klasifikátoru je rozdělena do menších kroků, z nichž každý představuje zpracování jednoho paketu.



Obrázek 4.1: Schematické znázornění přenosu paketů skrze síťové rozhraní a princip jejich klasifikace. Aby bylo možné zohlednit i historii daného toku, je nutné pro každý aktivní tok udržovat dodatečnou stavovou informaci.

Požadavky na navrženou neuronovou síť jsou tedy následující: (i) nízký počet parametrů, (ii) malá velikost stavové informace, (iii) vysoké F1 skóre klasifikace¹ a (iv) schopnost inkrementální klasifikace.

4.2 Kritika současného stavu

Hlavním problémem stávajících metod je správa stavové informace a s tím související schopnost inkrementální klasifikace. V případě řešení N3IC [33] představují stavovou informaci statistiky toku (256 bitů), které současně slouží jako vstupní příznaky klasifikátoru. Při přijetí nového paketu se tyto statistiky aktualizují a na jejich základě se znovu provede klasifikace toku. Tento systém tedy disponuje schopností inkrementální klasifikace, avšak F1 skóre jeho klasifikace je nízké (73,8 %) kvůli ztrátě časových vzorů v agregovaných statistikách.

Systém Kaleidoscope [37] neuchovává žádnou stavovou informaci, protože síťové toky klasifikuje vždy pouze podle aktuálního paketu. Tento systém sice disponuje schopností inkrementální klasifikace, ale jen díky tomu, že nezohledňuje žádnou historii toku. Vstupní příznaky jsou tvořeny prvními 64 bajty z daného paketu. Použití surových bajtů má následující nevýhody: (i) potenciálně vyšší výpočetní složitost modelu kvůli většímu objemu vstupních dat a (ii) možnost přeučení modelu na konkrétní trénovací data při neodstranění některých polí (např. IP adresy, porty atd.).

Zbylé dva systémy BoS [40] a FENIX [10] používají příznaky SPLT, ale pouze z omezeného počtu posledních paketů. V případě systému BoS je to posledních 8 paketů a v případě systému FENIX posledních 9 paketů. Omezení toků na pevný počet paketů znamená ztrátu informací, které by mohly být využity k přesnější klasifikaci. S tím také souvisí fenomén tzv. sloních toků (*elephant flows*). Sloní toky tvoří pouze malé procento všech toků v síti, ale obsahují dlouhé sekvence paketů, a také většinu přenášených dat [37]. Omezení dlouhých sekvencí paketů nejvíce ovlivní právě tuto skupinu toků, u kterých je správná klasifikace velmi důležitá.

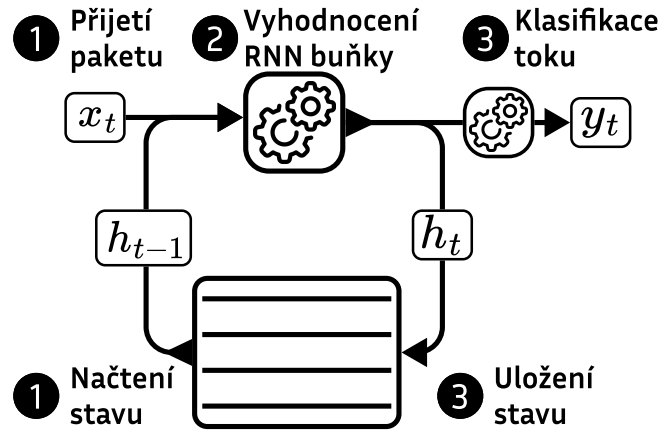
¹Jelikož je použita klasifikace do více tříd, je výsledné F1 skóre vypočteno jako průměr z F1 skóre dílčích tříd (tzv. *macro-averaged F1 score*).

Řešení BoS a FENIX inkrementální klasifikaci nepodporují. Systém BoS je kvůli omezenému *read-modify-write* přístupu k paměti síťového přepínače vždy znovu vyhodnocen na posledních 8 paketech toku. V případě systému FENIX je stav toku uchovávan v síťovém přepínači a pouze v určitých okamžicích jsou příznaky posledních 9 paketů odeslány do FPGA.

4.3 Volba architektury

Problémy zmíněné v předchozí sekci 4.2 je možné vyřešit použitím rekurentní architektury neuronové sítě (RNN). Hlavním principem těchto modelů je inkrementální úprava stavové informace. Stav je reprezentován vektorem hodnot označovaným jako tzv. *skrytý stav* (angl. *hidden state*). Úprava stavu je definována tzv. *rekurentní buňkou*.

Princip použití RNN pro online klasifikaci síťového provozu je zobrazen na obrázku 4.2. Příznaky nového paketu jsou společně s načteným skrytým stavem předány do rekurentní buňky. Výstupem rekurentní buňky je aktualizovaný skrytý stav, který je uložen zpět do příslušné paměti. Konkrétní klasifikovaná třída nebo pravděpodobnosti jednotlivých tříd jsou následně vypočteny podle aktualizovaného skrytého stavu.



Obrázek 4.2: Schematické znázornění rekurentní neuronové sítě použité pro online klasifikaci síťového provozu. Kroky označené stejným číslem mohou být prováděny paralelně.

Výhody RNN architektury jsou následující: (i) predikci lze zpřesňovat inkrementálně s příchodem každého nového paketu, (ii) velikost stavu je konstantní a nezávislá na počtu zpracovaných paketů a (iii) lze zpracovat variabilní počet paketů. Je vhodné poznamenat, že práce BoS a FENIX sice použily RNN pro klasifikaci síťového provozu, ale žádná z nich nepoužila RNN v inkrementálním režimu.

Jako vstup RNN modelů byly zvoleny časové řady SPLT (velikosti paketů, časové rozestupy a jejich směry). Důvody této volby jsou následující: (i) na rozdíl od statistik toku je v příznacích SPLT zachován časový rozměr, (ii) na rozdíl od surových bajtů paketů obsahují příznaky SPLT pouze tři hodnoty na každý paket, (iii) neobsahují citlivá pole (IP adresy, porty atd.), která by mohla způsobit přeučení modelu na konkrétní topologii sítě a (iv) největší veřejné datové sady (z pohledu počtu toků) obsahují právě příznaky SPLT.

4.4 Volba experimentálního prostředí

K hodnocení a vzájemnému porovnání navržených modelů byla zvolena veřejně dostupná datová sada CESNET-QUIC22. Z datových sad porovnaných v sekci 3.2 obsahuje CESNET-QUIC22 největší množství síťových toků (konkrétně 153 milionů). Je také důležité, že obsahuje síťové toky z reálného síťového provozu. Jelikož se v této datové sadě vyskytuje významný datový drift v prvním a druhém týdnu [23], bude pro trénování modelů použit až třetí týden. Na základě doporučení z práce [19] bude použito chronologické rozdělení trénovací a testovací množiny. Testování tedy bude probíhat na posledním čtvrtém týdnu datové sady.

Vstupem navrhovaných modelů jsou příznaky SPLT označované jako $X = (x_1, \dots, x_T)$. Ve zvolené datové sadě jsou tyto příznaky omezeny na prvních 30 paketů každého toku ($T = 30$). V každém časovém kroku t vstupují do modelu příznaky $x_t \in \mathbb{R}^3$, které odpovídají nově přijatému paketu. Vektor x_t konkrétně obsahuje velikost paketu, časový rozestup od předchozího paketu a směr paketu.

Všechny navrhované modely klasifikují síťové toky do jedné ze 102 webových aplikací (např. alza-www, alza-identity, google-www, google-colab atd.). Výstupní vektor $y_t \in \mathbb{R}^{102}$ v časovém kroku t obsahuje pravděpodobnosti příslušnosti k jednotlivým třídám (webovým aplikacím). Ačkoliv jsou všechny modely trénovány pro klasifikaci v každém časovém kroku, výsledné F1 skóre je vyhodnoceno na základě predikcí v posledním časovém kroku $t = 30$.

Také je vhodné zdůraznit, že u rekurentních modelů bude zobrazován výpočetní graf pouze pro jeden časový krok t (tedy pouze jedna rekurentní buňka). Pro zpracování T paketů bude výpočetní graf rozbalen T -krát. Červená barva bude označovat rekurentní skrytý stav h_t propojující jednotlivé časové kroky. Modely byly implementovány v nástroji PyTorch a trénovány na GPU NVIDIA GeForce RTX 5060 Ti. Pro zrychlení trénování a inference na GPU se osvědčilo použití funkce `torch.compile()`, která navržený model zkompiluje do optimalizovaných CUDA kernelů. Trénování probíhalo po dobu 20 epoch pomocí optimalizačního algoritmu AdamW a *StepLR* plánovače. Pro práci s datovou sadou byl použit nástroj CESNET DataZoo².

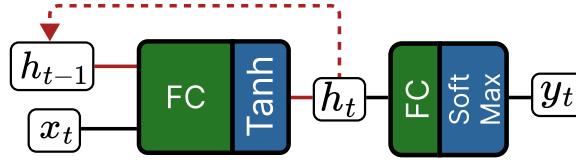
4.5 Základní rekurentní modely

Počáteční experiment ověřuje schopnosti základních rekurentních sítí. Jedna z nejjednodušších rekurentních sítí se označuje jako tzv. *Vanilla RNN*. Vstupní vektor x_t a předchozí skrytý stav h_{t-1} jsou konkatenovány, transformovány plně propojenou lineární vrstvou a poté nelineární aktivační funkcí. Nejčastěji je použita aktivační funkce $\tanh$, která hodnoty skrytého stavu h_t omezuje do rozmezí $(-1, 1)$. Výstupní pravděpodobnosti y_t jsou ze skrytého stavu h_t vypočteny pomocí dodatečné lineární vrstvy a softmax funkce.

Vanilla RNN je znázorněna na obrázku 4.3 a rovnicí 4.1. Parametry rekurentní buňky jsou označeny W a b , zatímco parametry výstupní vrstvy jsou označeny W_o a b_o . Je vhodné zdůraznit, že parametry W , b , W_o a b_o jsou sdílené napříč všemi časovými kroky t . Právě díky tomu lze zpracovat sekvence variabilní délky.

$$\begin{aligned} h_t &= \tanh \left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b \right) \\ y_t &= \text{softmax}(W_o h_t + b_o) \end{aligned} \tag{4.1}$$

²<https://github.com/CESNET/cesnet-datazoo>



Obrázek 4.3: Základní rekurentní síť (tzv. Vanilla RNN).

Hlavním problémem základní Vanilla RNN je propagace gradientů během učení. Gradienty během zpětné propagace musí procházet skrze nelineární aktivační funkce napříč všemi časovými kroky. To způsobuje buď mizení těchto gradientů (*vanishing gradients*), nebo naopak jejich extrémní zesílení (*exploding gradients*).

Problémy s propagací gradientů lze z velké části vyřešit pomocí tzv. *reziduálních spojení*. Namísto úplného nahrazení předchozího stavu h_{t-1} novým stavem h_t je výsledný stav vypočten jako lineární kombinace předchozího stavu h_{t-1} a nového kandidátního stavu $\tilde{h}_t$. Reziduální připojení předchozího stavu umožňuje gradientům prostupovat i jinou cestou než skrze nelineární aktivační funkce. Nejznámější rekurentní sítě využívající reziduální spojení jsou GRU (*Gated Recurrent Unit*) a LSTM (*Long Short-Term Memory*).

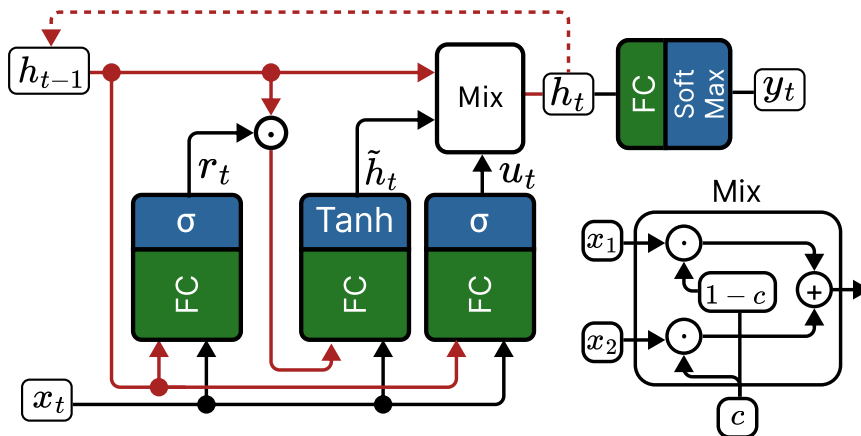
Sítě GRU a LSTM rozšiřují princip reziduálních spojení o tzv. brány (angl. *gates*), které umožňují dynamicky řídit hlavní datový tok uvnitř rekurentní buňky. Brány jsou implementovány pomocí lineární vrstvy se sigmoidní aktivační funkcí. Vstupem těchto bran je jak aktuální vstup x_t , tak i předchozí stav h_{t-1} .

Uvnitř GRU se kromě bloku pro výpočet kandidátního stavu $\tilde{h}_t$ nacházejí brány: reset gate $r_t \in \langle 0, 1 \rangle^H$ a update gate $u_t \in \langle 0, 1 \rangle^H$. Člen r_t určuje, do jaké míry bude každý člen minulého skrytého stavu zeslaben, zatímco u_t řídí výpočet h_t . Nový stav h_t je vypočten dynamicky váženým součtem kandidátního stavu $\tilde{h}_t$ a předchozího stavu h_{t-1} . Předchozí stav je vážený členem u_t , zatímco kandidátní stav je vážený členem $(1 - u_t)$. Tento vážený součet je v této práci označen jako tzv. *mix blok*. Celý GRU model je znázorněn na obrázku 4.4 a pomocí rovnic 4.2.

$$\begin{aligned}
 r_t &= \sigma \left(W_r \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_r \right) \\
 u_t &= \sigma \left(W_u \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_u \right) \\
 \tilde{h}_t &= \tanh \left(W_n \begin{bmatrix} x_t \\ r_t \odot h_{t-1} \end{bmatrix} + b_n \right) \\
 h_t &= (1 - u_t) \odot \tilde{h}_t + u_t \odot h_{t-1} \\
 y_t &= \text{softmax}(W_o h_t + b_o)
 \end{aligned} \tag{4.2}$$

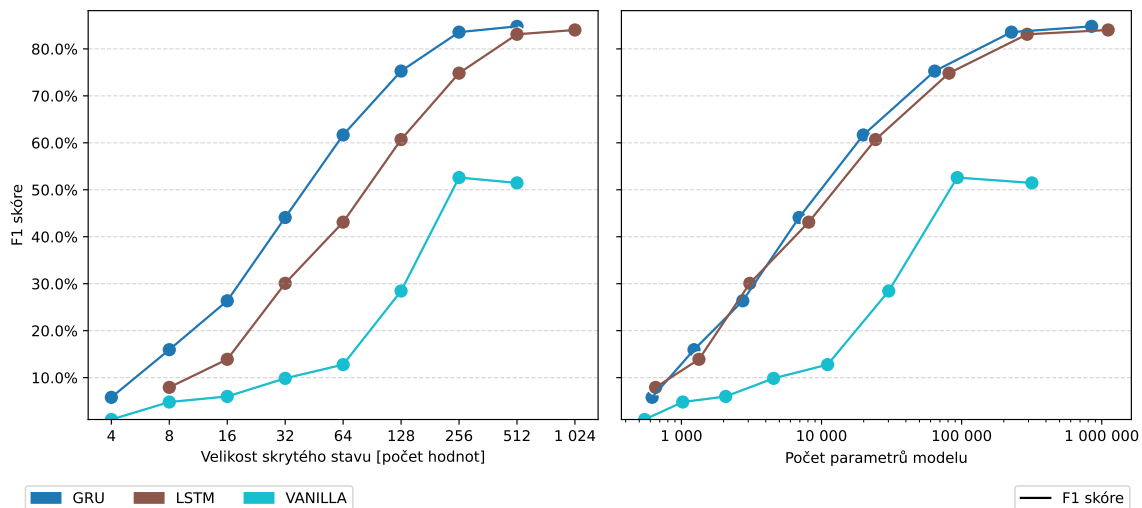
Nejdůležitějším parametrem rekurentních sítí je velikost skrytého stavu h , která určuje množství informace předávané napříč časovými kroky a zároveň přímo ovlivňuje počet parametrů modelu. Větší stav sice umožňuje dosáhnout vyšší přesnosti klasifikace, avšak za cenu vyšších paměťových nároků a kvadratického nárůstu počtu parametrů rekurentní buňky.

Obrázek 4.5 zobrazuje F1 skóre základních rekurentních modelů (Vanilla RNN, GRU a LSTM) v závislosti na velikosti skrytého stavu h . Nejvyššího F1 skóre dosahuje model GRU, který při velikosti stavu 512 hodnot dosahuje F1 skóre 84,8 %. Odpovídající LSTM model vyžaduje o 31,27 % více parametrů a dosahuje o 0,77 % nižšího F1 skóre. Navíc stav



Obrázek 4.4: Schematické znázornění rekurentní buňky GRU (Gated Recurrent Unit).

LSTM modelu je tvořen dvojicí vektorů (*long-term memory* a *short-term memory*). Protože oba vektory mají velikost 512 hodnot, je celkový stav dvojnásobný.



Obrázek 4.5: Srovnání 3 základních rekurentních modelů: Vanilla RNN, GRU a LSTM. Vertikální osa obou grafů zobrazuje F1 skóre. Horizontální osa levého grafu zobrazuje velikost skrytého stavu modelu, zatímco u pravého grafu zobrazuje počet parametrů modelu. Obě horizontální osy mají logaritmické měřítko (levá má základ 2, zatímco pravá má základ 10).

