



**Università
di Genova**

University of Genoa

Department of Electrical, Electronic, Telecommunications Engineering,
and Naval Architecture (DITEN)

From Cloud to Edge: A Quantized, Architecture-Searched Recommender for a Privacy-Preserving Multilingual RSS Reader

MSc IN ENGINEERING TECHNOLOGY FOR STRATEGY AND SECURITY

Supervisors:

Prof. Francesco Bellotti
Prof. Riccardo Berta

Candidate:

Houriyeh Emadoleslami

October 2026

Co-Supervisor:

Sepehr Mohammady

Academic Year 2025–2026

Declaration

I declare that this thesis is original and reports work carried out by me during my internship at the ELIOS Laboratory of the Department of Electrical, Electronic, Telecommunications Engineering and Naval Architecture (DITEN), University of Genoa, under the supervision of Prof. Francesco Bellotti and Prof. Riccardo Berta and the day-to-day guidance of my co-supervisor, Sepehr Mohammady, whose PhD research line this work belongs to.

Most of the work reported here is my own, carried out under that supervision: the byte-level content encoder and its distillation, the architecture search, the quantisation and binarisation experiments, their evaluation on MIND and xMIND, the on-device personalisation engine of FEEDWELL-EDGE, and the analysis in the chapters that follow. The base application FEEDWELL predates the internship and is the work of my co-supervisor; FEEDWELL-EDGE is built on it. The audit of the June 2026 cost model and the October 2026 work that Chapter 5 reports alongside the June experiments (the seeded reruns, the searches under the board budgets and the measurements on the boards and on the phone) were carried out within the same supervised project and are reported with their dates and commit identifiers so that each number can be traced to the run that produced it.

Every quantitative result in this thesis comes from a committed result file of the MIND-EDGE-RECOMMENDER repository or from the datasets themselves, and prior work by others is acknowledged through citation. Estimates and off-device figures, such as the energy proxy, the estimated flash and RAM figures and the laptop timings, are labelled as such in the text and are not presented as on-device measurements.

Genoa, October 2026

Houriyeh Emadoleslami

Abstract

A news recommender that runs on the reader’s own device can personalise the feed without sending reading behaviour to a server, and it keeps working offline. On a microcontroller, the main obstacle is the word-embedding table of a conventional news encoder. In the Neural news Recommendation with Multi-head Self-attention (NRMS) model trained in this thesis as a reference, a table of 34,304 words initialised from Global Vectors (GloVe) holds 10.29 of the 11.29 million parameters and takes about 39 MB in 32-bit floating point, nearly twenty times the 2 MB of flash on a typical microcontroller. Each further language would need a table of its own.

This thesis removes the table by reading titles as raw UTF-8 bytes, so every language uses the same 256 input values. The encoder is a small byte-level convolutional network, chosen by an evolutionary search in three settings: with no resource limit, within 256 KB of flash and 128 KB of memory, and with one-bit weights in mind. Each chosen architecture is trained with three seeds and tested with inner-layer weights at 32, 8 and 1 bits. The constrained encoder, with 68 thousand parameters, is also distilled from a frozen multilingual sentence-transformer with a Matryoshka cosine objective before training on clicks from the Microsoft News Dataset (MIND).

On the development impressions of MINDsmall, the constrained encoder with eight-bit weights reaches an area under the receiver operating characteristic curve (AUC) of 0.614 at an estimated 204 KB, against 0.664 for NRMS with GloVe (means over three seeds). At full precision, distillation raises it from 0.618 to 0.629 with 165 times fewer parameters than NRMS. Training on clicks shown in a random language lifts the mean AUC over the fourteen translations of the multilingual extension xMIND from 0.557 to 0.589 and costs 0.020 in English. Eight-bit weights change AUC by -0.005 to $+0.011$, within the spread over seeds, and cut the estimated energy by a factor of 3.4 to 9.3. One-bit weights stay below every eight-bit model even after recovery training.

As an eight-bit file, the distilled encoder loses 0.004 AUC on average. It encodes a title in 31.19 ms on an STM32H7B3I-DK board, within 142 KB of flash and 49.4 KB of working memory, and in 449.64 ms on the smaller NUCLEO-F401RE. A second search with μ NAS, under the budget of each board, finds encoders that rank slightly better than the best hand-designed ones, and its NUCLEO-F401RE encoder runs in 60.6 ms against 88.7 ms. On the DOOGEE S98Pro phone running the application, the eight-bit encoder needs 0.53 ms per title on one core and about 0.95 mJ per title.

Without the vocabulary table, architecture search and integer quantisation can shrink the encoder, and eight-bit weights are the recommended operating point. The encoder’s content score is now added by default to the heuristic ranking of FEEDWELL-EDGE 2.1.0, an RSS reader built during the internship on FEEDWELL 1.18.2. In an offline MINDsmall replay this gains 0.024 AUC, but no user has tried it yet.

Acknowledgements

I thank my supervisors, Prof. Francesco Bellotti and Prof. Riccardo Berta, for their guidance throughout this work and for the opportunity to carry it out at the ELIOS Laboratory. Their questions shaped the direction of the thesis, and their insistence on reporting only what had been measured shaped the way its results are presented.

I am especially grateful to my co-supervisor and tutor, Sepehr Mohammady, for his technical guidance and mentorship during the internship. His patience in discussing the design of the system, and his help whenever an experiment failed, made it possible to complete this project.

Finally, I thank my family and friends for their encouragement and support throughout this degree. This work is dedicated to them.

Contents

Declaration	i
Abstract	ii
Acknowledgements	iii
Abbreviations	x
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Device Constraint	2
1.4 Research Questions	4
1.5 Scope	5
1.6 Contributions	6
1.7 Internship Context	7
1.8 Outline	7
2 Background	8
2.1 News Recommendation	8
2.2 Multilingual Recommendation	10
2.3 On-Device Inference	11
2.4 Quantisation	12
2.5 Binarisation	13
2.6 Architecture Search	14
2.7 Distillation and Bytes	15
2.8 Table Compression	17
2.9 Research Gap	18
3 Methodology	20
3.1 System Overview	20
3.2 Hardware Budgets	21
3.3 Byte-Level Encoder	22
3.3.1 Input	22
3.3.2 Architecture	22
3.4 Distillation	23
3.5 Recommender	24
3.6 Architecture Search	25

3.7	Board-Specific Search	26
3.8	Precision Scheme	27
3.9	Cost Model	28
3.10	Protocols	29
3.11	Reference Conditions	30
4	Implementation	32
4.1	Datasets	32
4.2	Software	33
4.3	Export	33
4.4	Hardware	34
4.5	Audit	35
4.6	Target Application	36
4.6.1	Personalisation engine	37
5	Results	41
5.1	Reference Points	41
5.2	Search Outcomes	43
5.3	Precision	43
5.4	Binarisation	45
5.5	Cross-Lingual Transfer	46
5.6	History Length	47
5.7	Cost Trade-off	48
5.8	Board and Phone Measurements	49
5.9	Searches Under the Board Budgets	53
5.10	In-App Ranking	55
5.11	Discussion	55
5.12	Limitations	57
6	Conclusion	59
6.1	Summary	59
6.2	Future Work	60
6.2.1	Remaining measurements	60
6.2.2	Search fitness	60
6.2.3	Live application evaluation	61
6.2.4	On-device adaptation	61
6.2.5	Multilingual coverage	61
6.2.6	User study	61
6.3	Closing Remarks	61
A	Configuration	63

B	Complete Results	67
B.1	Matrix rerun	67
B.2	June ablations and one-bit rerun	67
B.3	June languages	68
B.4	Board-search finalists	68
B.5	October reference conditions	69
C	FeedWell-Edge Engine	71
D	Reproducibility	73
D.1	Repositories	73
D.2	Environment	73
D.3	Board, phone and replay runs	74
D.4	Figures and tables	74
D.5	Data manifest	75
	References	77

List of Figures

1.1	Two kinds of privacy guarantee	1
1.2	The embedding-table memory wall	3
1.3	Roadmap of the thesis.	7
2.1	News-encoder/user-encoder factorisation	9
2.2	Timeline of models and benchmarks	9
2.3	Memory hierarchy across tiers	11
2.4	Quantisation-aware training	13
2.5	The Bi-Real shortcut	14
2.6	Evolutionary search loop	15
2.7	Multilingual distillation	16
2.8	Byte input versus word tables	16
2.9	Responses to the table footprint	17
2.10	Quotient-remainder embeddings	18
2.11	Positioning of this work	19
3.1	System overview	21
3.2	The byte-level encoder	23
3.3	Depthwise-separable block	23
3.4	Training and evaluation	25
3.5	The precision axis	28
4.1	Application structure	37
4.2	The FEEDWELL-EDGE learning loop	38
4.3	Export interface	38
5.1	AUC against encoder parameters	42
5.2	Architectures selected by each arm	43
5.3	AUC cells of the matrix	45
5.4	AUC against size and energy	46
5.5	Size breakdown	47
5.6	One-bit recovery	48
5.7	AUC per translated set	50
5.8	AUC by history length	51
5.9	Board latency per title	53
5.10	Headline results	57
6.1	Staged path	60

List of Tables

1.1	Deployment tiers	3
1.2	Research questions	4
2.1	Neural news-recommendation models	10
2.2	The xMIND languages	10
2.3	Inference runtimes	12
2.4	PTQ versus QAT	13
2.5	Binarisation techniques	14
2.6	Search strategies	15
2.7	Embedding-table responses	18
3.1	Pipeline stages	20
3.2	Hardware tiers	22
3.3	Stored size per layer group	23
3.4	Search space	26
3.5	Per-operation energy	28
3.6	Training protocols	30
3.7	Reference conditions	31
4.1	Dataset statistics	32
4.2	Modules and scripts	39
4.3	The measurement laptop.	40
4.4	Learning-loop parameters	40
5.1	Reference points	41
5.2	Architecture $\times$ precision matrix	44
5.3	AUC per language	49
5.4	AUC by history length	49
5.5	Cost per title	51
5.6	Accuracy of deployed files	52
5.7	Board benchmarks	52
5.8	Encoders under the board budgets	54
5.9	Answers to the research questions	56
A.1	Run-script overrides	66
B.1	Matrix values per seed	67
B.2	June ablations	68
B.3	June per-language metrics	68

B.4	Board-search finalists	69
B.5	October values per seed	70
B.6	October run durations	70
D.1	Repositories and commits	73

Abbreviations

AUC	Area Under the ROC Curve, computed per impression
BNN	Binarised Neural Network
CMSIS-NN	Neural-network kernels of the Cortex Microcontroller Software Interface Standard
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSCnv	Depthwise-Separable Convolution
EMA	Exponential Moving Average
FP32	32-bit floating-point precision
GDPR	General Data Protection Regulation (Regulation (EU) 2016/679)
GPU	Graphics Processing Unit
INT8	8-bit integer precision
IoT	Internet of Things
JSON	JavaScript Object Notation
KD	Knowledge Distillation
LCE	Larq Compute Engine
MAC, MMAC	Multiply–accumulate operation; millions of them per inference
MCU	Microcontroller Unit
MIND	Microsoft News Dataset
MRR	Mean Reciprocal Rank
NAS	Neural Architecture Search
nDCG	Normalised Discounted Cumulative Gain
NLLB	No Language Left Behind (translation model)
NLP	Natural Language Processing
NRMS	Neural news Recommendation with Multi-head Self-attention
ONNX	Open Neural Network Exchange
ORT	ONNX Runtime
PTQ	Post-Training Quantisation
QAT	Quantisation-Aware Training
QDQ	Quantise–dequantise, the ONNX representation of an 8-bit model
QR	Quotient–Remainder (compositional) embeddings
RAM	Random-Access Memory
ROC	Receiver Operating Characteristic

RSS	Really Simple Syndication
SIMD	Single Instruction, Multiple Data
SRAM	Static Random-Access Memory
STE	Straight-Through Estimator
TFLite, TFLM	TensorFlow Lite; TensorFlow Lite for Microcontrollers
TinyML	Machine learning on microcontroller-class devices
URL	Uniform Resource Locator
UTF-8	8-bit Unicode Transformation Format
XNOR	Exclusive NOR (bitwise operation)
xMIND	Multilingual extension of MIND (fourteen translated title sets)

Chapter 1

Introduction

1.1 Motivation

Personalised content ranking is one of the most widely deployed applications of machine learning and also one of the most privacy-invasive. A news recommender infers a reader’s interests from the articles they open, the time they spend reading and the articles they skip. Over time this record can reveal political opinions, religious beliefs, health concerns and sexual orientation, four of the categories that Article 9 of the General Data Protection Regulation (GDPR) treats as special personal data [1]. In the usual deployment the record is sent to a remote service and stored there, so the reader’s privacy depends on the provider’s policies and retention practices.

Running the recommender on the reader’s own device changes the nature of this guarantee. If the ranking model runs on the phone or on an embedded reading device, and the interaction history stays in local storage, privacy depends on where the computation takes place and not on a data-handling policy that the reader has to trust. Local inference also meets a second requirement: a reader on an unreliable or metered connection keeps personalised ranking when the connection drops, because ranking does not need a round trip to a server. Figure 1.1 contrasts the two arrangements.

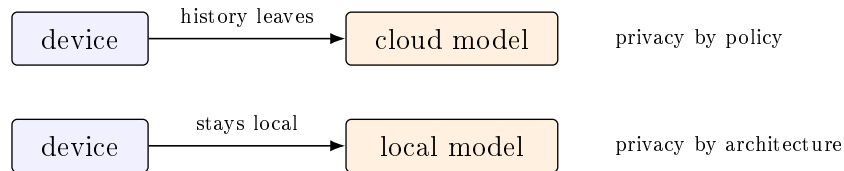


Figure 1.1: Two kinds of guarantee. In the cloud model (top) the behavioural history is transmitted and privacy rests on a policy; on the device (bottom) the history never leaves local storage and privacy follows from the system architecture.

This motivation matches the research agenda of the ELIOS Laboratory at DITEN, where the GEMINI project (Green Machine Learning for the IoT, PRIN 2022) studies machine learning that runs locally on battery-powered and energy-harvesting devices [2]. The difficulty is capacity: the devices for which the privacy argument is strongest are also the most constrained, and neural recommenders are normally trained and served on hardware with orders of magnitude more memory than these devices provide.

1.2 Problem Statement

The problem this thesis addresses is the embedding-table memory wall. A neural news recommender has to turn a headline into a vector before it can rank anything, and the standard construction reserves one row of a lookup table for each word in the vocabulary. The reference model trained in Chapter 5 is Neural news Recommendation with Multi-head Self-attention (NRMS) with the 300-dimensional Global Vectors (GloVe) embedding table distributed with the Microsoft News Dataset (MIND). Its table over 34,304 words holds 10.29 million of the model’s 11.29 million parameters (91%) and occupies 39 MB in 32-bit floating point. A reduced NRMS with a randomly initialised 256-dimensional table over the 25,355 words of the training titles has 7.08 million parameters, 92% of them in the table, and occupies 27 MB. The table performs no multiplications: each word’s vector is read from memory by its index.

Most of the techniques that make neural networks small target computation, and a lookup table involves almost none. Hardware-aware architecture search minimises the operations per inference, so it does not account for a layer that performs no operations. Eight-bit quantisation shrinks a table fourfold, and Chapter 5 confirms that it does so without loss for NRMS, but the quantised reduced NRMS still needs 6.8 MB, more than three times the flash of the target microcontroller. Binarisation replaces multiply-accumulates with bitwise operations, which a lookup table does not perform. The multilingual setting makes the problem worse. A vocabulary is language-specific, so a conventional design that serves n languages needs n tables, and the table memory grows linearly with the number of languages while the network stays the same size (Figure 1.2).

The problem lies in the input representation. The standard representation puts most of the cost in a structure that the usual compression techniques reduce only in part, and that cost grows with the number of languages.

Research problem

Can a multilingual news recommender be built that fits microcontroller-class memory, stays close to the ranking quality of a word-embedding model two orders of magnitude larger, and covers many languages with one model rather than one per language? With the vocabulary table removed, what do architecture search and precision reduction cost and gain on the network that remains?

1.3 Device Constraint

A mobile Really Simple Syndication (RSS) reader runs on a smartphone, and so does the application built in this work, but the design constraint is taken from the tier below, the microcontroller. Table 1.1 summarises the three deployment tiers. A cloud server has effectively no memory limit but gives up the privacy property that motivates the work. A modern smartphone, with gigabytes of random-access memory (RAM)

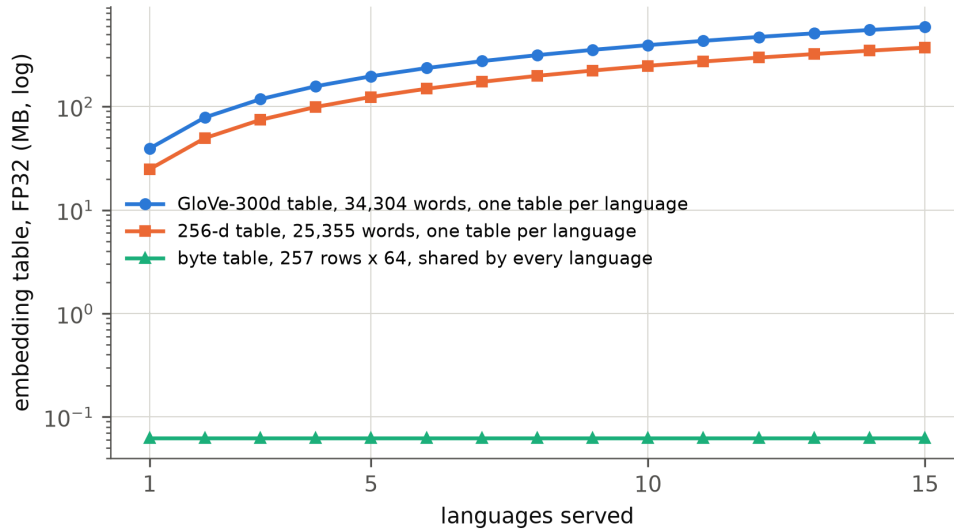


Figure 1.2: The embedding-table memory wall. If every language needed a word table the size of the English ones trained in this thesis (the 34,304-word GloVe table of the reference NRMS and the 25,355-word table of the reduced NRMS), storage would grow linearly with the number of languages. The byte table of the encoder proposed here has 257 rows of 64 values (64 KB) and is shared by every language. Vertical axis logarithmic.

and a neural accelerator, can run a word-embedding model; it hosts the application built here, but its capacity does not force a change of representation. A Cortex-M7 microcontroller such as the STM32H7 series has about two megabytes of flash and one megabyte of static random-access memory (SRAM), little or no operating system and an energy budget of milliwatts. A word-embedding table of the size above does not fit on such a device, and the memory cost of every component matters.

Table 1.1: The deployment spectrum and what each tier constrains.

Tier	Memory	Role here	What it forces
Cloud server	effectively none	rejected	nothing, but surrenders privacy
Smartphone (Android)	(Android) several GB	host of FEEDWELL-EDGE	little; word models still fit
Microcontroller (Cortex-M7, STM32H7)	~2 MB flash, ~1 MB SRAM	binding constraint	removal of the table; low precision

The microcontroller is chosen as the design target for methodological reasons: a model that fits the most constrained tier also fits every tier above it. FEEDWELL-EDGE runs on the smartphone, but the research question is defined by the microcontroller. The budget of the June search (256 KB of flash and 128 KB of RAM) is about eight times tighter than the STM32H7 limits, so that the constraint excludes part of the

search space instead of being met by every compact text encoder. A later search with μ NAS uses one budget per board, including a bound on the multiply-accumulates per title (Section 3.7). Measurements were made on the STM32H7B3I-DK and NUCLEO-F401RE boards through the ST Edge AI Developer Cloud and on the Android phone that hosts the application (Section 5.8); Chapter 4 states how each quantity in this thesis was obtained.

1.4 Research Questions

The aim of this thesis is to design, implement and evaluate a multilingual news-recommendation model that operates within microcontroller-class resource budgets, and to characterise its trade-offs between accuracy, memory and energy once the embedding table has been removed. The work is organised around five research questions, each answered by a specific part of the experiments (Table 1.2).