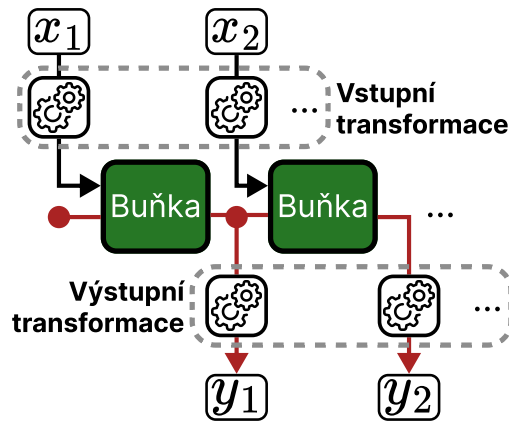
4.6 Předzpracování vstupních a výstupních příznaků

Řada existujících prací použila ve svých modelech dodatečné vrstvy pro předzpracování vstupních nebo výstupních příznaků [10, 40, 22]. Cílem této sekce je tedy ověřit, jakým způsobem tyto dodatečné vrstvy změní vlastnosti základních rekurentních modelů. Umístění těchto vrstev je zobrazeno na obrázku 4.6.

Vrstvy umístěné před buňkou GRU (tzv. *vstupní transformace*) slouží k předzpracování vstupních příznaků x_t . Každý vektor x_t konkrétně obsahuje velikost balíku x_t^{size} ,

časový rozestup x_t^{ipt} a směr paketu x_t^{dir} . Výstupem rekurentní buňky je nový skrytý stav h_t , ze kterého je následně vypočten výstupní vektor pravděpodobností y_t . Protože počet klasifikovaných tříd může být jiný než velikost skrytého stavu, je vhodné model rozšířit o dodatečné výstupní vrstvy (*výstupní transformace*). Již v předchozí sekci 4.5 byla použita výstupní transformace (konkrétně se jednalo o plně propojenou lineární vrstvu a softmax funkci).

Důležitou vlastností těchto dodatečných transformací je jejich nezávislost napříč časovými kroky. Díky tomu nepotřebují uchovávat žádnou stavovou informaci napříč jednotlivými pakety toku. Stejně jako parametry buňky GRU, jsou i parametry těchto vrstev sdílené napříč všemi časovými kroky.



Obrázek 4.6: Umístění dodatečných vstupních a výstupních transformací. Je důležité upozornit, že parametry těchto transformací i rekurentní buňky jsou sdíleny napříč všemi časovými kroky. Vstupní transformace slouží k předzpracování vstupních příznaků x_t . Výstupní vrstvy transformují nový skrytý stav h_t na pravděpodobnosti klasifikovaných tříd y_t .

Vstupní embedding tabulky

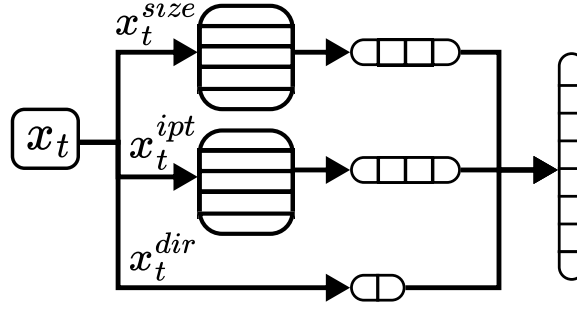
Efektivní způsob předzpracování vstupních příznaků (zejména diskretních³) představují tzv. *embedding tabulky*. Embedding tabulka se skládá z embedding vektorů, které jsou adresovány konkrétním vstupním příznakem (velikost paketu nebo časový rozestup).

Obsažené embedding vektory je možné učit spolu s ostatními parametry modelu, ale v takovém případě musí být embedding vrstva hned první vrstvou neuronové sítě. Adresování konkrétního embedding vektoru v tabulce je totiž nediferencovatelná operace, která by znemožnila propagování gradientů a tedy i učení všech předchozích vrstev modelu⁴.

Konkrétně byly zvoleny embedding tabulky použité v modelu *30pktTCNET* [22], který byl v sekci 3.3 zařazen mezi nejlepší offline klasifikátory síťového provozu. Velikosti paketů x_t^{size} a časové rozestupy x_t^{ipt} jsou předzpracovány dvěma embedding tabulkami, zatímco směr paketu x_t^{dir} je kódován pomocí one-hot kódování do vektoru se dvěma hodnotami. Embedding vektory pro kódování velikostí paketů mají velikost 20 hodnot, zatímco vektory pro kódování časových rozestupů mají velikost 10 hodnot. Tento způsob kódování SPLT příznaků je souhrnně označován jako *embedding blok* (viz obrázek 4.7).

³Příznaky SPLT obsahují celočíselné velikosti paketů, celočíselné časové rozestupy mezi pakety a dva možné směry paketů (příchozí a odchozí).

⁴Při zavedení určitých aproximací je možné gradienty propagovat skrze embedding tabulky. V takovém případě se spíše jedná o učitelnou vektorovou kvantizaci, která bude dále popsána v sekci 4.8.



Obrázek 4.7: Schematické znázornění embedding bloku pro předzpracování vstupních příznaků. Embedding blok je složen ze 2 embedding tabulek pro kódování velikostí paketů x_t^{size} a časových rozestupů x_t^{ipt} . Směr paketu x_t^{dir} je kódován např. pomocí one-hot kódování do vektoru se dvěma hodnotami.

Velikost paketů může nabývat hodnot od 0 do 1500 bajtů⁵. Protože autoři přiřadili každé velikosti paketu vlastní embedding vektor, obsahuje příslušná embedding tabulka $1501 \cdot 20 = 30\,020$ učitelných parametrů. Časové rozestupy mohou nabývat výrazně většího množství hodnot. Konkrétně v použité datové sadě CESNET-QUIC22 se jedná o celočíselný rozsah od 0 do 65 000 milisekund. Autoři modelu 30pktTCNET proto časové rozestupy nejprve kvantizují do 200 úrovní a teprve tyto úrovně jsou mapovány na embedding vektory. Konkrétně byla zvolena neuniformní kvantizace, jejíž intervaly byly vypočteny na základě kvantilů z příslušných trénovacích dat. Příslušná embedding tabulka tedy celkově obsahuje $200 \cdot 10 = 2\,000$ učitelných parametrů.

Podstatná je inicializace embedding tabulek. Autoři modelu 30pktTCNET k tomuto účelu použili tzv. PLE (*Piecewise Linear Embedding*) kódování představené v práci [14]. Toto kódování je podrobněji popsáno v následující části textu.

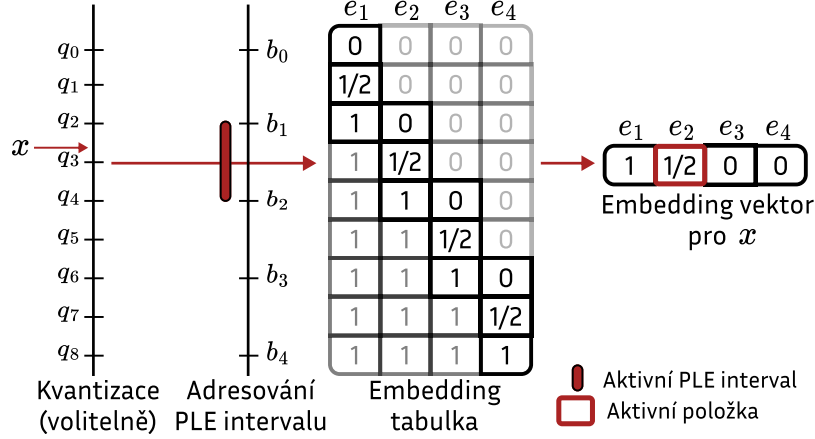
PLE kódování

PLE kódování přiřazuje každé vstupní hodnotě x embedding vektor $e = (e_1, e_2, \dots, e_B)$ podle rovnice 4.3. Rozsah vstupní veličiny x je rozdělen do B intervalů, které budou dále označeny jako tzv. *PLE intervaly*. Každá položka embedding vektoru e_i odpovídá jednomu PLE intervalu. Hranice PLE intervalů $b_0, b_1, \dots, b_B$ mohou být zvoleny například uniformně nebo na základě kvantilů vstupních dat (podobně jako kvantizace časových rozestupů). Konkrétní PLE interval, do kterého hodnota x spadá, označme jako tzv. *aktivní PLE interval* a odpovídající položku vektoru e_i jako tzv. *aktivní položku*.

$$e_i = \begin{cases} 0 & \text{pro } b_{i-1} > x \\ 1 & \text{pro } b_i \leq x \\ \frac{x-b_{i-1}}{b_i-b_{i-1}} & x \in \langle b_{i-1}, b_i \rangle \end{cases} \quad (4.3)$$

Aktivní položka e_i je získána transformací hodnoty x do rozsahu $\langle 0, 1 \rangle$. Tato hodnota odpovídá pozici hodnoty x v daném PLE intervalu. Všechny předcházející položky $e_1, \dots, e_{i-1}$ jsou nastaveny na hodnotu 1 a všechny následující položky $e_{i+1}, \dots, e_B$ jsou nastaveny na hodnotu 0. Příklad PLE inicializace embedding tabulky zobrazuje obrázek 4.8.

⁵Velikost paketů je shora omezena tzv. MTU (*Maximum Transmission Unit*), jejíž hodnota je v běžných sítích rovna 1500 bajtům.



Obrázek 4.8: Příklad PLE (*Piecewise Linear Embedding*) inicializace embedding tabulky. Vstupní hodnota x je kódována do vektoru $e = (e_1, e_2, \dots, e_B)$ podle rovnice 4.3. Pro redukcii počtu embedding vektorů uvnitř tabulky lze vstupní veličinu kvantizovat do hladin $q_0, q_1, \dots, q_E$. Následně bude mít každá hladina přiřazen právě jeden embedding vektor. Některé kvantizační hladiny reprezentují hranice PLE intervalů $b_0, b_1, \dots, b_B$ a každý PLE interval odpovídá jedné položce ve výsledném embedding vektoru. Pokud například hodnota x náleží do intervalu (b_{i-1}, b_i) (což odpovídá položce vektoru e_i), pak jsou předcházející položky $e_1, \dots, e_{i-1}$ nastaveny na hodnotu 1 a následující položky $e_{i+1}, \dots, e_B$ na hodnotu 0. Samotná položka e_i obsahuje pozici kvantizované hodnoty x v daném PLE intervalu. V tomto příkladě je použita uniformní kvantizace, která každému PLE intervalu přiřazuje 3 diskrétní hladiny. Proto může pozice x v daném intervalu nabývat pouze hodnot 0, 0,5 a 1.

Optimalizace embedding tabulek

Autoři modelu 30pktTCNET [22] zjistili, že embedding tabulka s PLE inicializací umožňuje dosáhnout vysoké přesnosti klasifikace i bez dalšího trénování embedding vektorů. Na základě tohoto pozorování jsem navrhl optimalizaci FPGA implementace embedding tabulky. Namísto ukládání celé tabulky s 32 020 parametry lze v FPGA vytvořit výpočetní blok, který na základě PLE kódování dynamicky sestaví embedding vektor e přímo pro konkrétní vstupní hodnotu x .

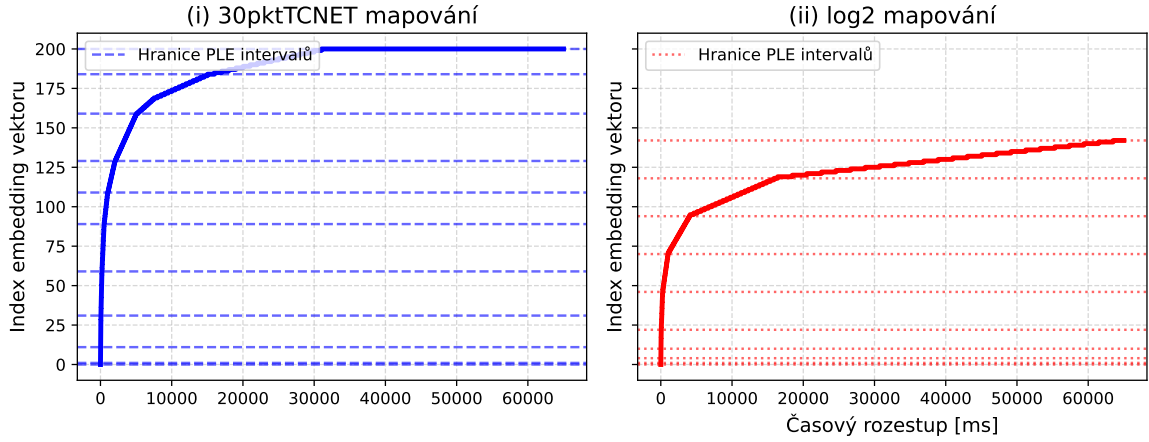
Aby bylo možné sestavit embedding vektor e , je nutné najít aktivní PLE interval a určit pozici hodnoty x uvnitř něj. Tento interval lze určit pomocí porovnání s hranami PLE intervalů $b_0, b_1, \dots, b_B$, nicméně pro ušetření výpočetních zdrojů FPGA byla zavedena optimalizace adresování PLE intervalu.

Pokud se šířka PLE intervalů omezí na mocniny dvou ($\Delta = 2^k$), bude možné PLE interval i určit pomocí bitového posunu $i = x \gg k$. Bity vysunuté bitovým posunem ($r = x \& (\Delta - 1)$) určují pozici hodnoty x v daném intervalu. Aktivní prvek výsledného vektoru je roven $e_i = r/\Delta$, avšak pokud budou aktivace kvantizovány do celočíselných hodnot, lze použít přímo $e_i = r$. Následně je nutný obvod, který nastaví indexy $1, \dots, i-1$ na hodnotu 1 a zbývající indexy $i+1, \dots, B$ na hodnotu 0.

Pro embedding tabulku příslušející velikosti paketů lze první optimalizaci aplikovat přímo. Maximální velikost balení byla zaokrouhlena na 1536 bajtů (z původních 1500 bajtů) a byly zvoleny embedding vektory o velikosti 24 hodnot. Tato velikost embedding vektorů

je nejbližší původní velikosti 20 hodnot a zároveň splňuje omezení velikosti PLE intervalů na mocniny dvou ($\Delta = 64$).

Situace je komplikovanější v případě časových rozestupů, které jsou nejprve kvantizovány do 200 neuniformních úrovní. Experimentálně byly nalezeny kvantizační úrovně (a tedy i PLE intervaly), které umožňují efektivní implementaci v FPGA a zároveň se podobají původním intervalům (viz obrázek 4.9(ii)). Za pravé okraje PLE intervalů byly zvoleny sudé mocniny dvou (konkrétně 2^{2n} pro $n = 0, 1, \dots, 8$). Velikost embedding vektorů je tedy 9 namísto původních 10 hodnot. Původní intervaly z práce [22] používají vyšší rozlišení pro kratší časové rozestupy a nižší rozlišení pro delší časové rozestupy. Proto byly pro velikosti kvantizačních úrovní uvnitř jednotlivých PLE intervalů zvoleny následující zvyšující se mocniny dvou: $2^0, 2^1, 2^2, 2^3, 2^5, 2^7, 2^9, 2^{11}$. První interval obsahuje stejně jako v původní práci pouze jednu kvantizační úroveň s hodnotou 0 ms.



Obrázek 4.9: Srovnání kvantizačních úrovní časových rozestupů pro: (i) původní metodu z práce [22] a (ii) nově navrženou optimalizaci umožňující snazší implementaci v FPGA.

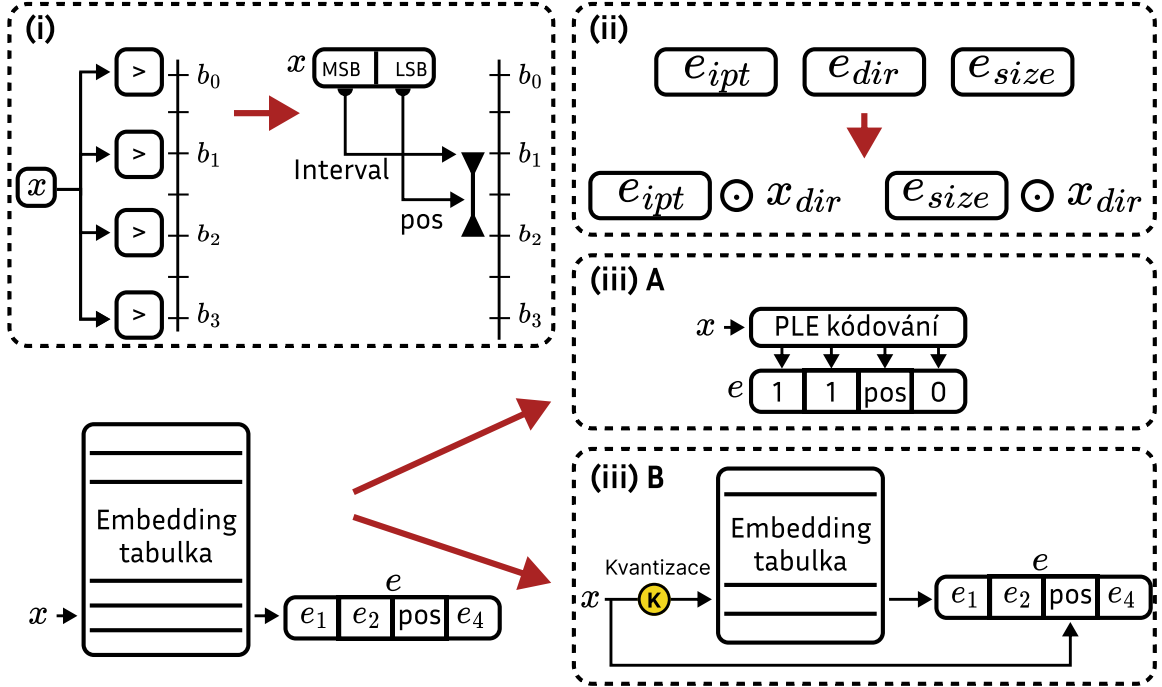
Jelikož jsou pravé okraje PLE intervalů umístěny podle 2^{2n} , je možné pro hodnotu x určit příslušný PLE interval pomocí $i = \lfloor \log_2(x)/2 \rfloor$. Operaci $\log_2(x)$ lze implementovat v FPGA pomocí prioritního dekodéru, který nalezne nejvyšší nastavený bit ve vstupní hodnotě x . Pozici hodnoty x v daném PLE intervalu lze určit pomocí $p = (x \gg s) - b$, kde s a b jsou konstanty pro každý PLE interval.