Table 1.2: Research questions mapped to the components that answer them.

RQ	Theme	Question	Answered by
RQ1	Representation	Does a distilled byte-level encoder carry a partly language-agnostic representation, so that it ranks translated titles with no per-language table?	Cross-lingual evaluation (Section 5.5)
RQ2	Architecture	When the search is held to an on-device budget, what topology is selected and how much quality is given up against an unconstrained search?	NAS and Micro-NAS arms; board searches (Sections 5.2 and 5.9)
RQ3	Precision	How does ranking quality vary with weights at 32, 8 and 1 bits, and where does the loss stop being acceptable?	Precision sweep (Section 5.3)
RQ4	Recovery	Where naive binarisation collapses, can a targeted training recipe restore a usable operating point, and at what footprint?	Binarisation experiments (Section 5.4)
RQ5	Reference	How far is the byte-level encoder from a word-vocabulary NRMS and from scorers that use MIND’s editorial labels, and what does distillation add?	Seeded reference runs (Section 5.1)

The questions lead to five objectives, addressed in the later chapters: (i) to build a byte-level encoder and a distillation procedure that transfers a multilingual teacher’s representation into it; (ii) to design and run a hardware-aware search with a budget that actually restricts it; (iii) to define and apply an integer-quantisation and binarisation scheme, including the recovery techniques that one-bit weights need; (iv) to assemble the datasets, the models and the measurement environment, and to prepare a host application whose event pipeline and storage schema the encoder is exported to; and (v) to report ranking quality, multilingual coverage and cost over the full grid

of architectures and precisions, together with the reference points needed to interpret them.

1.5 Scope

The scope covers the model that performs on-device multilingual news ranking, the pipeline that produces it and the application that hosts it. The results rest on three assumptions. First, the small configuration of MIND is representative enough for comparative conclusions: it keeps the absolute ranking numbers below what the large configuration would give, but the ordering of architectures and precisions and the comparison with the reference points are assumed not to depend on scale. Second, a frozen multilingual sentence-transformer is an adequate teacher; the distilled representation inherits its quality, and a weak teacher would limit the student whatever the compression. Third, UTF-8 bytes are a sufficient input representation for short news headlines, an assumption that the multilingual results test.

The following limits of scope apply.

- The encoder runs inside the application and its content score is part of the ranking, but its effect on ranking is measured only in an offline replay of MINDsmall, with MIND categories standing in for the reader’s topics (Section 5.10). A live comparison on real reading behaviour is future work.
- Latency, flash and RAM are measured on two STM32 boards for the three June architectures and for the encoders of the board searches, and latency and energy on the phone for the deployed eight-bit encoder; energy on the boards is not measured. For the nine-variant matrix, size and RAM are estimated from parameter and buffer counts, energy is a per-operation proxy and latency is a laptop timing.
- The eight-bit and one-bit results of the matrix are simulated on the weights of the inner convolutions, with activations, the byte embedding and the outer layers in floating point. The deployable eight-bit file of the constrained encoder, with eight-bit weights and eight-bit inputs to every convolution and to the output layer, is measured separately (Section 5.8); the one-bit model has no deployable file yet.
- The architecture-search run of June 2026 enforced its flash and RAM budgets but not its operation budget. Its operation counter skipped the quantised layers, and both constrained winners exceed the 2 million multiply-accumulate bound they were meant to respect. The costs reported here were recomputed with a corrected counter in October 2026. The June search was not rerun; the μ NAS searches of Section 3.7 enforce all three bounds instead, with one budget per board.
- No user study was conducted, so no claim is made about perceived recommendation quality. No new ranking algorithm is proposed either: the recommender uses a standard dot-product scorer, so the effect of the encoder and its precision is not mixed with the effect of a new ranking mechanism.

Chapter 6 returns to several of these limits as future work.

1.6 Contributions

The thesis makes six contributions.

1. The per-language vocabulary table is identified as the main memory cost of on-device multilingual news ranking. Reading titles as bytes removes it, which makes the model size independent of the number of languages and leaves a network whose size is set by its arithmetic layers, so that the techniques of machine learning for microcontroller-class devices (TinyML) apply to it.
2. A multilingual byte-level convolutional student of 68 thousand parameters is distilled from a frozen multilingual sentence-transformer with a Matryoshka cosine objective and carries part of the teacher’s cross-lingual alignment without any word or subword table.
3. An evolutionary search in three arms, crossed with weights at 32, 8 and 1 bits, gives nine variants that are trained with three seeds and costed with one per-layer counter for size, RAM, operations and energy; the June 2026 costs were recomputed after an audit found that the original counter skipped the quantised layers. A second search with μ NAS, under the memory and operation budgets of two STM32 boards and with the final choice made after full training, gives one encoder per board: 0.605 and 0.608 AUC, against 0.599 and 0.589 for the best hand-designed shapes of the same budgets.
4. Three-seed measurements place the encoder against NRMS with GloVe vectors (0.664 AUC), a reduced NRMS (0.604) and histogram scorers over MIND’s editorial labels (0.598 and 0.617). The distilled byte-level encoder scores 0.629 with 165 times fewer parameters than GloVe NRMS, and the eight-bit constrained variant scores 0.614 at an estimated 204 KB. One-bit weights cost 0.029 to 0.065 AUC; recovery training raises the constrained one-bit model from 0.554 to 0.587 at 194 KB, still below every eight-bit model, which is reported as a negative result. Exported as an eight-bit file, the distilled constrained encoder loses 0.004 AUC on average over three seeds and runs in 31.19 ms per title on an STM32H7B3I-DK and in 449.64 ms on a NUCLEO-F401RE.
5. One set of weights ranks titles in fifteen languages. Distillation raises the mean AUC over the fourteen translations from 0.528 to 0.557, training on clicks shown in a random language raises it to 0.589, and the per-language results show which languages an English-anchored encoder serves worst.
6. FEEDWELL-EDGE 2.1.0 is an on-device personalisation engine, rebuilt during the internship on the public RSS reader FEEDWELL 1.18.2 with its own application identity and installed on an Android phone. An export interface writes the trained encoder and a cold-start prior in the application’s storage schema. The eight-bit encoder is bundled into the reader and its content score is added to the heuristic ranking; in an offline replay of MINDsmall this adds 0.024 AUC to the heuristic rule, and the application now uses the combined score by default.

1.7 Internship Context

The internship produced two kinds of artifact, which should be distinguished when assessing the contribution. The first is the neural research pipeline: the byte-level encoder, the search and the quantisation study, which are the subject of Chapters 3 to 5 and are kept in the MIND-EDGE-RECOMMENDER repository. The second is application engineering. FEEDWELL, a complete and publicly released RSS reader, is the work of the co-supervisor and predates the internship; FEEDWELL-EDGE, the on-device personalisation engine, was built on top of it during the internship.

The FEEDWELL-EDGE engine is a heuristic system. It records interaction events locally, maintains per-topic interest weights, detects shifts in interest by comparing a fast and a slow exponential moving average (EMA) and re-ranks the feed with a transparent scoring rule. In this thesis it has two roles: it hosts the trained encoder, and it captures the behavioural events from which an on-device recommender could later learn. The eight-bit encoder is now bundled into the application, and its content score is added to the heuristic ranking through the engine’s storage schema. The evidence for this step is an offline replay of MINDsmall, in which the combined score is 0.024 AUC above the heuristic alone (Section 5.10); a live evaluation on readers’ behaviour is future work. Section 4.6 describes the engineering of FEEDWELL-EDGE and Appendix C its internals.

1.8 Outline

Figure 1.3 shows how the chapters of the thesis connect. Chapter 2 surveys the background and isolates the embedding-table compression problem as the gap this work addresses. Chapter 3 sets out the methodology, the system design and the two experimental protocols. Chapter 4 describes the implementation: the datasets, the software, the export pipeline, the measurement environment, the audit of the June results and the host application. Chapter 5 presents the results, the trade-off analysis and the answers to the research questions. Chapter 6 summarises the work and outlines future work. Four appendices give the full configuration, the complete per-variant and per-seed results, the FEEDWELL-EDGE internals and the reproducibility manifest.

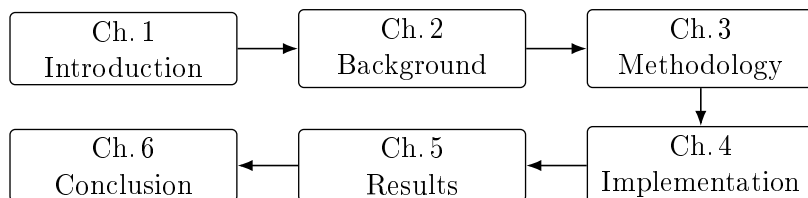


Figure 1.3: Roadmap of the thesis.

Chapter 2

Background

This chapter reviews the work on which the thesis builds. It covers neural news recommendation and its evaluation, multilingual ranking, the constraints and runtimes of on-device inference, quantisation and binarisation, and three ideas that make these compression techniques applicable to text: hardware-aware architecture search, distillation and byte-level encoding. A separate section then reviews the compression of embedding tables, the main obstacle addressed in this work, and the last section states the research gap.

2.1 News Recommendation

News recommendation ranks candidate articles for a reader given the articles that reader has previously opened. It differs from generic recommendation in the volatility of the catalogue: news items appear and expire within hours, so a model cannot rely on a static item-embedding table learned once over a fixed inventory, and it must instead read the content of an article, most economically its title, to decide whether it fits a reader’s interests. This makes the text encoder, and the vocabulary it depends on, the main design choice.

The MIND benchmark [3] fixed the standard setting for the neural form of the task: a large corpus of English news with impression logs, each recording the candidates shown in a browsing session and which of them were clicked. Training follows a negative-sampling convention, in which each clicked candidate is paired with a small number of shown-but-not-clicked candidates and the model learns to score the clicked one above the others. Evaluation is impression-based and threshold-free, and four metrics are reported together because each captures a different aspect of ranking quality.