V práci [22] je směr paketu kódován pomocí one-hot kódování do vektoru se dvěma hodnotami. V této práci byla použita drobná optimalizace, která směr paketu kóduje jako znaménko embedding vektorů velikostí paketů a časových rozestupů. To umožňuje zmenšit velikost embedding vektorů o 2 hodnoty. Protože rekurentní buňky obsahují plně propojené vrstvy, je touto optimalizací možné ušetřit stovky propojů, a tedy i parametrů. Při použití GRU s velikostí stavu 64 hodnot dojde konkrétně k ušetření 384 parametrů.

Optimalizace adresování PLE intervalů je označena jako optimalizace (i) (viz obrázek 4.10(i)). Kódování směru paketu jako znaménka embedding vektorů je označeno jako optimalizace (ii) (viz obrázek 4.10(ii)). Nahrazení celé embedding tabulky fixním PLE kódováním je označeno jako optimalizace (iii)A (viz obrázek 4.10(iii)A).

Pro optimalizaci (iii) byla navržena i alternativní varianta (viz obrázek 4.10(iii)B), která kombinuje fixní PLE kódování a z části učitelnou embedding tabulku. Počet embedding vektorů je redukován pomocí kvantizace velikostí paketů. Aby nedošlo ke ztrátě informace, je aktivní položka embedding vektoru e_i vypočtena pomocí fixního PLE kódování. Přesná

pozice hodnoty x v daném PLE intervalu je tedy reflektována položkou e_i . Tuto optimalizaci lze také chápat tak, že hodnoty, které byly v PLE kódování fixní (0 nebo 1), jsou nyní učitelné parametry, zatímco aktivní prvek je vypočten PLE kódováním.



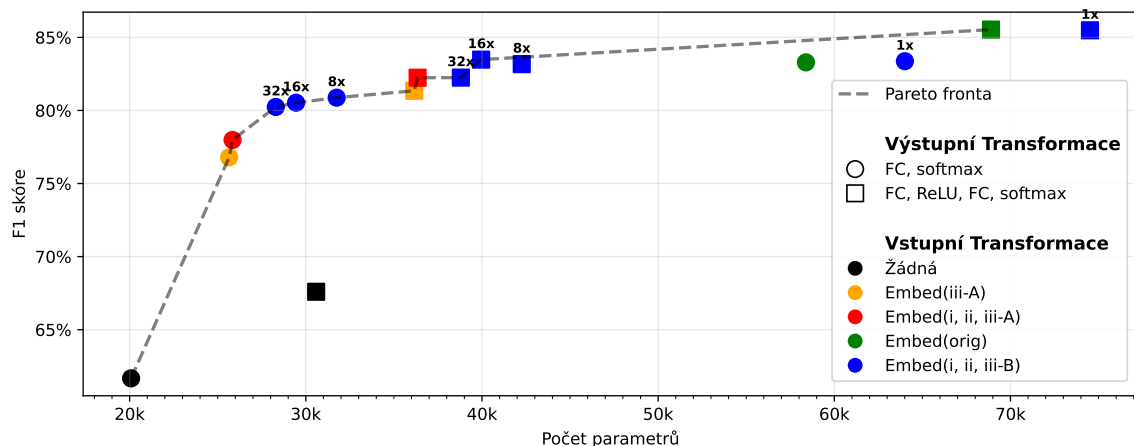
Obrázek 4.10: Srovnání navrhovaných optimalizací embedding tabulky s PLE inicializací. Optimalizace (i) zjednodušuje vyhledání PLE intervalu tím, že omezuje šířku PLE intervalů na mocniny dvou. Díky tomu je možné PLE interval určit pomocí bitového posunu místo porovnání s hranami intervalů. Optimalizace (ii) kóduje směr paketu jako znaménko embedding vektorů velikostí paketů a časových rozestupů. Optimalizace (iii) slouží k redukci počtu parametrů embedding tabulky. Optimalizace (iii)A dokonce nemusí embedding tabulku vůbec ukládat do FPGA, jelikož jsou embedding vektory sestavovány dynamicky pro konkrétní vstupní hodnotu x .

Vyhodnocení

Obrázek 4.11 srovnává různé konfigurace embedding bloku a 2 varianty výstupních transformací. U všech konfigurací je použita síť GRU se stavem o velikosti 64 hodnot, a proto jsou řešení porovnávána pouze podle počtu parametrů modelu (ne podle velikosti stavu).

Kruhové body označují základní výstupní transformaci, která byla použita v předchozí sekci 4.5 (plně propojená lineární vrstva následovaná funkcí softmax). Čtvercové body rozšiřují tuto transformaci o další vrstvu. Konkrétně se osvědčilo použít dodatečnou plně propojenou vrstvu následovanou ReLU aktivační funkcí. Přidání těchto vrstev vedlo ke zvýšení F1 skóre základní sítě GRU o 5,92 %, ale také ke zvýšení počtu parametrů modelu o 10 710.

Transformace vstupních příznaků pomocí embedding tabulek s PLE inicializací jsou v legendě označeny jako `Embed(...)`. Konfigurace `Embed(orig)` označuje původní konfiguraci embedding tabulek z modelu 30pktTCNET, která obsahuje 32 020 parametrů. Ostatní konfigurace označují různé kombinace již zmíněných optimalizací (použité optimalizace jsou uvedeny v závorkách). Číslice u modrých bodů příslušící optimalizaci (iii)B udávají faktor



Obrázek 4.11: Dosažené F1 skóre a počet parametrů sítě GRU (pro stav o velikosti 64 hodnot) při použití různých transformací vstupu (barvy bodů) a výstupních transformací (tvar bodů). Jednotlivé transformace jsou popsány v legendě. Nejlepší řešení není jednoznačné, ale závisí na maximálním přípustném počtu parametrů pro danou aplikaci. Proto jsou zvýrazněny body na Pareto frontě (přerušovaná čára), které pro daný počet parametrů dosahují nejlepšího F1 skóre.

redukce počtu embedding vektorů uvnitř tabulky (8x, 16x, ...). V případě faktoru 1x obsahuje embedding tabulka učitelny embedding vektor pro každou velikost paketu.

Zatímco varianta bez embedding bloku dosahuje F1 skóre 61,67 %, všechny konfigurace používající embedding blok dosahují F1 skóre nad 76 %. Optimalizace (iii)A, která nevyžaduje uložení embedding tabulky do paměti FPGA, dosahuje F1 skóre 81,35 % (pouze o 4,18 % méně než konfigurace Embed(orig)). Tato konfigurace zvyšuje počet parametrů pouze v rekurentní buňce GRU v důsledku většího rozměru embedding vektorů.

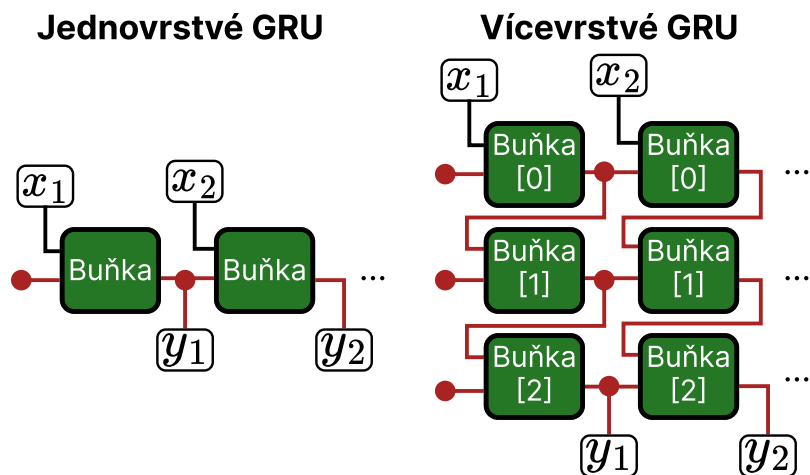
Optimalizace (i) a (ii) změny velikosti embedding vektorů. Pro velikosti paketů jsou použity vektory o velikosti 24 hodnot (místo původních 20 hodnot) a pro časové rozestupy vektory o velikosti 9 hodnot (místo původních 10 hodnot). Tato změna přináší mírné zvýšení počtu parametrů, ale i mírné zvýšení F1 skóre. Je vhodné poznamenat, že tyto konfigurace navíc umožní snížit množství FPGA zdrojů potřebných pro implementaci adresování PLE intervalů.

Nejvyššího F1 skóre, ale i nejvyššího počtu parametrů, dosahují konfigurace, ve kterých je celá embedding tabulka učitelná (Embed(orig) nebo Embed(i, ii, iii-B) s faktorem 1x). Poměr mezi F1 skóre a počtem parametrů je možné díky optimalizaci (iii) regulovat (viz modré body). Zajímavé je, že konfigurace s faktorem 16x konzistentně dosahuje lepšího F1 skóre než konfigurace se sousedními faktory 8x a 32x.

4.7 Vícevrstvé rekurentní buňky

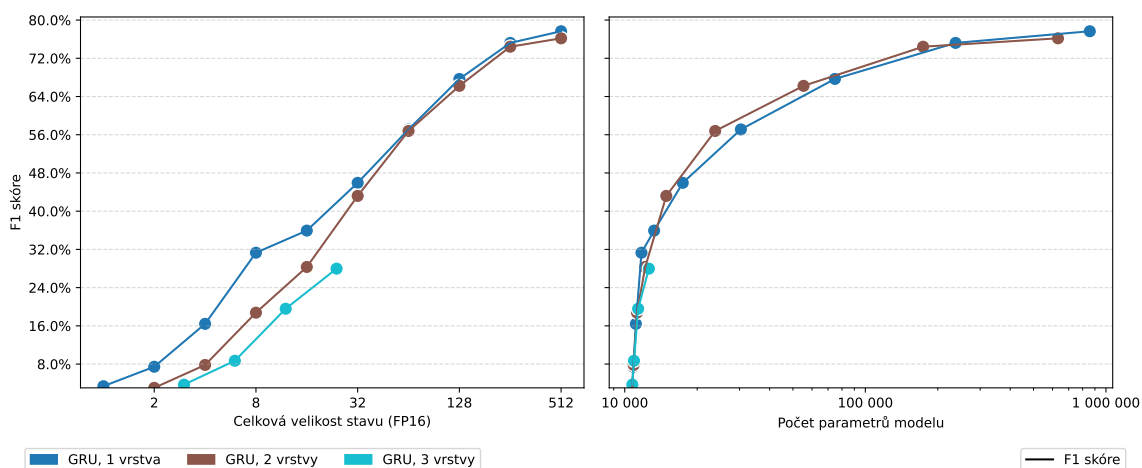
Jedním ze způsobů, jak zvýšit výpočetní kapacitu rekurentní buňky, a tím i potenciálně přesnost klasifikace, je zvětšit velikost skrytého stavu. Protože je však stav uložen pro každý aktivní tok separátně, je tento způsob škálování nevýhodný z hlediska paměťové složitosti. Navíc počet parametrů rekurentní buňky roste kvadraticky s velikostí stavu.

Běžným způsobem, jak zvýšit výpočetní kapacitu neuronových sítí bez velkého nárůstu počtu parametrů, je navýšení počtu vrstev. V kontextu RNN se nejčastěji používá tzv. *stacked* architektura (viz obrázek 4.12). Každá vrstva spravuje vlastní skrytý stav. Vstupem vyšších vrstev je skrytý stav vrstev předchozích. Počet parametrů již roste lineárně s počtem vrstev, nicméně opět dochází ke zvýšení paměťové složitosti kvůli nutnosti ukládat více stavů.



Obrázek 4.12: Srovnání standardního GRU modelu a vícevrstvého GRU modelu (tzv. *stacked* GRU). Rekurentní buňky s různými parametry jsou odlišeny indexy (viz Buňka [i]).

Srovnání sítí GRU s různým počtem vrstev a různými velikostmi stavu zobrazuje obrázek 4.13. Protože každá vrstva pracuje s vlastním skrytým stavem, reprezentuje levá horizontální osa celkovou velikost stavu (velikost stavu jedné vrstvy vynásobená počtem vrstev). Z měření vyplývá, že při srovnání s vícevrstevnými sítěmi GRU dosahuje jednovrstvá síť GRU s ekvivalentní velikostí stavu vyššího F1 skóre. Následuje popis několika netradičních architektur vícevrstevných rekurentních sítí, které umožňují zvýšit počet vrstev bez zvětšení celkového stavu.

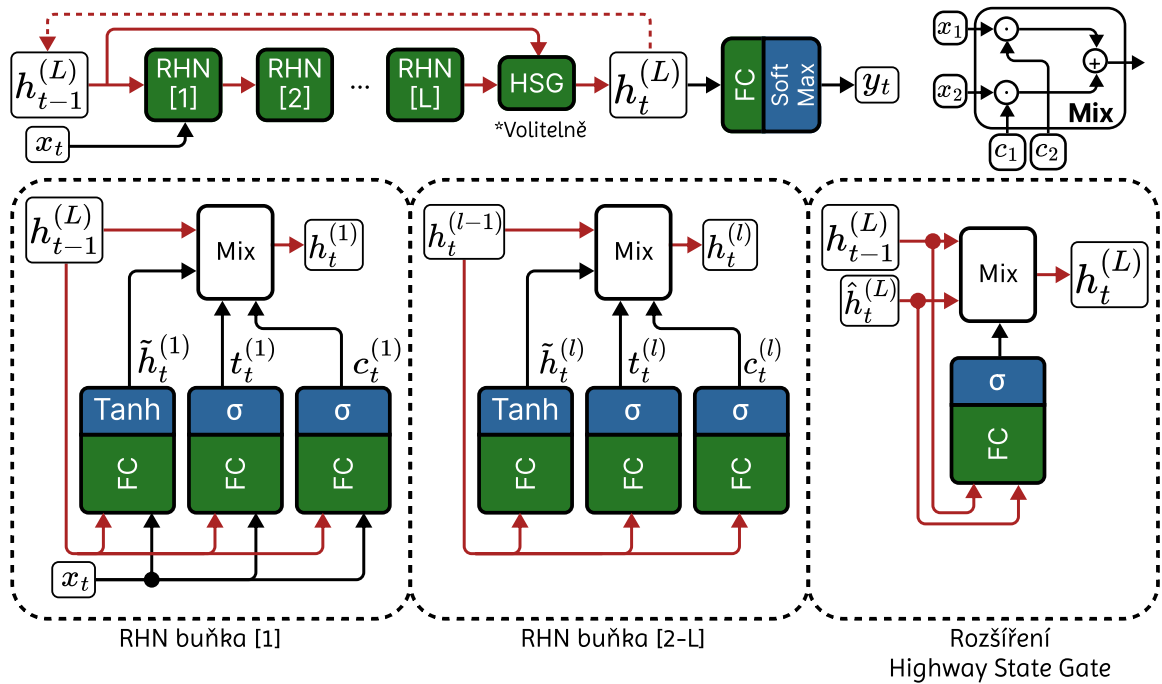


Obrázek 4.13: Srovnání vícevrstevných GRU sítí (tzv. *stacked* GRU) podle velikosti skrytého stavu a počtu parametrů. Formát grafu je stejný jako na obrázku 4.5.

Práce [28] představuje architekturu DT-RNN (Deep Transition RNN), která zvyšuje počet vrstev uvnitř rekurentní buňky, díky čemuž nedochází ke zvětšení stavu. Nový skrytý stav je vypočten pomocí několika sekvenčně zapojených lineárních a nelineárních vrstev. Problémem je, že gradienty musí během zpětné propagace projít přes několik nelineárních vrstev, což vede ke stejným problémům jako u základní rekurentní sítě (mizející nebo explodující gradienty).

Práce [15] představuje mechanismus ACT (Adaptive Computation Time), ve kterém se struktura samotné rekurentní buňky nemění, ale zvyšuje se počet aplikací této buňky uvnitř jednoho časového kroku (tzv. mikro časové kroky). Zatímco u sítě GRU byl skrytý stav upraven právě jednou pro každý vstupní vektor x_t , u ACT je možné stav upravit vícekrát. Práce navíc představuje tzv. halting mechanismus, který umožňuje měnit počet mikro kroků dynamicky podle vstupních dat. Problémem ACT je, že parametry rekurentní buňky zůstávají stejné napříč všemi mikro kroky, a proto nelze specializovat chování jednotlivých mikro kroků.