Ranking metrics

For an impression with candidate scores s_i and binary click labels, with P the set of clicked candidates,

$$\begin{aligned} \text{AUC} &= \Pr[s_+ > s_-], & \text{MRR} &= \frac{1}{|P|} \sum_{p \in P} \frac{1}{\text{rank}(p)}, \\ \text{DCG}@k &= \sum_{i=1}^k \frac{2^{\text{rel}_i} - 1}{\log_2(i + 1)}, & \text{nDCG}@k &= \frac{\text{DCG}@k}{\text{IDCG}@k}. \end{aligned}$$

The area under the receiver-operating-characteristic curve (AUC) is the probabil-

ity that a clicked candidate outranks a non-clicked one, with ties counted as one half; the mean reciprocal rank (MRR) rewards placing the first clicked item early; the normalised discounted cumulative gain (nDCG) at ranks five and ten credits correctly ordered items near the top. All are computed per impression and averaged, so sessions with many candidates do not dominate, and all four are used throughout the thesis.

The dominant architecture factors the task into a news encoder and a user encoder (Figure 2.1). The news encoder maps an article to a fixed-length vector; the user encoder aggregates the vectors of the reader’s clicked history into a single user vector; and a candidate is scored by the inner product of the two. The factorisation is efficient at serving time, since news vectors can be computed once per article and a user is represented by one vector, and the later models in this line all keep it.

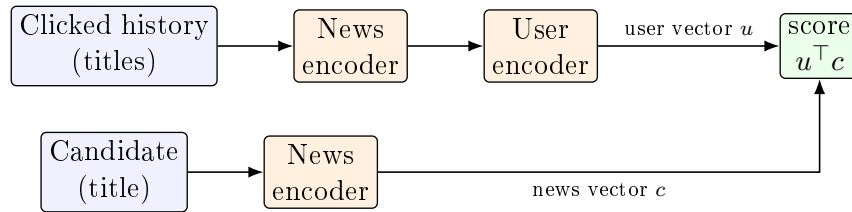


Figure 2.1: The news-encoder/user-encoder factorisation shared by the MIND lineage.

A sequence of models refined the encoders while keeping this shape (Figure 2.2). DKN [4] injected knowledge-graph entities into the news representation; NAML [5] pooled several article fields (title, abstract, category) through attention; NPA [6] conditioned the attention on the user; and LSTUR [7] separated long-term from short-term interests in the user encoder. NRMS [8] reached a strong, widely reproduced operating point with multi-head self-attention over the title words, additive attention to pool them, and self-attention again over the history. On the test split of the large MIND configuration the published AUC of this family runs from about 0.65 for DKN to 0.68 for NRMS [3]; recommenders built on pre-trained language models reach 0.70 to 0.73 [9, 10], but wrap a BERT-sized encoder of hundreds of megabytes. Table 2.1 summarises the line.

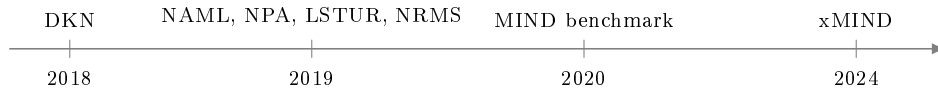


Figure 2.2: A compressed timeline of the models and benchmarks this thesis builds on.

NRMS is the reference baseline in this thesis because it is strong and simple. It is trained here in two configurations: the reference one distributed with MIND, with GloVe [11] vectors, and a reduced one with a randomly initialised table. As Chapter 1 explains, it does not suit the device because its news encoder begins with a word-

Table 2.1: Representative neural news-recommendation models on the MIND lineage.

Model	News-encoder idea	User encoder	Ref.
DKN	knowledge-graph-aware representation	attention	[4]
NAML	attentive multi-view pooling of fields	attention	[5]
NPA	personalised word and news attention	personalised attention	[6]
LSTUR	CNN title encoder with attention	long- and short-term	[7]
NRMS	multi-head self-attention over words	self-attention	[8]

embedding table whose size is set by the vocabulary. Efforts to make language-model recommenders efficient, such as Tiny-NewsRec [12], shrink the encoder but keep a subword vocabulary and target server-scale budgets, so the table remains. A recent survey of on-device recommender systems [13] reviews compression, on-device training and privacy for recommenders in general; this thesis concentrates on the news case, in which the content encoder and its vocabulary dominate the memory cost.

2.2 Multilingual Recommendation

Most of the recommendation literature is monolingual and in English. Whether one model can serve readers across languages was hard to study on benchmarks because no parallel resource existed: comparing a model on English and on, say, Thai meant comparing two different sets of articles, so language and content were confounded. xMIND [14] removes this confound by machine-translating the MIND titles into fourteen typologically diverse languages with the No Language Left Behind (NLLB) model [15]. A single model can then be evaluated on translations of the same content, and its transfer across languages measured directly. The languages cover several families and writing systems (Table 2.2), so the benchmark tests transfer across language families and not only within one.

Table 2.2: The fourteen xMIND languages by family and script.

Code	Family	Script	Code	Family	Script
zho	Sino-Tibetan	Han	som	Afro-Asiatic	Latin
fin	Uralic	Latin	swh	Niger-Congo	Latin
grn	Tupian	Latin	tam	Dravidian	Tamil
hat	French-based creole	Latin	tha	Kra-Dai	Thai
ind	Austronesian	Latin	tur	Turkic	Latin
jpn	Japonic	Kana and Kanji	vie	Austroasiatic	Latin
kat	Kartvelian	Georgian	ron	Indo-European	Latin

xMIND provides translated titles keyed to the same article identifiers but no translated behaviour logs. A cross-lingual evaluation therefore reuses the English impressions and replaces the text, as this thesis does (Chapter 3). The resulting numbers measure how well a content encoder transfers across languages; they do not describe

how native readers of those languages behave. This is the setting a device-resident multilingual recommender needs, and it makes the central claim of this thesis testable. Large translation models such as NLLB [15] reach more languages by adding capacity. A small device cannot do the same, so it needs a representation whose size does not grow with the number of languages.

2.3 On-Device Inference

Deploying a neural model on a microcontroller-class device imposes TinyML constraints of a different kind from those of server or smartphone deployment. Figure 2.3 shows the memory hierarchy: a representative Cortex-M7 offers on the order of two megabytes of flash for parameters and program and one megabyte of SRAM for activations, with no operating system, no reliable dynamic allocation, little floating-point throughput and an energy budget of milliwatts. Three constraints apply at the same time: the parameters must fit in flash, the peak activation memory of any single layer must fit in SRAM, and each inference must complete within a latency and energy budget.

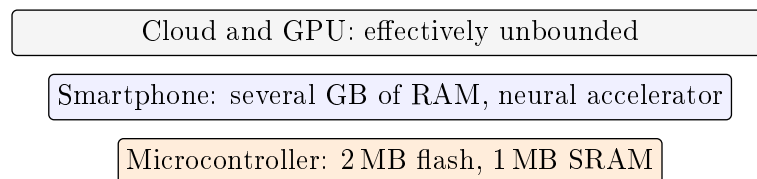


Figure 2.3: The memory hierarchy across deployment tiers; capacity narrows by orders of magnitude toward the microcontroller.

Work on TinyML addresses these constraints. MCUNet [16] brought image classification within a microcontroller budget by co-designing the network and its inference engine with a memory-conditioned search; μ NAS [17] treated flash, SRAM and latency as explicit search bounds; and patch-based inference later relaxed the peak activation bottleneck [18]. All of these lean on efficient convolutional backbones, from the depthwise-separable blocks of MobileNets [19] and the inverted residuals of MobileNetV2 [20] to the compound-scaled EfficientNet family [21]. Execution depends on runtimes built for the target, and the choice of runtime determines what can be deployed (Table 2.3). In this thesis the model is exported in a portable format, and conversion to a microcontroller runtime is the only step between the exported encoder and an on-device measurement; Chapter 4 describes the conversion and Section 5.8 the measurement. For one-bit models the laboratory has two options: the Larq Compute Engine on TensorFlow Lite for Microcontrollers, used in the laboratory’s work on binary networks for the WakeVision dataset [22], and CBin-NN [23], the laboratory’s own C inference engine for binarised networks.

Table 2.3: Inference runtimes across the deployment tiers.

Runtime	Tier, precision	Note
ONNX Runtime	laptop and phone; FP32, INT8	portable export target used here for the laptop timings
TensorFlow Lite for Microcontrollers [24]	microcontroller; INT8	no operating system, no dynamic allocation
CMSIS-NN [25]	Cortex-M; INT8, fixed point	SIMD-optimised kernels used underneath TFLM
ST Edge AI (X-CUBE-AI) [26]	STM32; INT8	vendor path from ONNX or TFLite, cloud benchmark on real boards
Larq Compute Engine [27]	edge, microcontroller; binary	XNOR and popcount kernels for Larq models
CBin-NN [23]	microcontroller; binary	generated C for binarised networks, no framework dependency

2.4 Quantisation

Integer quantisation, the first of the two model-compression techniques studied here, reduces the number of bits used to represent a network’s weights, and in some schemes its activations, by replacing floating-point values with low-precision integers. The standard affine scheme is defined as follows.

Affine quantisation

A weight w is represented as $\hat{w} = s \cdot \text{round}(w/s)$ with a scale s chosen per tensor or, more accurately, per output channel; the integer code is $q = \text{clip}(\text{round}(w/s), q_{\min}, q_{\max})$. For symmetric 8-bit weights, $q_{\max} = 127$ and $s = \max |w|/q_{\max}$.

Eight-bit weights take a quarter of the storage of 32-bit floating point, map onto the integer arithmetic that microcontrollers execute efficiently and, applied carefully, cost little accuracy. The two main approaches differ in when quantisation is applied (Table 2.4). Post-training quantisation (PTQ) calibrates scales on an already-trained network without further training; it is fast but loses more accuracy as the bit-width falls, because the network has not adapted to the rounding. Quantisation-aware training (QAT) simulates quantisation during training so that the network can adapt: a fake-quantisation operation rounds in the forward pass, while the straight-through estimator (STE) passes gradients through the non-differentiable rounding unchanged (Figure 2.4).