Problémy DT-RNN i ACT řeší rekurentní síť RHN (Recurrent Highway Network) představená v [43]. Stejně jako u ACT se rekurentní buňka aplikuje vícekrát uvnitř jednoho časového kroku, nyní však s různými parametry pro každý mikro krok (viz obrázek 4.14). Na rozdíl od DT-RNN jsou v RHN využity reziduální spoje, které řeší problém špatné propagace gradientů během trénování. Reziduální spoje tvoří hlavní datovou cestu napříč celou sítí (tzv. highway). Skrytý stav, který je přenášen přes highway, může být podobně jako u GRU upraven jen pomocí Mix bloku (viz sekce 4.5).

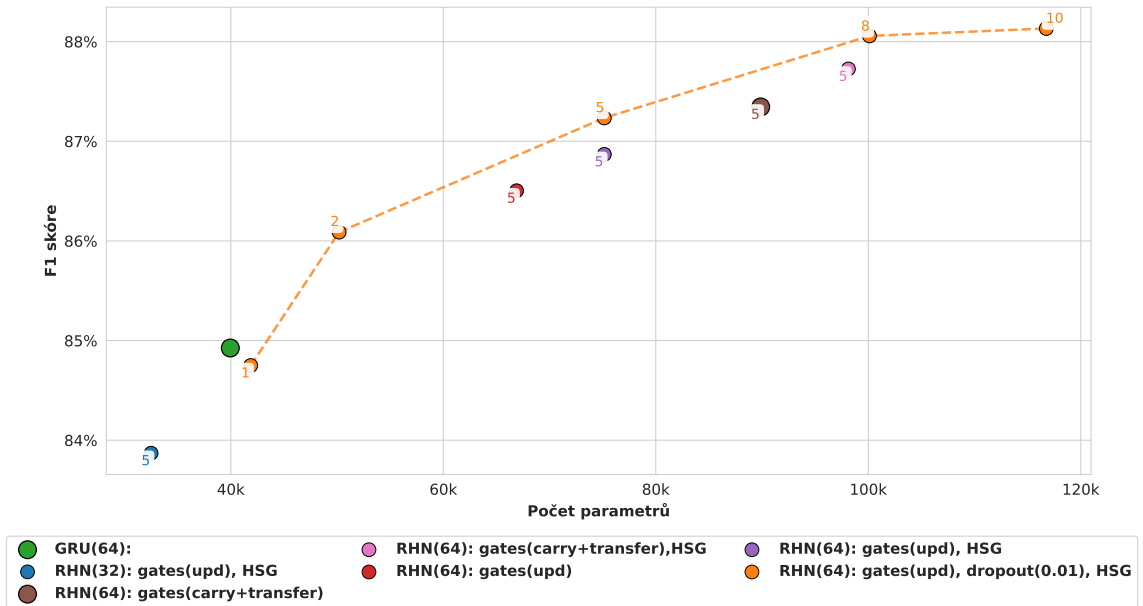


Obrázek 4.14: Schematické znázornění rekurentní buňky RHN (Recurrent Highway Network) s rozšířením HSG (Highway State Gating). Proměnná $l \in \{1, 2, \dots, L\}$ vyjadřuje index mikro kroku a proměnná t označuje aktuální časový krok.

Architektury sítí RHN a GRU se liší ve třech faktorech: (i) zavedení mikro kroků uvnitř jednoho časového kroku, (ii) vynechání reset brány r_t , která je použita v síti GRU

a (iii) mírně odlišný způsob aktualizace stavu (viz mix blok na obrázku 4.14). Zatímco u sítě GRU byla aktualizace stavu řízena jednou bránou u_t , v případě sítě RHN jsou v každém mikro kroku l použity dvě brány $t_t^{(l)}$ a $c_t^{(l)}$ (*transfer* a *carry*).

Od prvního představení sítě RHN vzniklo několik dalších publikací, které upravují a vylepšují původní architekturu. Některá z těchto vylepšení jsem implementoval a porovnal na úloze klasifikace šifrovaného provozu. Porovnání F1 skóre a počtu parametrů různých variant sítě RHN je zobrazeno na obrázku 4.15. Dvě konfigurace jsou zvýrazněny větší velikostí bodů: (i) GRU síť s velikostí stavu 64 (viz bod **GRU(64)**) a (ii) základní RHN síť s velikostí stavu 64 a hloubkou 5 mikro kroků (viz bod **gates(carry+transfer)**). Významy dalších bodů budou popsány níže. Číslice u jednotlivých bodů udávají počet mikro kroků, a tedy i počet RHN vrstev.



Obrázek 4.15: Srovnání F1 skóre různých konfigurací RHN sítě. Konfigurace využívající GRU síť a základní RHN síť jsou zvýrazněny větší velikostí bodů. Všechny modely využívají stejné transformace vstupních a výstupních příznaků: vstupní vrstva **Embed(i, ii, iii-B)**, 16x a výstupní vrstva **FC, ReLU, FC, softmax**. Velikost skrytého stavu je uvedena v legendě (buď 64, nebo 32 hodnot).

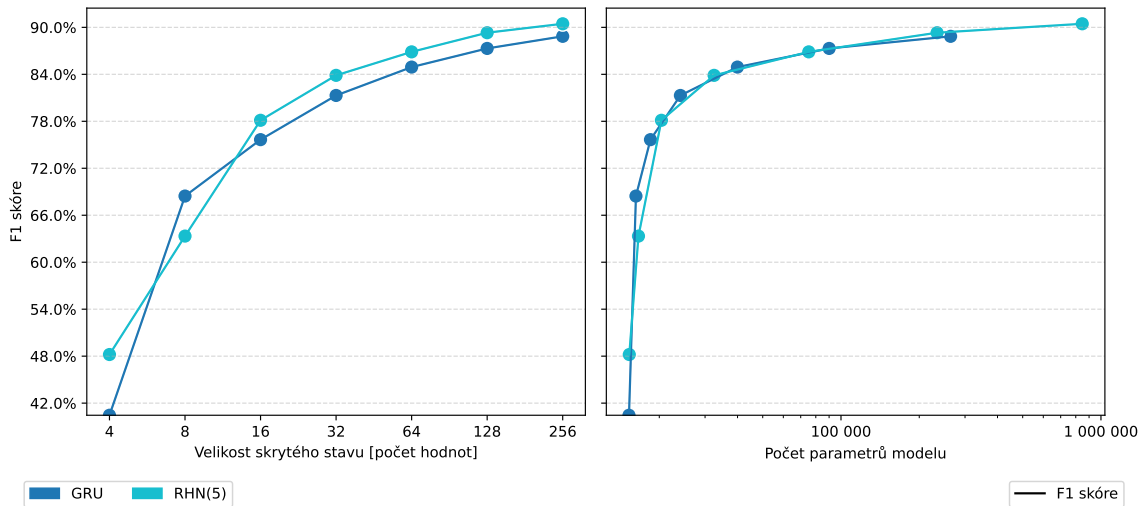
Síť RHN lze zjednodušit použitím jediné brány (update) namísto dvojice bran (*transfer* a *carry*), podobně jako u sítě GRU. Tato úprava sníží počet parametrů rekurentní buňky o třetinu a F1 skóre poklesne pouze o 0,84 %. Konfigurace využívající tuto úpravu jsou označeny jako **gates(update)**.

Často využívaným modulem v běžných hlubokých neuronových sítích je tzv. dropout, který regularizuje model tím, že během trénování náhodně přeruší některá spojení mezi vrstvami. Tato vrstva však není příliš často využívána uvnitř rekurentních buněk. Práce [9] však představuje způsob aplikace dropout vrstvy na skrytý stav rekurentní sítě, který vychází z teoretické analýzy rekurentních sítí. Hlavním výsledkem této práce je zjištění, že dropout maska musí být totožná pro všechny časové kroky. Při použití této techniky došlo ke zvýšení F1 skóre o 0,73 % bez jakékoliv změny počtu parametrů (viz konfigurace označené jako **dropout(rate)**).

Článek [32] představuje rozšiřující modul nazvaný HSG (Highway State Gating), který usnadňuje propagaci gradientů napříč časovými kroky. HSG modul je umístěn na konci highway cesty v daném časovém kroku a slouží ke sloučení skrytého stavu z předchozího časového kroku $h_{t-1}^{(L)}$ a nově vypočteného stavu z posledního mikro kroku $\hat{h}_t^{(L)}$ (viz obrázek 4.14). Tímto rozšířením byl model zlepšen pouze o 0,38 % F1 skóre, ale počet parametrů byl zvýšen o 8 256 (viz konfigurace označené jako HSG).

Pro konfiguraci `gates(upd)`, `dropout(0.1)`, HSG byl také změřen vliv hloubky rekurentní buňky (počet mikro kroků). V případě hloubky 1 se RHN model podobá GRU síti bez reset brány, přičemž F1 skóre RHN sítě bylo nižší o 0,17 %. Od hloubky 8, při které bylo dosaženo F1 skóre 88,05 %, již nedošlo k významnému zlepšení přesnosti klasifikace. Hloubka 5 představuje dobrý kompromis mezi přesností klasifikace a počtem parametrů.

Obrázek 4.16 srovnává síť RHN a GRU při různých velikostech stavu. Konkrétně byla použita RHN síť s hloubkou 5, rozšířením HSG a pouze s bránou update. Z pohledu velikosti stavu je RHN vždy výhodnější (kromě stavu o velikosti 8 hodnot). Naopak z pohledu počtu parametrů je výhodnější GRU síť, ale pouze pro velikost stavu menší než 128 hodnot. Pro stav o velikosti 128 hodnot dosahuje RHN model F1 skóre 89,31 % a GRU 87,31 %. Navíc RHN síť má pouze 88,7 % parametrů GRU sítě. RHN tedy škáluje lépe s velikostí skrytého stavu než GRU síť.



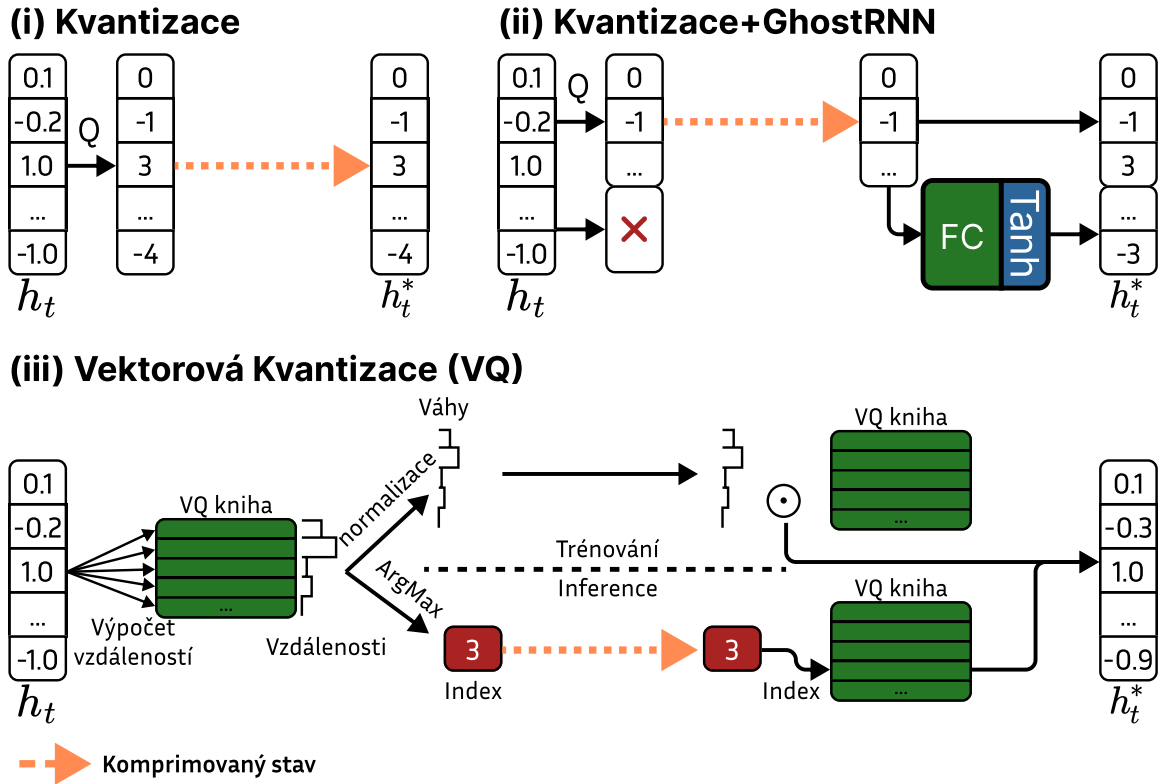
Obrázek 4.16: Srovnání přesnosti klasifikace sítí GRU a RHN s různou velikostí skrytého stavu. Všechny modely využívají stejné transformace vstupních a výstupních příznaků: vstupní vrstva `Embed(i, ii, iii-B)`, 16x, výstupní vrstva `FC, ReLU, FC, softmax`.

4.8 Komprese stavu

Protože je skrytý stav uložen pro každý aktivní tok zvlášť, má jeho velikost zásadní vliv na celkové paměťové nároky výsledného řešení. Proto se tato sekce zaměřuje na metody, které umožňují redukovat velikost skrytého stavu rekurentní sítě bez výrazné ztráty přesnosti klasifikace.

Jednoduchým způsobem redukce velikosti stavového vektoru je uniformní kvantizace jeho hodnot (viz obrázek 4.17(i)). Kvantizací celého modelu se podrobněji zabývá následující kapitola 5, nicméně již v této sekci je provedeno několik prvotních experimentů. Jedná

se však pouze o kvantizaci skrytého stavu. Konkrétně byla použita síť RHN, nástroj Brevitas a metoda QAT (viz sekce 2.4). Operátory simulované kvantizace byly umístěny na rekurentní zpětnou vazbu.

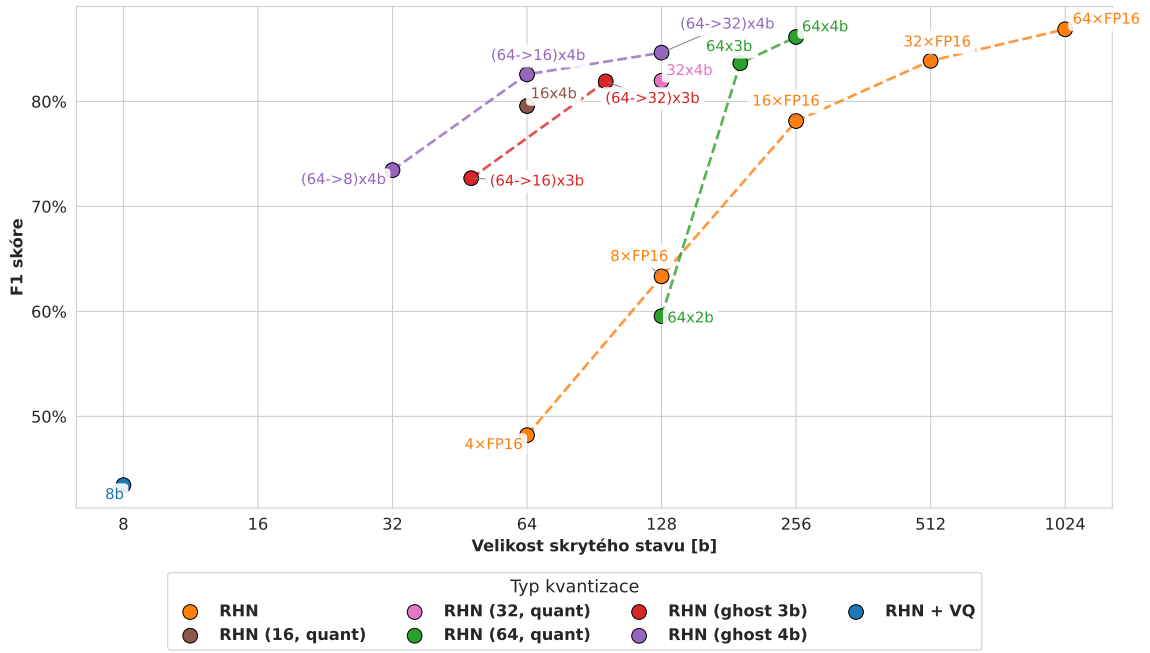


Obrázek 4.17: Srovnání tří metod komprese skrytého stavu h_t rekurentní sítě: (i) uniformní kvantizace všech položek stavu na menší počet bitů, (ii) spojení uniformní kvantizace s Ghost RNN architekturou rekurentní sítě a (iii) vektorová kvantizace. Metoda (ii) spočívá v uložení pouze primární části stavu a v odvození zbylé části stavu pomocí plně propojené vrstvy. Metoda (iii) spočívá v uložení pouze indexu nejbližšího vektoru z kódové knihy namísto celého stavového vektoru. Aby bylo možné neuronovou síť s vektorovou kvantizací trénovat, je nutné během trénování použít pouze aproximaci vektorové kvantizace.

Přesnost sítě RHN s kvantizovaným stavem zobrazuje obrázek 4.18 (viz body označené 64x4b, 64x3b, 64x2b, 32x4b, 16x3b). Pro velikost stavu 64 hodnot způsobila kvantizace stavu na 4 bity pokles F1 skóre pouze o 0,73 %. Při kvantizaci na 3 bity byl pokles F1 skóre již 3,22 % a při kvantizaci na 2 bity došlo k významnému poklesu až o 27,32 %. Kvantizace skrytého stavu na 4 bity tedy způsobí pouze minimální pokles přesnosti klasifikace, ale zároveň redukuje paměťovou náročnost stavu z 1024 bitů (pro datový typ FP16) na 256 bitů.

Další významné snížení velikosti skrytého stavu umožnila práce [42] představující *Ghost RNN* architekturu. Autoři zjistili, že skryté stavy rekurentních sítí obsahují redundantní hodnoty, které je možné odvodit z ostatních hodnot stavu. Skrytý stav je rozdělen na tzv. *primární stav* (angl. *intrinsic state*) a *odvozený stav* (angl. *ghost state*). Do FPGA stačí uložit pouze primární stav (případně jeho kvantizovanou verzi). Odvozený stav je dopočítán z primárního stavu pomocí jedné učitelné lineární vrstvy (viz obrázek 4.17(ii)).

Přesnost RHN s Ghost RNN rozšířením a kvantizovaným primárním stavem zobrazuje obrázek 4.18 (viz body označené (64→16)x4b, (64→8)x4b atd.). Popisky bodů mají ná-



Obrázek 4.18: Srovnání přesnosti klasifikace sítí RHN s různými způsoby komprese skrytého stavu. Všechny modely mají stejnou architekturu: vstupní vrstva $\text{Embed}(i, ii, iii-B)$, $16x$, rekurentní vrstva RHN s hloubkou 5 vrstev a výstupní vrstva FC, ReLU, FC, softmax. Křivka označená jako RHN nevyužívá žádnou kompresi stavu (každá položka stavového vektoru je reprezentována datovým typem FP16). Byly použity 3 druhy komprese stavu: (i) uniformní kvantizace (řešení označená *quant*), (ii) Ghost RNN rozšíření (řešení označená *ghost*), (iii) vektorová kvantizace (řešení označená *vq*).

sledující formát: $(P \rightarrow G) \times B$, kde P je velikost primárního stavu, G je velikost odvozeného stavu a B je počet bitů použitých pro kvantizaci primárního stavu. Například konfigurace $(64 \rightarrow 16) \times 4b$ snížila velikost skrytého stavu $16 \times$ (oproti $64x\text{FP16}$) a F1 skóre pokleslo o 4,29 %. Konfigurace $(64 \rightarrow 8) \times 4b$ snížila velikost skrytého stavu $32 \times$ a F1 skóre pokleslo o 13,41 %.

Kompresi stavu pomocí Ghost RNN rozšíření byla účinná, protože oproti uniformní kvantizaci byly uvažovány i závislosti mezi položkami skrytého stavu. Další metoda, která zohledňuje korelace uvnitř kódovaného vektoru, je vektorová kvantizace (zkráceně VQ). V případě VQ je každý vektor aproximován jedním nejbližším vektorem z tzv. *kódové knihy* (angl. *codebook*). Do paměti FPGA pak stačí uložit pouze index do kódové knihy, namísto původního vektoru.

Největším problémem VQ je vytvoření vhodné kódové knihy. V kontextu neuronových sítí existují způsoby, jak tuto kódovou knihu učit během trénování modelu. Stejně jako uniformní kvantizace je i VQ nediferencovatelná operace, a proto je během trénování nutné pracovat pouze s její aproximací. Modul vektorové kvantizace má tedy odlišné chování během trénování a během inference (viz obrázek 4.17(iii)).

V rámci experimentu byla zvolena tzv. *Smooth VQ* aproximace [26]. Zatímco během inference je vektor aproximován jedním nejbližším vektorem z kódové knihy, během trénování je vektor kódován jako vážený průměr všech vektorů z kódové knihy. Váhy w_i z tohoto váženého průměru by měly odpovídat podobnosti vstupního vektoru s vektory z kódové knihy.

Pro výpočet podobnosti s_i byla použita záporná Eukleidovská vzdálenost (viz rovnice 4.4). Čím blíže budou váhy w_i k tzv. *one-hot* kódování, tím bude tato aproximace přesnější. Například pokud by pouze jedna váha byla rovna 1 a ostatní váhy byly rovny 0, pak by tento vážený průměr přesně odpovídal zvolení nejbližšího vektoru z kódové knihy. Váhy w_i jsou následně získány normalizací podobností s_i pomocí funkce *Gumbel-Softmax* [18] (viz rovnice 4.5).

$$s_i = -\sqrt{\sum_k (v_k - c_{i,k})^2} \quad (4.4)$$

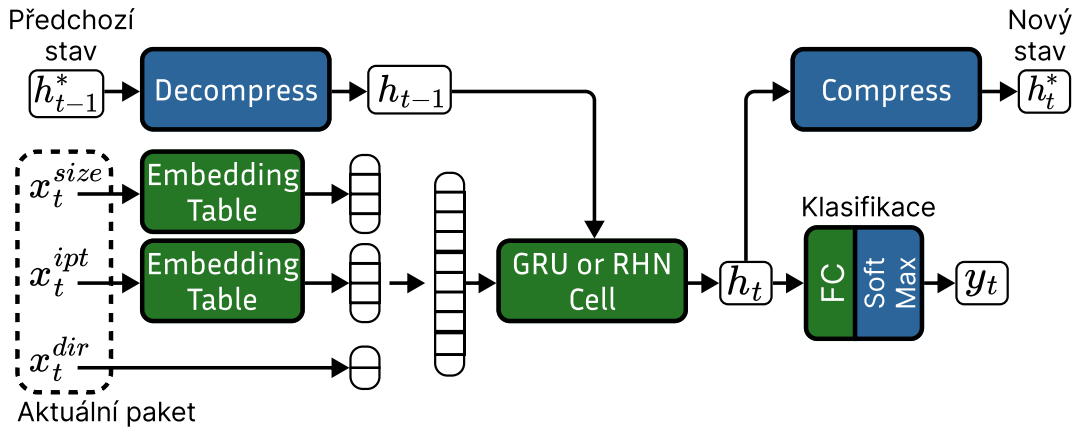
$$w_i = \frac{\exp((s_i + g_i)/\tau)}{\sum_j \exp((s_j + g_j)/\tau)} \quad (4.5)$$

$$g_i = -\log(-\log(u_i)), \quad u_i \sim \text{Uniform}(0, 1)$$

Velikost kódové knihy byla nastavena na 256 vektorů, díky čemuž je velikost indexu 8 bitů. F1 skóre sítě RHN s vektorovou kvantizací stavu zobrazuje obrázek 4.18 (viz bod označený 8b). Stav byl redukován na pouhých 8 bitů na každý tok, ale F1 skóre kleslo na 43,45 %. Nicméně vektorová kvantizace se prokázala jako užitečná při experimentech s menšími modely a při klasifikaci do menšího počtu tříd (viz sekce 4.9).

4.9 Ohodnocení navržených modelů

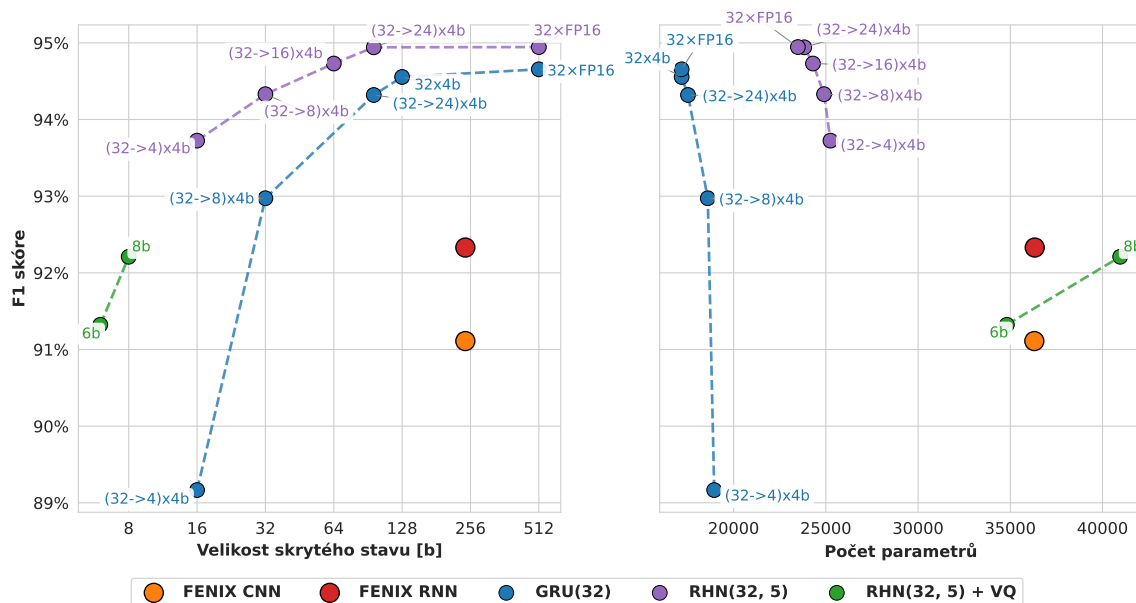
Na základě provedených experimentů byla zvolena struktura modelu zobrazená na obrázku 4.19. Vstupní příznaky jsou transformovány pomocí embedding tabulek s možnými konfiguracemi popsány v sekci 4.6. Jádrem modelu je rekurentní vrstva RHN nebo GRU (viz sekce 4.7). Při přijetí nového paketu se z paměti načte skrytý stav rekurentní buňky, aktualizuje se na základě vstupních příznaků a uloží se zpět do paměti. Případně je možné stav před uložením komprimovat pomocí metod popsány v sekci 4.8. Nekomprimovaná verze skrytého stavu je následně transformována výstupními vrstvami (viz sekce 4.6) do vektoru pravděpodobností jednotlivých tříd.



Obrázek 4.19: Architektura modelu navrženého pro klasifikaci síťových toků v FPGA.

Srovnání na datové sadě VPN-ISCX-2016

Online klasifikátory šifrovaného provozu prozkoumané v sekci 3.4 byly porovnány na datové sadě VPN-ISCX-2016. Proto byl i model navržený v této práci natrénován a ohodnocen na této datové sadě. Konkrétně byl zvolen embedding blok v konfiguraci Embed(i, ii, iii-B), 4x a pouze jedna plně propojená výstupní vrstva (viz sekce 4.6). Byly ověřeny rekurentní buňky GRU i RHN. Skrytý stav byl komprimován pomocí metody Ghost RNN s uniformní kvantizací na 4 bity, nebo pomocí vektorové kvantizace. Výsledky jsou zobrazeny na obrázku 4.20.



Obrázek 4.20: Srovnání F1 skóre navržených modelů a systému FENIX na datové sadě VPN-ISCX-2016. Levá horizontální osa zobrazuje velikost skrytého stavu v bitech a pravá horizontální osa zobrazuje počet parametrů modelu.

Jako referenční řešení byl zvolen systém FENIX, který z prozkoumaných online klasifikátorů dosáhl nejvyššího F1 skóre (viz sekce 3.4). Autoři konkrétně představili rekurentní a konvoluční model (viz body FENIX RNN a FENIX CNN na obrázku 4.20). Autoři systému FENIX neposkytují přesnost klasifikace nekvantizovaných modelů. Aby tedy porovnání bylo spravedlivé, byly modely systému FENIX znovu natrénovány bez kvantizace pomocí kódu zveřejněného v původním článku⁶. Rekurentní model FENIX dosáhl F1 skóre 92,33 % a konvoluční model skóre 91,11 %. Stavem modelů systému FENIX jsou příznaky (velikost paketu a časový rozestup) pro posledních 9 paketů. Velikost skrytého stavu systému FENIX pro jeden síťový tok je tedy $9 \cdot (\log_2(1500) + \log_2(65001)) = 243$ bitů.

Modely s Ghost RNN kompresí a rekurentní vrstvou RHN převyšují oba modely systému FENIX z pohledu F1 skóre, počtu parametrů i velikosti skrytého stavu. Například konfigurace (32→8)x4b dosáhla o 3,22 % vyššího F1 skóre než nejlepší FENIX model, a to s pouhými 68,5 % jeho parametrů a 7,6× menším skrytým stavem. Vektorová kvantizace umožňuje dosáhnout mnohem menší velikosti skrytého stavu, avšak za cenu vyššího počtu parametrů a nižšího F1 skóre. Například skrytý stav konfigurace 8b má 30× menší velikost než modely FENIX, zatímco F1 skóre je srovnatelné a počet parametrů je o 12,75 % vyšší.

⁶<https://github.com/IntelliSwitch/FENIX>

Srovnání na datové sadě CESNET-QUIC22

V předchozí části byla řešení porovnána na datové sadě VPN-ISCX-2016. Ta však obsahuje pouze malé množství síťových toků (10 500) oproti datové sadě CESNET-QUIC22 (153 milionů toků). Navíc byla vytvořena v laboratorním prostředí. Proto bylo provedeno i další srovnání na datové sadě CESNET-QUIC22.

Kromě systému FENIX byl jako referenční model zvolen i CESNET-MM-V2 [23], který v sekci 3.3 dosáhl nejvyššího F1 skóre na této datové sadě. Jedná se však o velkou konvoluční síť učenou pro offline klasifikaci, a proto jsou její cíle mírně odlišné od modelu navrženého v této práci. Navíc je vhodné poznamenat, že model CESNET-MM-V2 používá kromě příznaků SPLT i statistiky toku, které jsou agregovány ze všech paketů toku (ne pouze z prvních 30 paketů jako v případě SPLT). Tento referenční model tedy může mít k dispozici více informací než navržený model.

Model CESNET-MM-V2 byl v této práci natrénován a ohodnocen na stejné trénovací a testovací množině dat jako navržené modely (viz sekce 4.4). Pro trénování byly použity hyperparametry z práce [23]. Model dosáhl přesnosti 96,0 % a F1 skóre 89,8 %.

Vyhodnocení na datové sadě CESNET-QUIC22 bylo rozděleno do dvou scénářů: (i) online klasifikace, v níž jsou uvažovány pouze malé rekurentní modely implementovatelné v FPGA a (ii) offline klasifikace, v níž je ověřena škálovatelnost rekurentních modelů. Klasifikace síťových toků do všech 102 tříd vyžaduje výstupní vrstvy s velkým množstvím parametrů. Právě tyto vrstvy by limitovaly malé modely ověřované ve scénáři online klasifikace. Z tohoto důvodu je počet klasifikovaných webových aplikací pro tento scénář snížen na 16⁷.

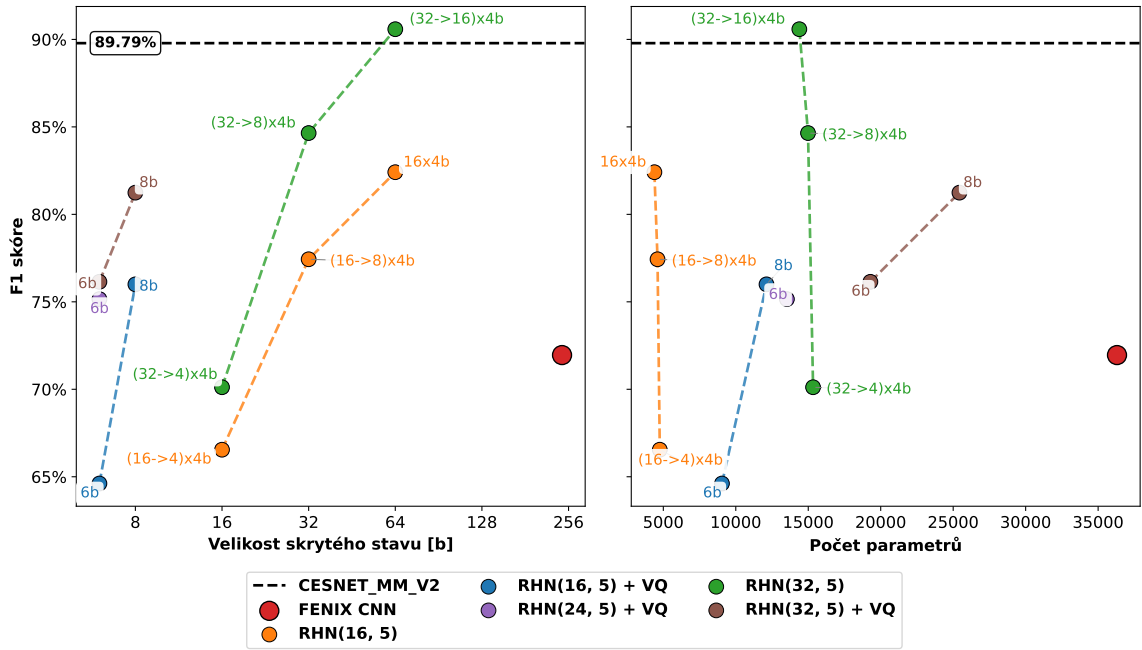
Srovnání modelů v případě scénáře online klasifikace zobrazuje obrázek 4.21. Pro tuto kategorii byla použita vstupní vrstva označená jako `Embed(i, ii, iii-A)` (viz sekce 4.6), která embedding vektory sestavuje dynamicky během inference, a nemá tedy žádné parametry. Skrytý stav rekurentní vrstvy je transformován do výstupního vektoru pomocí vrstev s konfigurací `FC, ReLU, FC, softmax`. Je použita rekurentní vrstva RHN s velikostí stavu 16, 24 a 32 hodnot (viz legenda). Pro kompresi skrytého stavu byly použity dvě metody ze sekce 4.8: (i) Ghost RNN s uniformní kvantizací na 4 bity (např. konfigurace (32->8)x4b) a (ii) vektorová kvantizace (např. konfigurace 8b).

Z pohledu velikosti skrytého stavu je nejvýhodnější vektorová kvantizace. Například při použití kódové knihy s 256 vektory o velikosti 32 hodnot je velikost skrytého stavu pouze 8 bitů (index do kódové knihy). Nevýhodou je vyšší počet parametrů (konkrétně 25 431) a nižší F1 skóre (konkrétně 81,24 %).