This work draws on the literature on low-bit-width training, including DoReFa-Net [28], learned activation clipping [29] and learned step sizes [30], within the broader work on pruning, trained quantisation and coding [31]. The eight-bit models of this thesis use quantisation-aware fine-tuning with per-output-channel scales, applied to the weights of the inner convolutions only; activations, the byte embedding and the

outer layers stay in floating point. A deployable eight-bit file also quantises the inputs of the quantised layers, so the eight-bit rows in Chapter 5 are read with this caveat.

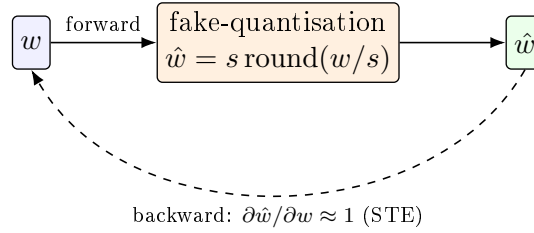


Figure 2.4: Quantisation-aware training: rounding in the forward pass, identity gradient (straight-through estimator) in the backward pass.

Table 2.4: Post-training versus quantisation-aware quantisation.

Aspect	Post-training (PTQ)	Quantisation-aware (QAT, used here)
Training cost	none (calibration only)	fine-tuning with fake-quantisation
Low-bit accuracy	degrades sooner	better retained
Mechanism	post-hoc scale fitting	rounding and STE in the training loop

Quantisation and the binarisation that follows are two points on one precision axis (32, 8 and 1 bits) and are treated as such throughout.

2.5 Binarisation

Binarisation, the second compression technique and the basis of binarised neural networks, is the extreme case of quantisation: it restricts the weights, and in its full form also the activations, to a single bit. A binary weight occupies one thirty-second of a floating-point weight, and when the activations are binary as well, multiply-accumulate operations become a bitwise exclusive-NOR (XNOR) followed by a population count, which costs much less energy and silicon area. Training with binary weights [32], and later with binary activations as well [33], came before the refinements that made binarisation competitive. XNOR-Net [34] established the standard construction: weights are binarised by their sign and rescaled by the mean magnitude of each output channel, which preserves a layer’s output scale while reducing its weights to one bit each.

Binary weights (XNOR-Net)

For the n weights of an output channel, $\hat{w} = \text{sign}(w) \cdot \frac{1}{n} \sum_k |w_k|$; the scalar $\frac{1}{n} \sum_k |w_k|$ restores the magnitude the sign discards. Gradients flow through the sign by a straight-through estimator.

Naive sign binarisation degrades accuracy severely; on the ranking task of this thesis it costs the constrained encoder 0.065 AUC (Chapter 5). Two developments recover much of the loss (Table 2.5). Bi-Real Net [35] adds a real-valued shortcut around each

binary convolution, so that the full-precision activation bypasses the one-bit bottleneck and fine-grained information is preserved (Figure 2.5). ReActNet [36] generalises the binary activation with a learnable-threshold sign (RSign) and a parametric activation (RReLU), which let the network reshape its activation distributions for binarisation. NAS-BNN [37] searches architectures specifically for binary networks, and the ELIOS Laboratory has contributed to this line with a search for binary networks on the Wake-Vision dataset [22] and the thesis that preceded it [38]. The recovered binary encoder in this thesis combines sign-with-scale weights, binarised block inputs, real-valued shortcuts and ReActNet activations with a distilled initialisation.

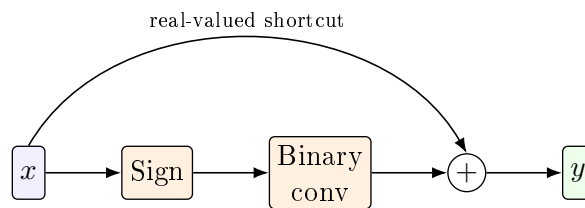


Figure 2.5: The Bi-Real shortcut: a full-precision path added around the binary convolution.

Table 2.5: Binarisation techniques and what each contributes.

Method	Contribution
XNOR-Net [34]	sign weights with per-channel magnitude scaling
Bi-Real Net [35]	real-valued shortcut around each binary block
ReActNet [36]	learnable-threshold sign, parametric activation
NAS-BNN [37]	architecture search targeted at binary networks

2.6 Architecture Search

Neural architecture search (NAS) automates network design by searching a defined space against an objective. In its accuracy-only form it is expensive and ignores deployment cost. Hardware-aware search conditions the objective on a resource model, so the returned architecture must satisfy, or is rewarded for satisfying, a latency, memory or energy budget alongside performing the task. Search strategies differ in how they explore (Table 2.6): reinforcement-learning controllers sample and reward architectures; gradient-based methods relax the discrete space into a differentiable one [39]; and evolutionary methods maintain a population and recombine good candidates. MCUNet’s TinyNAS [16] and μ NAS [17] are the microcontroller-oriented instances, folding flash and SRAM limits into the search so that infeasible architectures are excluded rather than penalised after the fact; on larger budgets, hardware-targeted differentiable search [40] and train-once, specialise-many approaches [41] pursue the same deployment-aware goal. The ELIOS Laboratory maintains a version of μ NAS

for one-dimensional convolutional networks [42], which this thesis uses for its searches under the budgets of two specific boards.

Table 2.6: Families of architecture-search strategy.

Strategy	Mechanism
Reinforcement learning	a controller samples architectures and is rewarded by the validation score
Gradient-based	continuous relaxation of the discrete choice, trained by descent
Evolutionary (used here)	a population evolved by mutation and recombination under constraints

The strategy adopted here is an evolutionary search in the style used for binary architecture search [37]: a population of candidates over channel width, depth and output dimension is evolved across generations, each candidate’s fitness is estimated from a short distillation run at the target precision, and hard constraints enforce the budget (Figure 2.6). Comparing a constrained search with an unconstrained one measures the cost of fitting an on-device budget. A second search, with μ NAS, explores a much wider space of convolutional blocks under the budgets of the two target boards (Section 3.7).

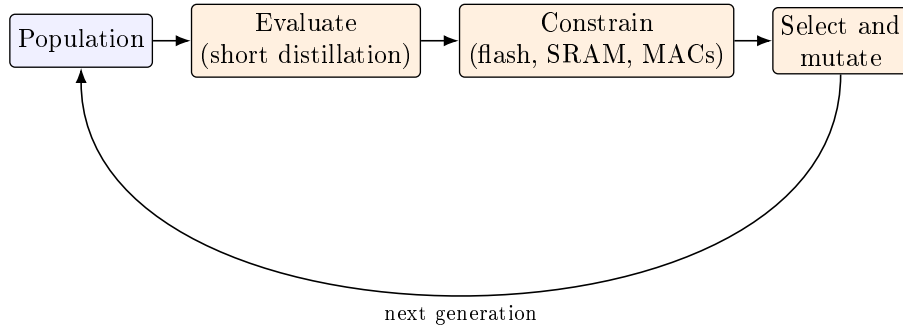


Figure 2.6: The evolutionary search loop with hard resource constraints.

2.7 Distillation and Bytes

The encoder in this thesis combines two lines of work that are usually pursued separately: knowledge distillation and byte-level representation. Distillation, in its original form, trains a compact student to reproduce the soft outputs of a larger teacher [43]. The transfer can target the teacher’s outputs, its intermediate features or the relations between its representations; the variant used here aligns the student’s output embedding with the teacher’s.

Sentence-BERT [44] derived fixed-length sentence embeddings from a transformer [45, 46]. MiniLM [47] distilled such a model into a much smaller one through self-attention transfer, as did the task-agnostic DistilBERT [48], and the multilingual

extension of Reimers and Gurevych [49] trained a student to place a sentence and its translation at the same point using parallel data (Figure 2.7). The teacher used here belongs to this family. Matryoshka representation learning [50] trains an embedding so that its leading prefixes are usable representations of decreasing dimension; the distillation loss in this thesis adopts this nested structure so that a truncated, low-dimensional embedding remains usable on the device.

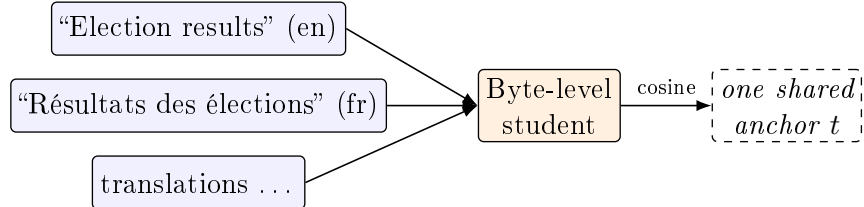


Figure 2.7: Multilingual distillation: translations of one article are pulled toward a single teacher anchor.

The second line of work is vocabulary-free, or token-free, encoding, with character-level convolutional text models [51] and byte-level multilingual processing [52] as early examples. CANINE [53] operates directly on Unicode characters, ByT5 [54] processes raw UTF-8 bytes end to end, and MambaByte [55] applies a selective state-space model to byte sequences; language-agnostic sentence embeddings such as LaBSE [56] are a multilingual counterpart to the teacher used here. These models show that competitive language representations can be learned without a word or subword table. Closer to the budget of this thesis, character-level inputs have been used for click-through prediction in sponsored search [57], and vocabulary-free projection networks give on-device text classifiers of about 200 KB [58]. The classical efficient baseline fastText [59], which represents text with hashed character n -grams, is also related. This thesis uses token-free encoding for a different purpose, to remove the memory wall: reading titles as UTF-8 bytes fixes the alphabet at 256 symbols shared by every language, so the input table does not grow with the number of languages (Figure 2.8). A search of arXiv and IEEE Xplore made in October 2026 found no news recommender that reads raw bytes and none evaluated on a microcontroller; the closest work on on-device search and binary networks concerns vision [16, 37, 22].

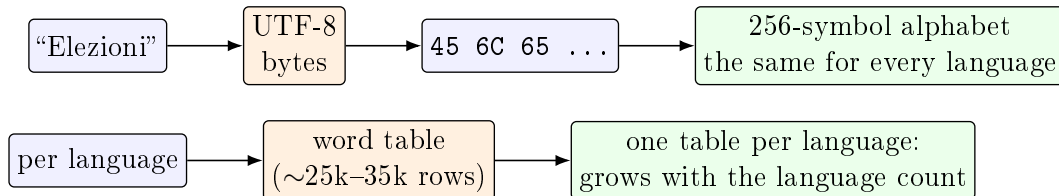


Figure 2.8: Reading a title as UTF-8 bytes (top) replaces a per-language word table (bottom; 25,355 and 34,304 rows in the two NRMS tables of this thesis) with a single 256-symbol alphabet shared across all languages.

2.8 Table Compression

The compression of embedding tables has a literature of its own, but none of its approaches solves the multilingual on-device case by itself. The problem arose in large-scale recommendation before it arose in natural-language processing: categorical features with millions of values produce embedding tables much larger than the networks that use them. Four families of techniques address it (Figure 2.9, Table 2.7).

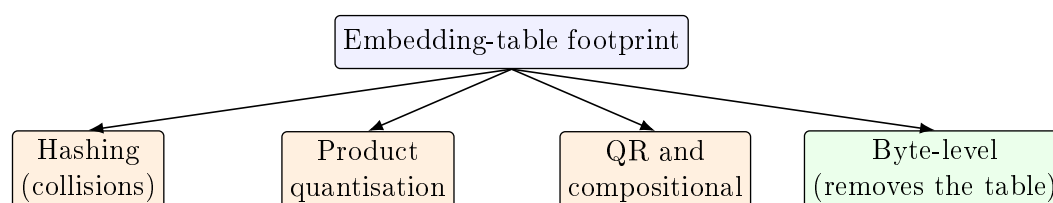


Figure 2.9: A taxonomy of responses to the embedding-table footprint. The first three shrink a per-language table; byte-level input removes it.

The first family is feature hashing. The hashing trick [60] maps a large index space into a small fixed number of buckets through a hash function, trading a controlled rate of collisions for a table whose size no longer tracks the vocabulary; HashedNets [61] applied the same idea inside a network, forcing groups of weights to share values selected by a hash. Hashing bounds the table but accepts collisions: distinct words share a row, and the representation must tolerate the resulting interference.

The second family is product quantisation and its relatives [62], which compress a table after the fact by splitting each vector into sub-vectors and replacing each with the index of a nearby centroid from a small learned codebook. The table becomes a set of short codes plus a codebook, which can be much smaller, at the cost of a reconstruction step and some fidelity. This compresses an existing table well but presupposes that a table exists and was trained at full size.

The third family, closest to the present work, is compositional or quotient-remainder (QR) embeddings [63]. Instead of one row per value, a vector is composed from two or more smaller tables indexed by complementary functions of the identifier, for instance its quotient and remainder under some modulus, so that a vocabulary of size V is covered by tables of total size proportional to $\sqrt{V}$ instead of V (Figure 2.10). The compression is substantial and collision-free, and the tables remain differentiable and trainable end to end. QR does not, however, change the unit of representation, which is still a word in a specific language: a compositional table is still a table of words, and covering another language still requires another set of tables.

Byte-level representation is therefore a different kind of approach from table compression. Hashing, product quantisation and compositional embeddings shrink a per-language vocabulary table, but none of them removes the table or its dependence on the number of languages. Reading the input as UTF-8 bytes replaces the word as the unit of representation: n language-specific tables of tens of thousands of rows become one shared table of 257 rows (256 byte values and padding). The two approaches can

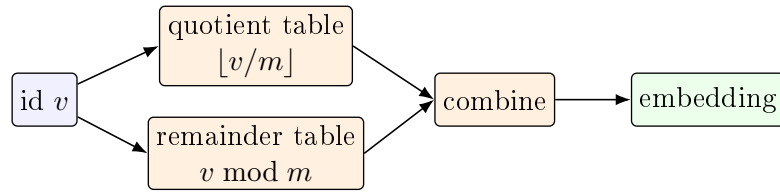


Figure 2.10: Quotient-remainder embeddings: two small tables of total size proportional to $\sqrt{V}$ replace one of size V , but the unit is still a per-language word.

Table 2.7: Embedding-table responses compared. Only byte-level input makes the representation independent of the language count.

Approach	Removes the table?	Scales with languages?	Cost
Feature hashing	no (shrinks it)	yes	collisions
Product quantisation	no (shrinks it)	yes	reconstruction, fidelity
QR and compositional	no (shrinks it)	yes	combine step
Byte-level (this work)	yes	no	longer sequences

be combined, since the small embedding of a byte-level model could itself be hashed or compressed, but only the change of representation makes the table independent of the number of languages, which is what a device serving many languages needs. The cost is sequence length: an English title of a dozen words is about sixty bytes, so the encoder trades flash for computation, as Chapter 5 quantifies.

Bytes versus compression

Hashing, product quantisation and QR embeddings make a per-language table smaller; byte-level input makes it unnecessary. With one alphabet of 256 symbols for all languages, the representation no longer grows with the number of languages, and the remaining cost lies in the network’s arithmetic, which the compression techniques developed for TinyML address.

2.9 Research Gap

These lines of work have not been combined before. News recommendation has been studied at scale, but with word-vocabulary encoders that do not fit on a microcontroller. TinyML techniques have been developed mainly for vision, where compressing the arithmetic compresses the model; for text, the vocabulary table, which these techniques do not target, has prevented the same transfer. The embedding-compression literature shrinks that table but keeps it language-specific. Distillation and token-free encoding have been developed for representation quality and robustness to noise, without a microcontroller budget in mind. Figure 2.11 places this work in the quadrant that earlier work leaves empty: on-device multilingual text, where most of the cost lies in a table and not in arithmetic.

This thesis combines these lines of work in a fixed order. The token-free represen-

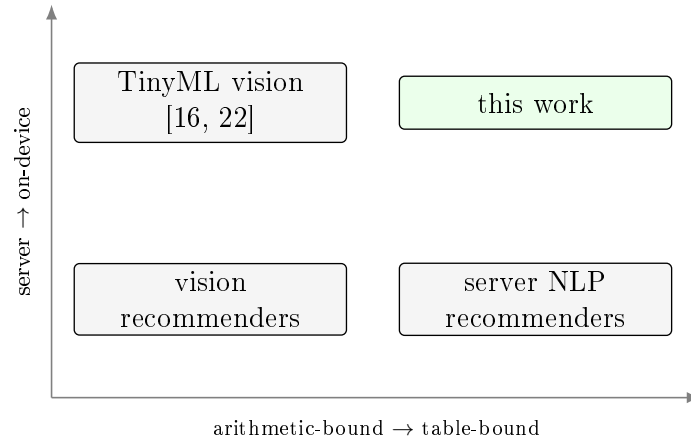


Figure 2.11: Positioning: the on-device, table-bound quadrant, multilingual text on a microcontroller, is the one this thesis addresses.

tation comes first: it removes the vocabulary table, whose cost grows with the number of languages, and replaces n tables with one 256-symbol alphabet. Multilingual distillation then transfers a strong teacher’s cross-lingual representation into the byte-level student. Only after that are hardware-aware search, integer quantisation and binarisation applied to the remaining arithmetic. The contribution lies in showing that, after the vocabulary table has been removed, existing techniques transfer to on-device multilingual news recommendation, and in measuring how far they transfer, including the point at which one-bit precision fails and what is needed to recover from it. In the laboratory’s work on vision models [38, 22], model size follows the arithmetic, so the search is well posed from the start. In a word-based text model the two are largely independent, and the input representation has to change before the search becomes meaningful. Chapter 3 describes how this is done.

Chapter 3

Methodology

3.1 System Overview

The system keeps the news-encoder/user-encoder factorisation established on MIND but replaces the word-vocabulary news encoder with a byte-level convolutional encoder, produced by a pipeline designed for a microcontroller-class budget. Figure 3.1 shows the whole system, and Table 3.1 lists its stages with their inputs and outputs. A frozen multilingual teacher produces one anchor vector for every article from its English title. A byte-level student is distilled to reproduce these anchors from raw UTF-8 bytes, so that translations of the same article are drawn toward one representation without a per-language vocabulary. An architecture search over the student’s topology runs in three arms. For the architecture selected in each arm, a recommender is trained on English clicks and evaluated with its inner weights at full, eight-bit and one-bit precision. Finally, an export stage writes the trained encoder and a cold-start prior in the schema of the host application. A second experimental programme, run in October 2026 with three seeds per condition, reruns the nine variants, compares the reference encoder with the baselines and scorers of Section 3.11, and searches again with μ NAS under the budgets of the two target boards (Section 3.7).

Table 3.1: The methodology as a multi-stage pipeline of inputs and outputs.

#	Stage	Input and operation	Output
1	Anchor	English titles through the frozen teacher, L2-normalised	one 384-d anchor per article
2	Distil	byte-level student fitted to the anchors with the Matryoshka cosine loss	distilled encoder
3	Search	encoder space (120 architectures) under an evolutionary search per arm	three winning architectures
4	Train	winner plus click impressions, ranking loss with sampled negatives; precision sweep at 32, 8 and 1 bits	nine variants
5	Export	encoder to ONNX plus a JSON cold-start prior in the application’s schema	deployable artifact
6	Reference	seeded reruns of the 64-5-384 encoder, NRMS in two configurations, teacher-based scorers and label histograms	reference points, three seeds
7	Board search	μ NAS under the budget of each board; finalists ranked again after full distillation	one encoder per board