Nejvyššího F1 skóre (konkrétně 90,58 %) dosáhla konfigurace založená na Ghost RNN ((32->16)x4b). Velikost skrytého stavu této konfigurace je 64 bitů a počet parametrů je 14 385. Z pohledu počtu parametrů je výhodná například konfigurace 16x4b, která používá pouze uniformní kvantizaci skrytého stavu (celkově 64 bitů). Konfigurace dosáhla F1 skóre 82,41 %, a to pouze se 4 385 parametry.

Pro ověření škálovatelnosti navržených modelů byla použita rekurentní síť RHN s velikostí stavu 256 hodnot a hloubkou 5 vrstev. Vstupní embedding blok má konfiguraci `Embed(i, ii, iii-B)`, 1x a výstupní vrstva má konfiguraci `FC, ReLU, FC, softmax`. Pro kompresi skrytého stavu je použita pouze uniformní kvantizace na 4 bity. Srovnání

⁷Konkrétně byly zvoleny následující webové aplikace: google-www, youtube, instagram, spotify, snapchat, facebook-web, discord, facebook-messenger, google-pay, alza-www, tiktok, whatsapp, rohlík, gmail, shazam, uber



Obrázek 4.21: Srovnání F1 skóre modelů ohodnocených na datové sadě CESNET-QUIC22. Levá horizontální osa zobrazuje velikost skrytého stavu v bitech a pravá horizontální osa zobrazuje počet parametrů modelu.

této konfigurace rekurentní sítě a referenční konvoluční sítě CESNET-MM-V2 zobrazuje tabulka 4.1.

Navržený model překonal referenční konvoluční síť z pohledu F1 skóre (o 0,97 %), počtu parametrů (obsahuje pouze 38,98 % parametrů konvoluční sítě) i z pohledu velikosti skrytého stavu (o 160 bitů). Z těchto experimentů vyplývá, že navržené rekurentní modely jsou nejen schopny překonat nejlepší řešení online klasifikace, ale mohou konkurovat i velkým konvolučním sítím určeným pro offline klasifikaci.

Tabulka 4.1: Srovnání F1 skóre, počtu parametrů a velikosti skrytého stavu navrženého modelu RHN s konvoluční CESNET-MM-V2 na datové sadě CESNET-QUIC22. Velikost stavu modelu CESNET-MM-V2 je reflektována velikostí vstupních příznaků. Konkrétně příznaky SPLT obsahují pro prvních 30 paketů toku: velikost paketu (1 až 1500 bajtů) časový rozestup (omezený na rozsah 0 ms až 65 000 ms) a směr paketu. Velikost SPLT příznaků je tedy $30 \cdot (\log_2(1500) + \log_2(65001) + 1) \approx 840$ bitů. Pro statistiky toku by bylo možné použít $43 \cdot 8 = 344$ bitů. Celková velikost stavu modelu CESNET-MM-V2 je tedy 1184 bitů.

Konfigurace	F1 skóre	Počet parametrů	Velikost stavu
RHN(256, 5)	90,76 %	881 747	1 024 b
CESNET-MM-V2	89,79 %	2 261 653	1 184 b

Kapitola 5

Implementace FPGA akcelérátoru

V této kapitole je implementován FPGA akcelérátor pro rekurentní modely navržené v kapitole 4. Kvantizace navržených modelů je popsána v sekci 5.1. Následně jsou v sekci 5.2 prozkoumány schopnosti existujících nástrojů pro FPGA akceleraci neuronových sítí. Jelikož žádný z prozkoumaných nástrojů neposkytuje požadované vlastnosti, bylo navrženo vlastní řešení, které je popsáno v sekci 5.3.

5.1 Kvantizace navrženého modelu

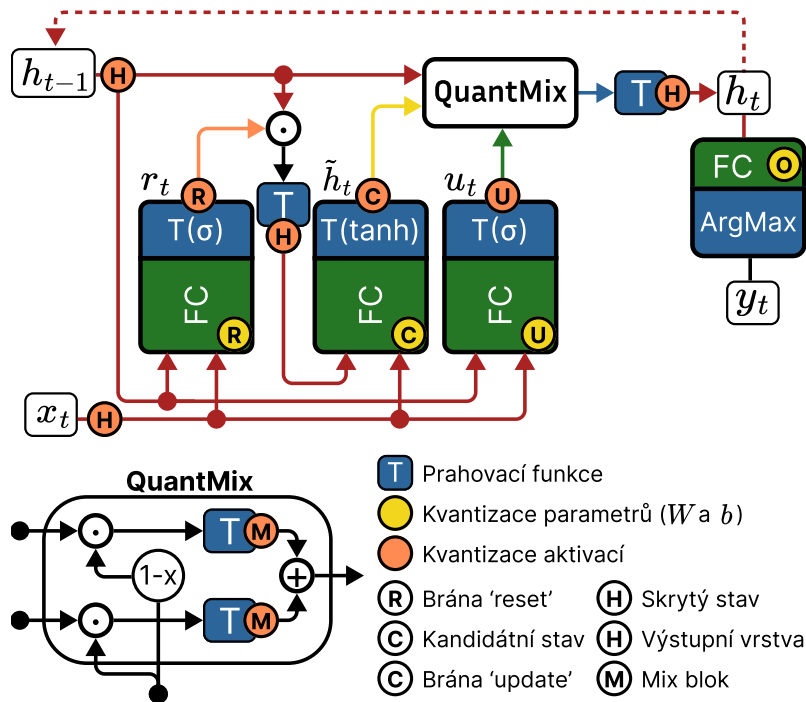
Z datových typů prozkoumaných v sekci 2.1 byl zvolen symetrický celočíselný datový typ. Oproti FP typům není nutné spravovat exponent a kvantizační rozsahy mohou být zvoleny přesněji než v případě FX typů. Navíc pro zjednodušení implementace byla zvolena symetrická varianta celočíselné kvantizace. Pro kvantizaci modelu je tedy nutné vhodně zvolit škálovací faktory S , zatímco posuny Z budou vždy nulové.

Na základě sekce 2.4 byla zvolena metoda QAT, při které jsou kvantizační parametry optimalizovány již během trénování modelu. Kvantizace je v průběhu trénování simulována pomocí operátorů simulované kvantizace. Aby bylo možné gradienty propagovat i přes tyto operátory, byla použita STE aproximace (viz sekce 2.4). Konkrétně byl zvolen nástroj *Brevitas*, který nabízí plnou kontrolu nad celým průběhem kvantizace.

Během kvantizace lineárních vrstev byly zohledněny požadavky popsané v sekci 2.3. Konkrétně byla zvolena následující omezení kvantizace lineárních vrstev: (i) celý vstupní vektor x sdílí stejný škálovací faktor S_x , (ii) celá matice vah W sdílí stejný škálovací faktor S_W a (iii) škálovací faktor prahu b je roven $S_W \cdot S_x$.

Nelineární aktivační funkce jsou aproximovány pomocí prahovacích funkcí. Aby bylo možné vypočítat prahy prahovacích funkcí podle výrazu 2.16, je nutné určit kvantizační parametry vstupu a výstupu nelineární vrstvy. Vstupem je akumulátor lineární vrstvy, který má stejný škálovací faktor jako práh ($S_a = S_b = S_W \cdot S_x$). Kvantizace výstupu nelineární vrstvy je realizována přidáním dalších operátorů simulované kvantizace za příslušné nelineární vrstvy (viz operátory R , C , U na obrázku 5.1).

Při použití rekurentních sítí vznikají dvě nová omezení kvantizace: (i) kvůli rekurentní zpětné vazbě musí mít vstupní i výstupní skrytý stav (h_{t-1} a h_t) stejné kvantizační parametry, (ii) protože vstupem lineárních vrstev je embedding vektor x_t a také skrytý stav h_{t-1} , musí x_t sdílet stejné kvantizační parametry jako skryté stavy h_t a h_{t-1} (viz operátor H na obrázku 5.1). Druhému omezení by se dalo vyhnout rozdělením lineární vrstvy na dvě



Obrázek 5.1: Umístění operátorů simulované kvantizace v navrženém GRU modelu. Žluté body označují kvantizaci parametrů lineárních vrstev (váhy W a prahy b) a oranžové body označují kvantizaci aktivací.

samostatné části (jedna pro transformaci x_t a druhá pro transformaci h_{t-1}). Následně by ale bylo nutné sečíst oba mezivýsledné akumulátory s rozdílnými kvantizačními parametry.

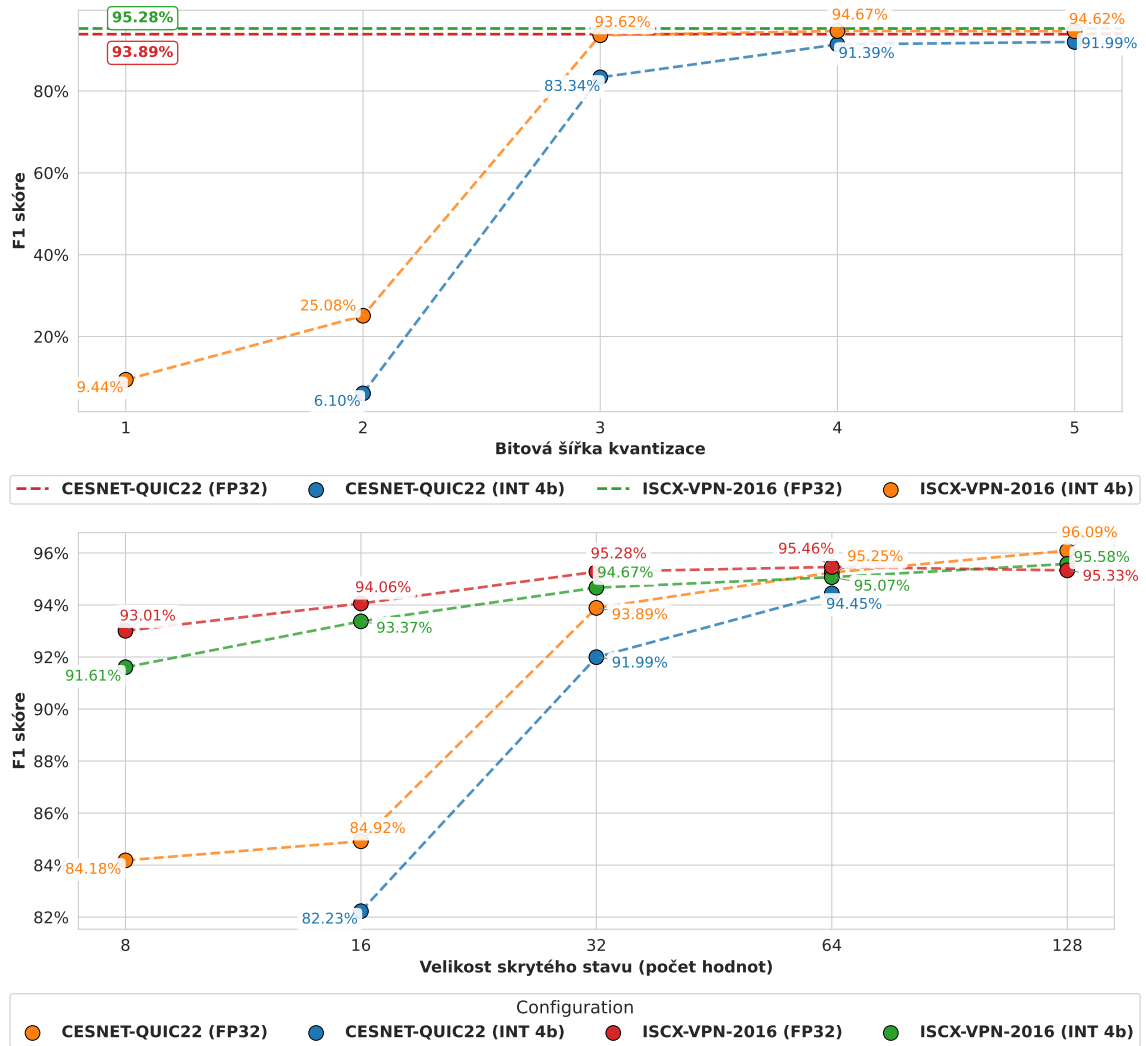
V GRU rekurentní buňce se navíc nacházejí dva specifické bloky: reset skrytého stavu pomocí součinu po složkách a tzv. mix blok. Součin $r_t \cdot h_{t-1}$ lze realizovat v celočíselné aritmetice, i pokud mají oba operandy různé škálovací faktory (viz rovnice 2.12). Nicméně výsledná bitová šířka bude $b_{r_t} + b_{h_{t-1}}$ a výsledný škálovací faktor bude roven $S_{r_t} \cdot S_{h_{t-1}}$. Aby tento výsledek mohl být použit následující lineární vrstvou, je zde nutné provést rekvantizaci. V FPGA akcelérátoru je tato rekvantizace realizována dodatečnou prahovací funkcí, ale během QAT je pouze simulována pomocí operátoru simulované kvantizace.

Podobný problém nastává i v případě mix bloku, který kombinuje předchozí skrytý stav h_{t-1} s novým kandidátním stavem $\tilde{h}_t$. Problém konkrétně nastává u součtu, jehož operandy musí mít stejné kvantizační parametry. Proto kvantizovaná varianta mix bloku (*QuantMix*) obsahuje dvě dodatečné prahovací funkce. Během QAT jsou tyto funkce simulovány operátory simulované kvantizace (viz M na obrázku 5.1). Za mix blokem následuje poslední prahovací funkce, díky které má výstupní skrytý stav h_t stejný kvantizační rozsah jako vstupní skrytý stav h_{t-1} .

Celkově je tedy v případě aktivací použito 5 unikátních operátorů simulované kvantizace (viz R , C , U , H a M na obrázku 5.1). Pro kvantizaci parametrů lineárních vrstev je použito 8 unikátních operátorů simulované kvantizace (každý ze symbolů R , C , U , O reprezentuje kvantizaci vah i prahů). FPGA akcelérátor bude obsahovat 7 prahovacích funkcí (dvě nahrazující sigmoidní aktivační funkce, jedna nahrazující $\tanh$ aktivační funkci a čtyři pro dodatečné rekvantizace).

Další problém představuje *softmax* funkce, jejíž přesné vyhodnocení by kvůli exponenciálám bylo komplikované (viz vzorec 2.10). Nicméně ve většině případů není nutné vypočítat přesné pravděpodobnosti všech tříd, ale stačí pouze určit index nejpravděpodobnější třídy. V takovém případě je možné použít jednodušší funkci $\text{ArgMax}(x) = \arg \max_i(x_i)$.

Vstupní příznaky byly transformovány pomocí embedding tabulek popsanych v sekci 4.6. Konkrétně byla zvolena konfigurace `Embed(i, ii, iii-B)`, `1x`, ve které je tabulka složena z plně učitelných embedding vektorů. Druhá varianta optimalizace embedding tabulky (`Embed(i, ii, iii-A)`), která umožní embedding vektory sestavovat dynamicky bez potřeby uložení všech embedding vektorů, zatím nebyla implementována. Kvantizační parametry embedding vektorů jsou určeny kvantizačním operátorem H .



Obrázek 5.2: Srovnání F1 skóre kvantizovaných modelů pro různé bitové šířky datových typů (horní graf) a pro různé velikosti skrytého stavu (dolní graf). Výsledky jsou uvedeny pro datové sady ISCX-VPN-2016 a CESNET-QUIC22. V případě sady CESNET-QUIC22 bylo použito pouze 16 tříd (stejně jako v sekci 4.9 pro tzv. online scénář). V grafech jsou uvedeny i F1 skóre původních modelů bez kvantizace (viz *FP32* v legendě). Byla použita rekurentní buňka GRU, vstupní embedding vrstva v konfiguraci `Embed(i, ii, iii-B)`, `1x` a pouze jedna plně propojená výstupní vrstva.

Kvantizace rekurentních modelů byla ověřena na stejných datových sadách, které byly použity v sekci 4.9 (konkrétně datové sady ISCX-VPN-2016 a CESNET-QUIC22). Z datové sady CESNET-QUIC22 bylo použito pouze 16 tříd. Obrázek 5.2 zobrazuje F1 skóre modelů, jejichž parametry i aktivace byly kvantizovány pomocí celočíselné kvantizace popsané výše. Horní graf zobrazuje F1 skóre pro různé bitové šířky kvantizace. Důležitým poznatkem je, že kvantizace do 4 bitových datových typů nezpůsobila výraznou ztrátu přesnosti (0,61 % v případě ISCX-VPN-2016 a 2,5 % v případě CESNET-QUIC22). Naopak při použití binární nebo ternární kvantizace došlo k poklesu F1 skóre pod 30 %.

Současným problémem QAT je dlouhá doba trénování těchto modelů. Zatímco trénování na datové sadě ISCX-VPN-2016 trvalo přibližně 2 hodiny, trénování na datové sadě CESNET-QUIC22 trvalo přibližně 17 hodin. Trénování modelů bez kvantizace bylo výrazně rychlejší (přibližně 10 minut pro datovou sadu ISCX-VPN-2016 a přibližně 2 hodiny pro datovou sadu CESNET-QUIC22). Důvodem může být velká obecnost nástroje Brevitas, která komplikuje kompilaci modelů pomocí funkce `torch.compile()`. Z důvodu dlouhé doby trénování také nebyla příliš prozkoumána kvantizace RHN modelů. QAT trénování RHN modelů bylo totiž 3× pomalejší než trénování GRU modelů. Jedním z cílů pokračování práce je proto navrhnout vlastní nástroj pro kvantizaci, který bude lépe přizpůsobený rekurentním architekturám.