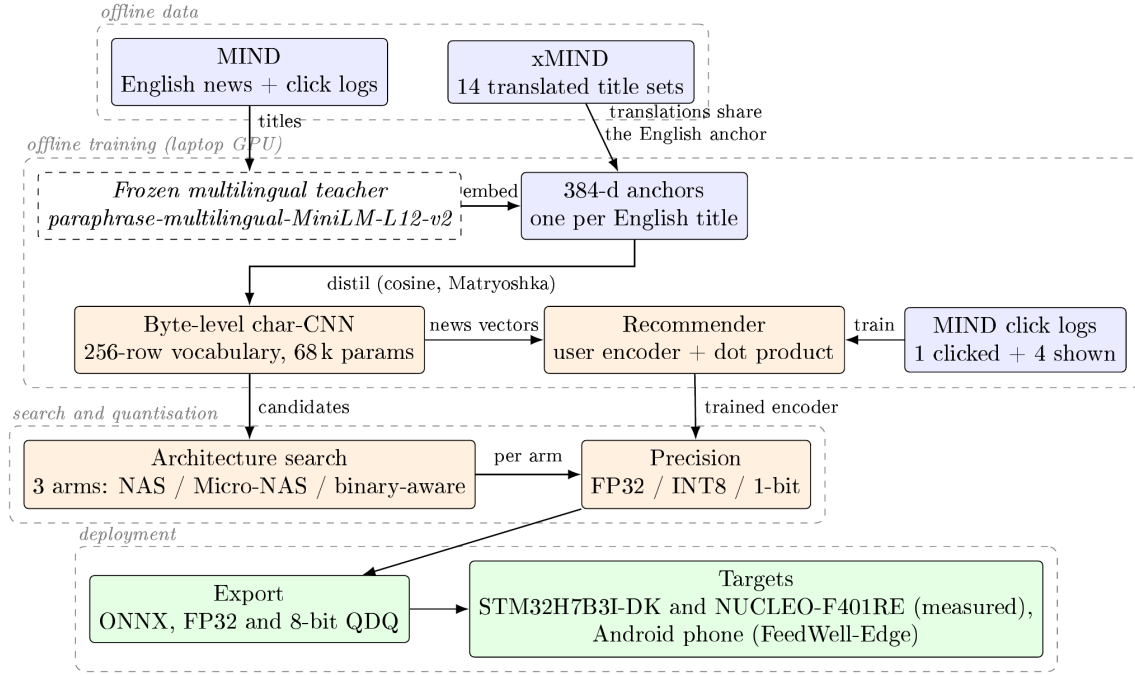


Figure 3.1: System overview. Offline data (top) feed the offline training stages on the laptop GPU; the searched and quantised encoders are exported for the targets (bottom). The teacher is used only offline and is never shipped to a device.

The recommender itself is conventional, so that the effect of the encoder and its precision is not mixed with that of a new ranking mechanism. The news encoder maps a title to a vector, the user encoder pools the history vectors through additive attention, and a candidate is scored by the inner product of its vector with the user vector (Section 3.5).

3.2 Hardware Budgets

The system is designed for three hardware tiers with different roles, and stating their budgets at the outset makes the size requirement measurable (Table 3.2). The laptop is used for training, search and evaluation and imposes no deployment constraint. The Android phone runs the host application; it could hold any of the models in this thesis and serves as the integration target. The STM32H7, a Cortex-M7 microcontroller with about two megabytes of flash and one megabyte of SRAM, sets the constraint that shapes the search. A smaller board, the NUCLEO-F401RE with 512 KB of flash and 96 KB of SRAM, is the second target of the laboratory’s board-benchmark procedure and is included because the constrained encoder also fits it.

The constrained search arms use limits tighter than those of the STM32H7: 256 KB of flash, 128 KB of working memory and two million multiply-accumulates per title. With the full two megabytes, a small text encoder would fit easily and the constrained and unconstrained searches would return similar architectures, which would remove the comparison on which this thesis relies. Section 3.6 records which of the three limits

Table 3.2: Target-hardware tiers and their roles.

Tier	Device	Budget	Role in this thesis
Host	laptop, RTX 5070 GPU	none	train, search, evaluate; laptop timings
Phone	Android phone with FEEDWELL-EDGE	gigabytes	host of the application; integration target
Microcontroller	STM32H7B3I-DK (Cortex-M7)	~2 MB flash, ~1 MB SRAM	binding budget that drives the search constraints
Microcontroller	NUCLEO-F401RE (Cortex-M4)	512 KB flash, 96 KB SRAM	second measurement board

the June search actually enforced. The μ NAS searches of Section 3.7 use one budget per board instead: 256 KB of weights, 128 KB of activations and 2 million multiply-accumulates per title for the STM32H7B3I-DK, and 128 KB, 48 KB and 1 million for the NUCLEO-F401RE.

3.3 Byte-Level Encoder

3.3.1 Input

The content encoder reads a title as a sequence of raw UTF-8 bytes instead of words. Each byte value maps to an identifier in $[1, 256]$, with 0 reserved for padding, and the sequence is truncated or padded to 128 bytes. The whole “vocabulary” is therefore the 256 byte values, the same for every language that UTF-8 can encode, so no table grows when a language is added and none is language-specific. The cost is sequence length: an English title of a dozen words is about sixty bytes, and titles in scripts whose characters take three bytes each, such as Thai, Georgian or Tamil, fill the 128-byte window with fewer characters. The same truncation rule applies to every language.

3.3.2 Architecture

The student is a byte-level convolutional network (Figure 3.2). A byte-embedding layer maps the 257 identifiers to a 64-dimensional vector; a pointwise (1×1) convolution projects to the working channel width C ; a stack of D depthwise-separable blocks [19], each a depthwise convolution of kernel size three followed by a pointwise convolution, batch normalisation and a ReLU, mixes information along the sequence and across channels at low parameter cost (Figure 3.3); a masked mean over the non-padding positions pools the sequence; and a linear head projects the pooled vector to the output dimension, 384 for every searched winner. The output is nested in the sense of Matryoshka representation learning [50]: the leading 64 dimensions are trained to be a usable embedding on their own, which is why the distillation loss of Section 3.4 is defined over nested prefixes.

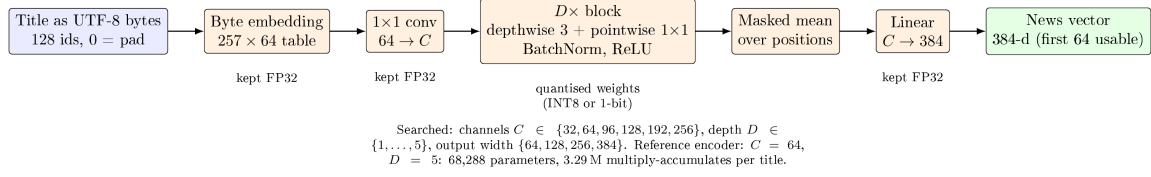


Figure 3.2: The byte-level encoder. The embedding, the first projection and the head stay in full precision at every setting; only the weights of the inner blocks are quantised. The reference encoder ($C = 64$, $D = 5$) has 68,288 parameters.

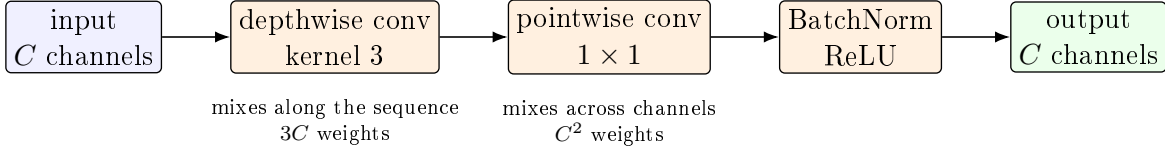


Figure 3.3: The depthwise-separable block: a depthwise convolution mixes information along the sequence and a pointwise convolution mixes it across channels, with $3C + C^2$ weights instead of the $3C^2$ of a full convolution of the same kernel.

The parameter count follows from the three hyperparameters. With a byte embedding of 64, the embedding has 257×64 entries, the first projection $64C + C$, each block $3C + C + C^2 + C + 2C$ (the depthwise and pointwise weights, their biases and the batch-norm scale and shift), and the head $384C + 384$. Table 3.3 applies the count to the three architectures the search returned, at full precision and with the block weights at eight bits; the same formula produces every size in Chapter 5.

Table 3.3: Stored size per layer group of the three searched architectures, in KB, at full precision and with the inner block weights at eight bits. Biases and batch-norm parameters stay at 32 bits. The embedding, first projection and head are the same at both precisions.

Architecture (C - D -out)	Embedding	First projection	Blocks FP32	Blocks INT8	Head	Total FP32 / INT8
256-4-384 (NAS)	64.3	65.0	1052.0	275.0	385.5	1566.8 / 789.8
64-5-384 (Micro-NAS)	64.3	16.3	88.8	25.9	97.5	266.8 / 203.9
96-2-384 (binary-aware)	64.3	24.4	77.3	21.6	145.5	311.4 / 255.7

The count has two consequences for the results. In the constrained architecture the quantisable weights are 31% of the parameters, so eight-bit weights reduce its size by about a quarter instead of three quarters, and one-bit weights save only another 18 KB. In the unconstrained architecture they are 66%, and eight-bit quantisation halves the model. The byte embedding, at 64 KB, is the largest fixed cost of the small encoders; it is the only table the design keeps, and all languages share it.

3.4 Distillation

The teacher is the frozen multilingual sentence-transformer **paraphrase-multilingual-MiniLM-L12-v2** of the Sentence-Transformers family [47, 49], which produces 384-dimensional embeddings. It runs offline only and is never deployed to the device. For

each article the teacher encodes the English title, and the L2-normalised result is stored as that article’s anchor. Because xMIND provides parallel translations keyed to the same article identifiers, the translation of an article into any supported language is trained against the same English anchor, which teaches the student to map a title and its translations to one target (Figure 2.7).

Matryoshka distillation loss

Let $f_\theta(x)$ be the L2-normalised student output for a byte sequence x and t the normalised anchor of its article. Over the nested prefix dimensions $\mathcal{D} = \{64, 128, 256, 384\}$,

$$\mathcal{L}_{\text{distil}}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \left(1 - \cos(f_\theta(x)_{1:d}, t_{1:d}) \right),$$

so each prefix is independently encouraged to align with the corresponding prefix of the anchor.

Distillation is used in three places. It is the fitness function of the architecture search, where a candidate is scored by a short distillation. It initialises the encoder in the ablations of Chapter 5 and in the recovered one-bit model. Finally, the distilled student, kept frozen, serves as a news encoder under a trained user encoder, which tests how much of the teacher’s cross-lingual alignment the 68-thousand-parameter student carries. The nine models of the precision matrix are trained on clicks from a random initialisation, so the matrix measures the architecture and precision axes without the effect of distillation.

3.5 Recommender

The recommender is the NRMS scoring rule with the byte-level encoder in place of the word-level one (Figure 3.4). History titles and candidate titles pass through the same encoder; additive attention pools the history into a user vector; the score is a dot product.