5.2 FPGA implementace s využitím existujících nástrojů

V této sekci jsou prozkoumány schopnosti existujících nástrojů pro FPGA akceleraci neuro-nových sítí. Konkrétně byly zvoleny nástroje *FINN* a *HLS4ML* (viz sekce 2.5). Nejprve bylo prozkoumáno využití zdrojů lineární vrstvy, která představuje výpočetně nejnáročnější část navržených modelů. Byla zvolena plně propojená vrstva s 49 vstupy a 16 výstupy, která se v modelu *GRU(16)* vyskytuje 3×. Na základě sekce 5.1 byla zvolena 4 bitová kvantizace parametrů i aktivací.

Nejprve bylo nutné zvolit úroveň paralelismu lineární vrstvy (tzv. *reuse* faktor). Plně paralelní implementace (*reuse* faktor roven 1) má kromě nižší latence a vyšší propustnosti i další výhody. Protože má každý součin $x_i \cdot W_{ij}$ dedikovaný vlastní výpočetní element, je možné příslušný obvod optimalizovat přímo pro konkrétní hodnoty vah. Navíc je možné jednotlivé vrstvy propojit pomocí registrů namísto front, které by zvýšily latenci a množství FPGA zdrojů. Proto se tato sekce zaměřuje na plně paralelní implementaci lineárních vrstev.

Využití zdrojů FPGA pro implementaci zmíněné lineární vrstvy zobrazuje tabulka 5.1. Pro porovnání uvedených nástrojů byla zvolena frekvence 200 MHz a FPGA Zynq UltraScale+ (konkrétně *xczu7ev-ffvc1156-2-e*). Řešení jsou srovnány podle počtu využitých *LUT*, registrů (*FF*) a *DSP* bloků.

Nástroj FINN v základním nastavení využívá DSP bloky. Nicméně v FPGA je počet dostupných DSP bloků i jejich fyzické umístění na čipu značně omezené. Již pro jednu lineární vrstvu bylo spotřebováno 45,37 % dostupných DSP bloků. Proto bylo ověřeno i nastavení, které využívá LUT namísto DSP bloků (viz druhý řádek tabulky 5.1).

Nástroje FINN a HLS4ML spotřebovaly netriviální množství LUT pro implementaci jediné lineární vrstvy (konkrétně 20,35 % v případě nástroje FINN a 8,44 % v případě nástroje HLS4ML). Tento výsledek nasvědčuje tomu, že oba nástroje využívají obecné násobičky, i když je při plně paralelní implementaci možné každou násobičku optimalizovat na hodnoty jednotlivých vah. Pro ověření této hypotézy byly vytvořeny dvě vlastní HLS implementace lineární vrstvy (viz poslední dva řádky tabulky 5.1). V první verzi byly váhy W

Tabulka 5.1: Využití zdrojů plně propojené lineární vrstvy na FPGA Zynq UltraScale+ (xczu7ev-ffvc1156-2-e) a při frekvenci 200 MHz. Horní část tabulky zobrazuje výsledky pro nástroje FINN a HLS4ML, zatímco dolní část tabulky zobrazuje výsledky pro vlastní HLS implementaci. Výsledky byly zjištěny po syntéze.

Nástroj	LUT	FF	DSP
FINN (DSP)	34 164 (14,85 %)	22 136 (4,80 %)	784 (45,37 %)
FINN	46 816 (20,35 %)	21 118 (4,58 %)	0
HLS4ML	19 419 (8,44 %)	1 096 (0,24 %)	0
Vlastní HLS implementace			
Proměnné parametry W a b	20 042 (8,71 %)	704 (0,15 %)	0
Konstantní parametry W a b	4 448 (1,93 %)	1 157 (0,25 %)	0

a prahy b proměnné, zatímco ve druhé verzi byly konstantní. Ve druhém případě bylo spotřebováno 4,5× méně LUT než v prvním případě. Odtud vyplývá, že pro využití potenciálu plně paralelní implementace je nutné navrhnout vlastní řešení.

5.3 Vlastní FPGA implementace

Cílem této sekce je implementace FPGA akcelérátoru specializovaného pro rekurentní modely navržené v kapitole 4. Existující nástroje zmíněné v sekci 5.2 nedokázaly využít potenciálu plně paralelní implementace lineárních vrstev. Implementovaly totiž obecné násobičky i v případě, kdy měl každý součin $x_i \cdot W_{ij}$ dedikovaný vlastní výpočetní element. Ve stejné sekci byla také prezentována vlastní HLS implementace lineární vrstvy, která umožnila jednotlivé výpočetní elementy optimalizovat pro konkrétní hodnoty vah.

Je vhodné poznamenat, že nově navržené řešení umožňuje pouze plně paralelní implementaci. Proto nebude možné použít příliš velké neuronové sítě. V případě existujících nástrojů je možné takové modely implementovat, ale za cenu výrazně vyšší latence a nižší propustnosti.

Existující nástroje postupně transformují výpočetní graf modelu pomocí obecných transformací. To umožňuje implementovat širokou škálu modelů. Nicméně návrh FPGA akcelérátoru je možné ovládat pouze nepřímo přes tyto transformace. Pro vlastní řešení byl zvolen přístup, ve kterém je nutné akcelérátor sestavit ze sady předdefinovaných modulů (tzv. *kernelů*). Tím je docíleno větší kontroly nad návrhem akcelérátoru.

Každý kernel je definován Python třídou, která spravuje výpočet potřebných konstant (např. prahy pro prahovací funkce) a generuje odpovídající C++ HLS kód. Oproti existujícím nástrojům, které neuronovou síť skládají z předdefinovaných HLS tříd, je v tomto případě generován HLS kód přímo pro konkrétní nastavení parametrů. HLS syntéza může díky tomu provést více optimalizací než v případě existujících nástrojů.

Využití FPGA zdrojů a latence jednotlivých kernelů zobrazuje tabulka 5.2. Pro každý kernel je uveden i počet jeho výskytů v implementovaném GRU modelu (viz obrázek 5.1). Podle počtu výskytů jednotlivých kernelů byl vytvořen odhad množství zdrojů potřebných pro implementaci celého modelu. Uvedená procenta jsou vztažena k tomuto celkovému odhadu zdrojů, nikoli k dostupným zdrojům FPGA. V celé této sekci je již uvedeno konečné využití zdrojů na daném FPGA. Konkrétně byly použity následující příkazy pro Vivado: `synth_design`, `opt_design`, `place_design`, `route_design` a `phys_opt_design`.

Nejvyšší množství zdrojů je spotřebováno pro implementaci lineárních vrstev (75,1 % pro skrytý stav o velikosti 32 a 81,09 % pro skrytý stav o velikosti 64). Mezi další náročné kernely patří součin po složkách, prahovací funkce a vstupní embedding vrstva. Nicméně s rostoucí velikostí skrytého stavu se vliv těchto kernelů stává ve srovnání s lineárními vrstvami zanedbatelný.

Tabulka 5.2: Využití FPGA zdrojů a latence kernelů nutných pro implementaci GRU modelu. Konkrétně bylo použito FPGA Zynq UltraScale+ (xczu7ev-ffvc1156-2-e) a frekvence 200 MHz. U každého kernelu je uveden i počet jeho výskytů v implementovaném GRU modelu. Proto jsou také reportované zdroje uvedeny ve formátu 3 x 10 874. Sloupce LUT [%] a FF [%] udávají procentuální využití zdrojů daného druhu kernelu vzhledem k celkovému množství zdrojů GRU modelu.

Modul	Počet	LUT	LUT [%]	FF	FF [%]	[ns]
Skrytý stav o velikosti 32						
FC	3	3 x 10 874	75,11 %	3 x 2 767	55,56 %	5
Elementwise Součin	3	3 x 1 060	7,32 %	3 x 320	6,43 %	0
Embedding blok	1	1 x 2 749	6,33 %	1 x 234	1,57 %	5
Prahovací funkce	7	7 x 323	5,21 %	7 x 610	28,58 %	5
Výstupní FC	1	1 x 1 954	4,50 %	1 x 541	3,62 %	5
Elementwise Součet	1	1 x 395	0,91 %	1 x 320	2,14 %	0
ArgMax	1	1 x 169	0,39 %	1 x 155	1,04 %	5
Elementwise $(1 - x)$	1	1 x 101	0,23 %	1 x 160	1,07 %	0
Celkem	1	43 431	100,00 %	14 941	100,00 %	
Skrytý stav o velikosti 64						
FC	3	3 x 29 295	81,90 %	3 x 7 061	62,42 %	5
Elementwise Součin	3	3 x 2 377	6,65 %	3 x 640	5,66 %	0
Prahovací funkce	7	7 x 647	4,22 %	7 x 1 218	25,12 %	5
Výstupní FC	1	1 x 3 920	3,65 %	1 x 982	2,89 %	5
Embedding blok	1	1 x 2 683	2,50 %	1 x 211	0,62 %	5
Elementwise Součet	1	1 x 789	0,74 %	1 x 640	1,89 %	0
Elementwise $(1 - x)$	1	1 x 203	0,19 %	1 x 320	0,94 %	0
ArgMax	1	1 x 169	0,16 %	1 x 155	0,46 %	5
Celkem	1	107 309	100,00 %	33 937	100,00 %	

Ze zmíněných kernelů byl sestaven GRU model popsáný v sekci 5.1. Pro ověření jeho správnosti byly vytvořeny 3 úrovně verifikace: (i) verifikace na úrovni Pythonu, díky které je možné rychle najít a opravit hlavní chyby, (ii) verifikace na úrovni C++ simulace, která ověří samotnou implementaci kernelů a (iii) verifikace na HDL úrovni (tzv. *C/RTL co-simulation*). Všechny 3 úrovně ověřují mezivýsledky jednotlivých kernelů i finální výstup modelu vzhledem k referenčním výsledkům z kvantizovaného modelu v sekci 5.1. Tím je ověřeno, že přesnost FPGA implementace GRU modelu je shodná s přesností modelu kvantizovaného pomocí QAT (viz reportované hodnoty v sekci 5.1).

Celý model je implementovaný zřetězenou linkou a kvůli rekurentní zpětné vazbě mohou vznikat datové konflikty. Konkrétně konflikt vznikne v případě, kdy se v lince zpracovávají dva pakety (vzorky x_i a x_j) ze stejného síťového toku. Během verifikací je tento konflikt vyřešen pozastavením zřetězené linky do dokončení zpracování prvního vzorku.

Srovnání navrženého FPGA akcelérátoru a existujících metod online klasifikace (viz sekce 3.4) je zobrazeno v tabulce 5.3. Vlastní řešení byla srovnána se systémy N3IC, Kaleidoscope a FENIX z pohledu F1 skóre, využití FPGA zdrojů, latence, počtu inferencí za sekundu a velikosti stavu. U systému Kaleidoscope je uvedena přesnost klasifikace namísto F1 skóre, protože autoři neuvádějí jiné metriky. Systém BoS není uveden, neboť k akceleraci využívá síťový přepínač a GPU.

Pro všechna vlastní řešení je použita 4 bitová kvantizace parametrů i aktivací. Systém N3IC používá binární parametry i aktivace a ostatní systémy Kaleidoscope a FENIX používají 8 bitovou kvantizaci parametrů i aktivací.

Aby bylo možné přímo porovnat spotřebu hardwarových zdrojů, byla vlastní řešení implementována na stejném FPGA a při stejné frekvenci, kterou použili autoři referenčních prací při publikování svých výsledků. Systém N3IC byl implementován na FPGA Virtex-7 (*xc7vx690t-ffg1761-2*), systém Kaleidoscope na FPGA Virtex UltraScale+ (*xcvu13p-flga2577-2-e*) a systém FENIX na FPGA Zynq UltraScale+ (*xczu19eg-ffvc1760-2-i*).

Velikost stavu (sloupec *[b]*) reprezentuje množství bitů, které je nutné uložit pro každý aktivní síťový tok. FPGA zdroje nutné pro uložení tohoto stavu nejsou započítány do zdrojů uvedených v tabulce. V případě vlastního řešení je nutné uložit skrytý stav rekurentní sítě (jedná se o vektor 4 bitových hodnot). Stavem systému N3IC jsou statistiky toku reprezentované 256 bitovým vektorem. Systém Kaleidoscope nevyžaduje žádný stav, protože zpracovává každý paket nezávisle. Jeho vstupem je přímo prvních 64 bajtů daného paketu. Stavem systému FENIX jsou velikosti a časové rozestupy z posledních 9 paketů daného toku (konkrétně 243 bitů).

Latence (sloupec *[ns]*) reprezentuje dobu potřebnou pro zpracování jednoho paketu a získání příslušné klasifikace toku. Autoři systému FENIX uvádějí latenci klasifikátoru v FPGA na 2300 ns. Protože se ale jedná o dobu potřebnou pro zpracování 9 paketů, byla tato hodnota vydělena 9. V tabulce je tedy uvedeno 255 ns (zaokrouhleno dolů). Je ale nutné poznamenat, že systém FENIX pro počáteční zpracování toků používá malý rozhodovací strom implementovaný v síťovém přepínači Tofino. Autoři ale neuvádějí přesnost, ani latenci tohoto klasifikátoru, a proto je vlastní implementace srovnána pouze s klasifikátorem v FPGA.

Všetchna vlastní řešení mají inicializační interval zřetěžené linky roven 1. Propustnost lze tedy přímo určit z frekvence, na které bylo řešení implementováno. Např. pro frekvenci 200 MHz (použito u systému N3IC) dosahuje vlastní řešení propustnosti 200 milionů klasifikací (respektive zpracovaných paketů) za sekundu. Práce [33] uvádí, že systém N3IC je schopný provést pouze 1,75 milionů klasifikací za sekundu. Systém Kaleidoscope dosáhl propustnosti 28,5 milionů klasifikací za sekundu. U systému FENIX se propustnost FPGA akcelérátoru nepodařilo zjistit. Tento systém je také specifický tím, že předzpracování toků je provedeno v síťovém přepínači. Během hardwarových testů byl FENIX schopen zpracovat pomocí kombinace síťového přepínače a FPGA celkově 800 Gb/s.

Z provedeného srovnání vyplývá následující: (i) dokonce i nejmenší varianta vlastního řešení (**GRU(8, 4b)**) měla vyšší F1 skóre a přesnost klasifikace než všechny ostatní systémy, (ii) největší varianta vlastního řešení (**GRU(64, 4b)**) spotřebovala méně zdrojů než systémy Kaleidoscope a FENIX, (iii) všechny implementované varianty měly nižší latenci než všechny ostatní systémy, (iv) vlastní řešení mělo 114× vyšší propustnost než systém N3IC a 8,8× vyšší propustnost než systém Kaleidoscope a (v) většina vlastních řešení měla menší velikost stavu než systémy N3IC a FENIX.

Tabulka 5.3: Srovnání vlastních řešení s již existujícími metodami online klasifikace šifrovaného provozu (N3IC [33], Kaleidoscope [37] a FENIX [10]). U každého řešení je uvedeno F1 skóre (nebo přesnost v případě Kaleidoscope), využití FPGA zdrojů, latence (sloupec [ns]), počet inferencí (respektive zpracovaných paketů) za sekundu (sloupec MIPS z angl. *Millions Inferences Per Second*) a velikost stavu každého síťového toku v bitech (sloupec [b]). Pro vyhodnocení přesnosti byla použita datová sada ISCX-VPN-2016. Použitá FPGA a frekvence jsou uvedeny v jednotlivých částech tabulky. Vlastní řešení jsou značena jako GRU(H, B), kde H je velikost skrytého stavu a B je bitová šířka parametrů i aktivací.

Model	F_1 / A	LUT	BRAM	DSP	[ns]	MIPS	[b]
Virtex-7 (200 MHz)							
N3IC	F_1						
MLP (1b)	73,8 %	3k (0,6 %)	18 (1,2 %)	0 (0,0 %)	1000	1,75	256
Vlastní implementace							
GRU(8, 4b)	91,6 %	10k (2,2 %)	12 (0,8 %)	0 (0,0 %)	110	200	32
GRU(16, 4b)	93,4 %	21k (5,0 %)	15 (1,0 %)	0 (0,0 %)	150	200	64
GRU(32, 4b)	94,2 %	52k (12,0 %)	12 (0,8 %)	0 (0,0 %)	150	200	128
GRU(64, 4b)	95,1 %	127k (29,3 %)	15 (1,0 %)	0 (0,0 %)	250	200	256
Virtex UltraScale+ (250 MHz)							
Kaleid.	A						
MLP (8b)	92,9 %				256	28,5	0
CNN (8b)	96,1 %				2700	28,5	0
Celkem		225k (13,0 %)	887 (33,0 %)	5k (38,0 %)		28,5	0
Vlastní implementace							
GRU(8, 4b)	97,3 %	10k (0,6 %)	12 (0,4 %)	0 (0,0 %)	60	250	32
GRU(16, 4b)	97,8 %	22k (1,3 %)	12 (0,4 %)	0 (0,0 %)	60	250	64
GRU(32, 4b)	98,0 %	51k (3,0 %)	12 (0,4 %)	0 (0,0 %)	60	250	128
GRU(64, 4b)	98,3 %	127k (7,3 %)	1 (0,0 %)	0 (0,0 %)	120	250	256
Zynq UltraScale+ (300 MHz)							
FENIX	F_1						
RNN (8b)	87,3 %	134k (25,6 %)	62 (6,3 %)	91 (4,6 %)	255	–	243
CNN (8b)	89,2 %	201k (38,4 %)	70 (7,1 %)	159 (8,1 %)	255	–	243
RNN (FP32)	92,3 %	–	–	–	–	–	–
CNN (FP32)	91,1 %	–	–	–	–	–	–
Vlastní implementace							
GRU(8, 4b)	91,6 %	10k (1,9 %)	12 (1,2 %)	0 (0,0 %)	83	300	32
GRU(16, 4b)	93,4 %	22k (4,2 %)	12 (1,2 %)	0 (0,0 %)	83	300	64
GRU(32, 4b)	94,2 %	50k (9,6 %)	12 (1,2 %)	0 (0,0 %)	83	300	128
GRU(64, 4b)	95,1 %	118k (22,5 %)	12 (1,2 %)	0 (0,0 %)	133	300	256

Kapitola 6

Závěr

Cílem této diplomové práce je návrh neuronové sítě přizpůsobené pro online klasifikaci šifrovaného provozu a její implementace v FPGA. Vstupem navržené neuronové sítě jsou příznaky SPLT, obsahující velikosti paketů, jejich časové rozestupy a jejich směry v síťovém toku. Natrénovaný model dokáže rozpoznat konkrétní webové aplikace, které daný síťový tok vytvořily.

Na rozdíl od konvolučních sítí, které dominují současným offline metodám, byla zvolena rekurentní architektura. Její hlavní výhodou je schopnost inkrementálního zpracování paketů bez nutnosti jejich opakované analýzy. Navržený model je složený z embedding tabulek (pro které byly navrženy nové optimalizace) a z rekurentní vrstvy GRU nebo RHN (*Recurrent Highway Network*).

Protože pakety mohou na síťová rozhraní přicházet v různém pořadí, je nutné pro každý aktivní tok udržovat určitou stavovou informaci. Aby velikost tohoto stavu nebyla limitujícím faktorem celé implementace, bylo navrženo několik způsobů komprese stavové informace, včetně vektorové kvantizace skrytých stavů rekurentní buňky.

Navržená síť byla nejprve porovnána s velkou konvoluční sítí CESNET-MM-V2 [23] na datové sadě CESNET-QUIC22 [20]. Přestože se jedná o model učený pro offline klasifikaci, navržená rekurentní síť jej překonala z pohledu F1 skóre (o 0,97 %), počtu parametrů (navržený model obsahuje pouze 38,98 % parametrů konvoluční sítě) i z pohledu velikosti stavové informace (o 160 bitů méně na každý tok).

Parametry i aktivace navržené sítě byly úspěšně kvantizovány do 4 bitů pomocí metody QAT. F1 skóre kleslo pouze o 0,61 % v případě datové sady ISCX-VPN-2016 nebo o 2,5 % v případě datové sady CESNET-QUIC22. Následně bylo zjištěno, že existující nástroje pro implementaci neuronových sítí v FPGA (FINN a HLS4ML) spotřebují příliš mnoho hardwarových zdrojů. Proto byla vytvořena vlastní FPGA implementace pomocí nástroje HLS. Neuronová síť je transformována do zřetězené linky optimalizované přímo pro konkrétní hodnoty parametrů. Tím bylo možné ušetřit značné množství FPGA zdrojů a zároveň dosáhnout vysoké propustnosti a nízké latence.

Model implementovaný v FPGA byl porovnán se 3 existujícími metodami online klasifikace šifrovaného provozu (N3IC [33], Kaleidoscope [37] a FENIX [10]) na datové sadě ISCX-VPN-2016 [12]. Rekurentní model GRU překonal všechny 3 systémy z pohledu: F1 skóre, počtu FPGA zdrojů (LUT, FF, DSP i BRAM), latence, propustnosti i velikosti stavové informace. Například při srovnání modelu GRU(16, 4b) a systému FENIX, měl nový model o 4,2 % vyšší F1 skóre (93,4 %), spotřeboval 9× méně LUT a měl 3× nižší latenci (83 ns). Navíc oproti systému FENIX není omezen pouze na posledních 9 paketů síťového toku, ale může zpracovat libovolný počet paketů.

Navržené řešení stále nabízí prostor pro další rozšíření. Jedním z nich je schopnost určit jistotu klasifikace. To by umožnilo přesněji stanovit časový krok, od kterého je klasifikace dostatečně spolehlivá k provedení konkrétní akce (např. zablokování síťového toku). Některé způsoby určení jistoty klasifikace již byly prezentovány v práci [24].

Největší limitací současné FPGA implementace jsou plně propojené lineární vrstvy, které tvoří největší část spotřebovaných zdrojů FPGA. Nabízí se proto možnost prozkoumat pokročilejší metody kvantizace (např. neuniformní kvantizace), které by umožnily použít ještě menší bitové šířky bez výrazné ztráty přesnosti klasifikace.

V literatuře již byla navržena řada dalších optimalizací lineárních vrstev, které například redukuje bitovou šířku akumulátorů [6], snižují počet propojů lineárních vrstev [11] nebo lépe využívají konfigurovatelnou logiku FPGA [16]. Díky těmto optimalizacím by bylo možné v FPGA implementovat ještě větší modely s vyšší přesností klasifikace.

Literatura

- [1] AXLER, S. *Linear Algebra Done Right*. 3. vyd. Springer, 2015. ISBN 978-3-319-11079-0.
- [2] BAALEN, M. van; KUZMIN, A.; NAIR, S. S. et al. *FP8 versus INT8 for efficient deep learning inference* online. 2023. Dostupné z: <https://doi.org/10.48550/arXiv.2303.17951>. [cit. 2026-05-17].
- [3] BOUGUERRA, A.; MONTTOYA, D.; GOMEZ VILLA, A. et al. *Less Precise Can Be More Reliable: A Systematic Evaluation of Quantization's Impact on VLMs Beyond Accuracy* online. 2025. Dostupné z: <https://doi.org/10.48550/arXiv.2509.21173>. [cit. 2026-05-17].
- [4] CHEHADE, A.; RAGUSA, E.; GASTALDO, P. et al. Hardware-Aware Neural Architecture Search for Encrypted Traffic Classification on Resource-Constrained Devices. *IEEE Transactions on Network and Service Management* online, 2026. Dostupné z: <https://doi.org/10.1109/TNSM.2026.3666676>. [cit. 2026-05-17].
- [5] CHMIEL, B.; FISHMAN, M.; BANNER, R. et al. *FP4 All the Way: Fully Quantized Training of LLMs* online. 2025. Dostupné z: <https://doi.org/10.48550/arXiv.2505.19115>. [cit. 2026-05-17].
- [6] COLBERT, I.; PAPPALARDO, A.; PETRI KOENIG, J. et al. *A2Q+: Improving Accumulator-Aware Weight Quantization* online. 2024. Dostupné z: <https://doi.org/10.48550/arXiv.2401.10432>. [cit. 2026-05-17].
- [7] DARVISH ROUHANI, B. et al. *OCP Microscaling Formats (MX) Specification* online. Specification. Open Compute Project, září 2023. Dostupné z: <https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf>. [cit. 2026-05-17].
- [8] FAUVEL, K.; CHEN, F. a ROSSI, D. *A Lightweight, Efficient and Explainable-by-Design Convolutional Neural Network for Internet Traffic Classification* arXiv preprint. 2023. Dostupné z: <https://arxiv.org/abs/2202.05535>. [cit. 2026-05-17].
- [9] GAL, Y. a GHAHRAMANI, Z. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks* online. 2016. Dostupné z: <https://arxiv.org/abs/1512.05287>. [cit. 2026-05-17].
- [10] GAO, X.; LI, T.; ZHANG, Y. et al. *FENIX: Enabling In-Network DNN Inference with FPGA-Enhanced Programmable Switches* online. 2025. [cit. 2026-05-17].

- [11] GHOLAMI, A.; KIM, S.; DONG, Z. et al. *A Survey of Quantization Methods for Efficient Neural Network Inference* online. 2021. Dostupné z: <https://doi.org/10.48550/arXiv.2103.13630>. [cit. 2026-05-17].
- [12] GIL, G. D.; LASHKARI, A. H.; MAMUN, M. S. I. et al. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In: SciTePress. *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP)* online. 2016, s. 407–414. [cit. 2026-05-17].
- [13] GONGYO, S.; LIANG, J.; AMBAI, M. et al. Learning Non-Uniform Step Sizes for Neural Network Quantization. In: *Proceedings of the Asian Conference on Computer Vision (ACCV)* online. Prosinec 2024, s. 4385–4402. Dostupné z: https://openaccess.thecvf.com/content/ACCV2024/papers/Gongyo_Learning_Non-Uniform_Step_Sizes_for_Neural_Network_Quantization_ACCV_2024_paper.pdf. [cit. 2026-05-17].
- [14] GORISHNIY, Y.; RUBACHEV, I. a BABENKO, A. *On Embeddings for Numerical Features in Tabular Deep Learning* online. 2022. [cit. 2026-05-17].
- [15] GRAVES, A. *Adaptive Computation Time for Recurrent Neural Networks* online. 2016. [cit. 2026-05-17].
- [16] GUO, Z. *A Survey on LUT-based Deep Neural Networks Implemented in FPGAs* online. 2025. Dostupné z: <https://doi.org/10.48550/arXiv.2506.07367>. [cit. 2026-05-17].
- [17] HASANPOUR, M. A.; FAFOUTIS, X. a ROVERI, M. A Survey of Quantization Techniques in Embedded AI Toolchains. In: *Proceedings of the 2025 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)* online. 2026. Dostupné z: <https://doi.org/10.1109/AIoT66900.2025.00079>. [cit. 2026-05-17].
- [18] JANG, E.; GU, S. a POOLE, B. *Categorical Reparameterization with Gumbel-Softmax* online. 2016. Dostupné z: <https://arxiv.org/abs/1611.01144>. [cit. 2026-05-17].
- [19] JERABEK, K.; LUXEMBURK, J.; PLNÝ, R. et al. *When Simple Model Just Works: Is Network Traffic Classification in Crisis?* online. 2025. Dostupné z: <https://doi.org/10.48550/arXiv.2506.08655>. [cit. 2026-05-17].
- [20] KOUMAR, J.; HYNEK, K.; PEŠEK, J. et al. CESNET-QUIC22: A large one-month QUIC network traffic dataset from backbone lines. *Data in Brief* online. 1. vyd. Elsevier, Únor 2023, sv. 46, s. 108875. Dostupné z: <https://doi.org/10.1016/j.dib.2023.108875>. [cit. 2026-05-17].
- [21] KOUMAR, J.; HYNEK, K.; PEŠEK, J. a ČEJKA, T. NetTiSA: Extended IP flow with time-series features for... *Computer Networks* online. Elsevier, 2024, sv. 240, s. 110147. ISSN 1389-1286. Dostupné z: <https://doi.org/10.1016/j.comnet.2023.110147>. [cit. 2026-05-17].
- [22] LUXEMBURK, J.; HYNEK, K.; PLNY, R. et al. *Universal Embedding Function for Traffic Classification via QUIC Domain Recognition Pretraining: A Transfer Learning Success* online. Únor 2025. Dostupné z: <https://arxiv.org/abs/2502.12930>. [cit. 2026-05-17].

- [23] LUXEMBURK, J.; HYNEK, K. a ČEJKA, T. Encrypted traffic classification: the QUIC case. In: *2023 7th Network Traffic Measurement and Analysis Conference (TMA)* online. Prague, Czech Republic: IFIP, 2023. ISBN 978-3-903176-58-4. Dostupné z: <https://doi.org/10.23919/TMA58422.2023.10199052>. [cit. 2026-05-17].
- [24] LUXEMBURK, J. a ČEJKA, T. Fine-grained TLS services classification with reject option. *Computer Networks* online, 2022, sv. 220, s. 109467. Dostupné z: <https://doi.org/10.1016/j.comnet.2022.109467>. [cit. 2026-05-17].
- [25] MIYASHITA, D.; LEE, E. H. a MURMANN, B. *Convolutional Neural Networks using Logarithmic Data Representation* online. 2016. Dostupné z: <https://arxiv.org/abs/1603.01025>. [cit. 2026-05-17].
- [26] MORITA, T. *Pushing Toward the Simplex Vertices: A Simple Remedy for Code Collapse in Smoothed Vector Quantization* online. Září 2025. Dostupné z: <https://doi.org/10.48550/arXiv.2509.22161>. [cit. 2026-05-17].
- [27] NASCITA, A.; MONTIERI, A.; ACETO, G. et al. Improving Performance, Reliability, and Feasibility in Multimodal Multitask Traffic Classification with XAI. *IEEE Transactions on Network and Service Management* online. IEEE, Červen 2023, sv. 20, č. 2, s. 1267–1289. Dostupné z: <https://doi.org/10.1109/TNSM.2023.3246794>. [cit. 2026-05-17].
- [28] PASCANU, R.; GULCEHRE, C.; CHO, K. et al. *How to Construct Deep Recurrent Neural Networks* online. 2013. [cit. 2026-05-17].
- [29] RESCORLA, E.; OKU, K.; SULLIVAN, N. et al. *TLS Encrypted Client Hello* Internet-Draft draft-ietf-tls-esni-25 (work in progress). Internet Engineering Task Force, červen 2025. Dostupné z: <https://datatracker.ietf.org/doc/draft-ietf-tls-esni/>. [cit. 2026-05-17].
- [30] SCHULTE, J.-F.; RAMHORST, B.; SUN, C. et al. *hls4ml: A Flexible, Open-Source Platform for Deep Learning Acceleration on Reconfigurable Hardware* online. 2025. Dostupné z: <https://doi.org/10.48550/arXiv.2512.01463>. [cit. 2026-05-17].
- [31] SHARMA, A. a HABIBI LASHKARI, A. A survey on encrypted network traffic: A comprehensive survey of... *Computer Networks* online. Elsevier, Únor 2025, sv. 257, s. 110984. Dostupné z: <https://doi.org/10.1016/j.comnet.2024.110984>. [cit. 2026-05-17].
- [32] SHOHAM, R. a PERMUTER, H. Highway State Gating for Recurrent Highway Networks: Improving Information Flow Through Time. In: *Proceedings of the Second International Symposium on Cyber Security Cryptography and Machine Learning (CSCML 2018)* online. Springer, 2018, s. 120–128. [cit. 2026-05-17].
- [33] SIRACUSANO, G.; GALEA, S.; SANVITO, D. et al. Re-architecting Traffic Analysis with Neural Network Interface Cards. In: *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI '22)* online. Renton, WA, USA: USENIX Association, Duben 2022, s. 513–533. Dostupné z: <https://www.usenix.org/system/files/nsdi22-paper-siracusano.pdf>. [cit. 2026-05-17].

- [34] SONG, Z.; ZHAO, Z.; ZHANG, F. et al. I²RNN: An Incremental and Interpretable Recurrent Neural Network for Encrypted Traffic Classification. *IEEE Transactions on Dependable and Secure Computing* online. IEEE, 2023, PP, č. 99, s. 1–14. Dostupné z: <https://doi.org/10.1109/TDSC.2023.10056861>. [cit. 2026-05-17].
- [35] SUN, C.; QUE, Z.; ÅRRESTAD, T. K. et al. *HGQ: High Granularity Quantization for Real-time Neural Networks on FPGAs* online. 2024. Dostupné z: <https://doi.org/10.48550/arXiv.2405.00645>. [cit. 2026-05-17].
- [36] UMUROGLU, Y. a JAHRE, M. *Streamlined Deployment for Quantized Neural Networks* online. 2017. Dostupné z: <https://doi.org/10.48550/arXiv.1709.04060>. [cit. 2026-05-17].
- [37] WEN, D.; LIU, Z.; YANG, T. et al. *Inference-to-complete: A High-performance and Programmable Data-plane Co-processor for Neural-network-driven Traffic Analysis* online. Listopad 2024. [cit. 2026-05-17].
- [38] WICKRAMASINGHE, N.; SHAGHAGHI, A.; TSUDIK, G. a JHA, S. SoK: Decoding the Enigma of Encrypted Network Traffic Classifiers. In: *2025 IEEE Symposium on Security and Privacy (SP)* online. IEEE, Květen 2025, s. 1825–1843. Dostupné z: <https://doi.org/10.1109/SP61157.2025.00165>. [cit. 2026-05-17].
- [39] WU, H.; JUDD, P.; ZHANG, X. et al. *Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation* online. 2020. Dostupné z: <https://doi.org/10.48550/arXiv.2004.09602>. [cit. 2026-05-17].
- [40] YAN, J.; XU, H.; LIU, Z. et al. *Brain-on-Switch: Towards Advanced Intelligent Network Data Plane via NN-Driven Traffic Analysis at Line-Speed* online. 2024. Dostupné z: <https://doi.org/10.48550/arXiv.2403.11090>. [cit. 2026-05-17].
- [41] ZHAO, R.; ZHAN, M.; DENG, X. et al. Yet Another Traffic Classifier: A Masked Autoencoder Based Traffic Transformer with Multi-Level Flow Representation. In: *Proceedings of the AAAI Conference on Artificial Intelligence* online. 2023, sv. 37, č. 4. Dostupné z: <https://doi.org/10.1609/aaai.v37i4.25674>. [cit. 2026-05-17].
- [42] ZHOU, H.; ZHENG, X.; WANG, Y. et al. *GhostRNN: Reducing State Redundancy in RNN with Cheap Operations* online. 2024. Dostupné z: <https://arxiv.org/abs/2411.14489>. [cit. 2026-05-17].
- [43] ZILLY, J. G.; SRIVASTAVA, R. K.; KOUTNÍK, J. et al. Recurrent Highway Networks. In: PMLR. *Proceedings of the 34th International Conference on Machine Learning* online. 2017, s. 4189–4198. [cit. 2026-05-17].