User vector and candidate score

For a user with history news vectors $\{h_1, \dots, h_H\}$ and a candidate news vector c ,

$$u = \sum_{i=1}^H \alpha_i h_i, \quad s = u^\top c,$$

$$\alpha_i = \frac{\exp(q^\top \tanh(W h_i))}{\sum_{j=1}^H \exp(q^\top \tanh(W h_j))},$$

with padded history positions masked out of the softmax. Only the news encoder that produces h_i and c differs between the baseline and the study models.

Training follows the negative-sampling convention of MIND: each clicked candidate is presented with four shown-but-not-clicked candidates from the same impression, the five scores pass through a softmax with the clicked one first, and the model is trained by cross-entropy with AdamW [64]. Up to fifty clicked titles form the history. Two details of the training loop changed between the two experimental programmes (Section 3.10) and affect how the results are read: whether the negatives are drawn once per impression or redrawn at every step, and whether the network initialisation is seeded.

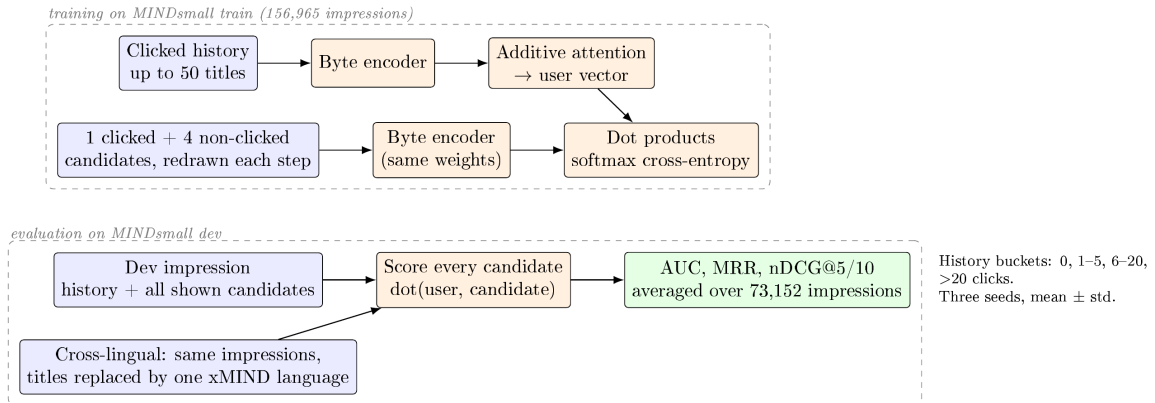


Figure 3.4: Training and evaluation. Top: the ranking loss over one clicked and four non-clicked candidates. Bottom: every candidate of a development impression is scored and the impression-level metrics are averaged over the 73,152 labelled impressions; the cross-lingual evaluation keeps the impressions and replaces the titles with one xMIND language.

3.6 Architecture Search

The architecture is selected by search over the space of Table 3.4: six channel widths, five depths and four output widths, 120 architectures in all. The search is evolutionary, in the style used for binary architecture search [37]. A population of thirty candidates is evolved for fifteen generations, and each candidate’s fitness is the teacher cosine reached by a two-epoch distillation on twenty thousand titles, measured on four thousand held-out titles, at the precision of the arm. The search therefore favours architectures that distil well at the precision they will be deployed with, and each candidate takes minutes to evaluate, whereas a full recommender training takes hours.

The search runs in three arms, which together separate the architecture and precision axes. The unconstrained arm (NAS) shows what the search selects when only quality matters. The constrained arm (Micro-NAS) searches at eight-bit precision under the hard budget and returns the best feasible architecture. The binary-aware arm searches at one-bit precision under the same budget, so that the selected architecture suits binary weights instead of being assumed equal to the full-precision optimum. As in microcontroller-oriented search [16, 17], a candidate that violates a bound is

Table 3.4: Search space and search parameters of the June 2026 run.

Axis	Values
Channels C	{32, 64, 96, 128, 192, 256}
Depth D	{1, 2, 3, 4, 5}
Output dimension	{64, 128, 256, 384}
Population, generations	30, 15
Fitness	teacher cosine after a two-epoch distillation on 20,000 titles, 4,000 for validation
Constraints (constrained arms)	256 KB flash, 128 KB RAM, 2 million multiply-accumulates

removed from the population.

One limitation of the June run affects how the constrained winners should be read. The flash and RAM bounds were enforced by the size and buffer rules of Section 3.9, but the operation count used in the search came from a library counter that skipped the quantised convolution layers, so the bound of 2 million multiply-accumulates was checked against an undercount. With the corrected counter, both constrained winners exceed it (3.29 and 3.26 million). The flash and RAM bounds were effective and are what separated the constrained arms from the unconstrained one; the operation bound was not. The June search was not rerun with the corrected count; the board-specific searches of Section 3.7 enforce the operation bound instead.

3.7 Board-Specific Search

The June search space contains only channel-depth grids of one block type, and its operation count was wrong. In October 2026 the encoder was therefore searched again for the two boards, with the laboratory’s one-dimensional version [42] of μ NAS [17]. The μ NAS space is much wider: up to ten convolution blocks of standard, pointwise or depthwise layers (kernel size 3 to 7, stride 1 or 2, optional max-pooling, batch normalisation and ReLU), optional residual branches, and a global-average-pooling head with a 384-unit output. A candidate reads a title’s bytes through the fixed byte table of the distilled reference encoder plus a padding-mask channel. Its fitness comes from a short distillation to the teacher, at most fifteen epochs on 96,000 rows in fifteen languages, and its error is one minus the mean cosine to the teacher on validation rows. Aging evolution, with 150 candidates, a population of 40 and a sample of 12, trades this error against the multiply-accumulates, the eight-bit weight size and the peak activation memory given by the fork’s resource model, under the budget of one board (Section 3.2).

The eight best candidates within each budget are trained again with three seeds under the search recipe, and the best of them is the recipe choice. Because the short recipe can rank candidates differently from full training, the final choice among the same eight is made after full distillation (fifteen epochs on all 769,000 training rows),

by the mean validation cosine over three seeds; both choices use validation rows only. For comparison, the thirty hand-designed shapes of the June space with a 384-wide output are trained under the same recipe and costed with the same resource model, and the best shape within each budget is kept. For the ranking experiments, the chosen encoders, the two hand-designed comparators and 64-5-384 are rebuilt in PyTorch with the same input and pooling, with outputs within 2.3×10^{-5} of the Keras builds, and trained like the reference encoder: distillation on all training rows, then eight epochs of English clicks, with seeds 42, 12 and 1.

3.8 Precision Scheme

Full precision, eight-bit integer and one-bit binary are three points on one precision axis, applied through a common mechanism: a convolutional layer is replaced by a quantisation-aware variant that substitutes a fake-quantised version of its weights in the forward pass, with gradients passing through the rounding by the straight-through estimator. Following XNOR-Net practice [34], the byte embedding, the first projection and the linear head are kept in full precision at every setting, and so are all activations; only the weights of the inner depthwise and pointwise convolutions are quantised. For eight-bit weights the fake-quantisation uses a symmetric per-output-channel scale, and the models are produced by two epochs of quantisation-aware fine-tuning after the ranking training. For one-bit weights the weight is binarised by its sign and rescaled by the per-channel mean of magnitudes.

Weight quantisation

INT8, per output channel, with $q_{\max} = 127$:

$$\hat{w} = s \cdot \text{clip}(\text{round}(w/s), -q_{\max} - 1, q_{\max}), \quad s = \frac{\max |w|}{q_{\max}}.$$

Binary, with XNOR-Net scaling over the n weights of a channel:

$$\hat{w} = \text{sign}(w) \cdot \frac{1}{n} \sum_k |w_k|.$$

In storage, eight-bit and one-bit weights take a quarter and a thirty-second of the space of 32-bit ones; in resolution, a weight has effectively continuous values at 32 bits, 256 levels at eight bits and two at one bit (Figure 3.5). Because the simulation acts on weights only, the eight-bit and one-bit rows of Chapter 5 give the accuracy of a model whose inner weights have been rounded, which may differ from the accuracy of the deployable eight-bit file. That file also quantises the inputs of every convolution and of the output layer, while activation functions and averages stay in floating point; Section 5.8 reports it for the constrained encoder, evaluated from the exported file and measured on the boards.

Naive sign binarisation of the inner weights is one of the nine matrix cells, and Chap-

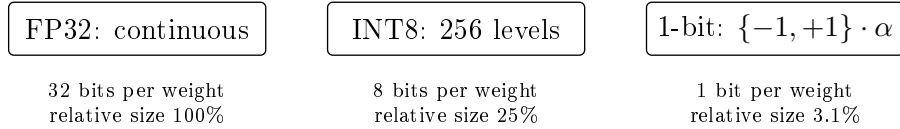


Figure 3.5: The precision axis as resolution and as storage, exact for the quantised layers.

ter 5 shows that it costs the constrained architecture 0.065 AUC. The recovered one-bit model changes the block design as well as the weights. Each depthwise-separable block is replaced by a full binary one-dimensional convolution of kernel three whose inputs are binarised by a learnable-threshold sign, followed by the parametric activation of ReActNet [36] and a real-valued shortcut after Bi-Real Net [35], and the network is initialised from the distilled full-precision encoder instead of from scratch. Together, these three changes form the recovery recipe evaluated in Section 5.4.

3.9 Cost Model

Every estimated size, memory and energy figure in this thesis is computed from the model definition by one per-layer counter. The rules are given here so that each number can be recomputed by hand.

- **Size** is the sum over layers of parameter count times bit-width, with the quantised block weights at 8 or 1 bits and every other parameter at 32 bits. It is the size of the weights that an integer file would store and excludes the overhead of any particular file format.
- **RAM** is the input and output buffer of the widest block, $2 \times 128 \times C$ activations, at four bytes per activation in full precision and one byte otherwise. It is a lower bound on working memory, since it ignores the runtime’s own overhead.
- **Operations** are multiply-accumulates counted per layer by forward hooks on every convolution and linear layer, including the quantised subclasses; the embedding lookup counts none.
- **Energy** multiplies each layer’s operations by a per-operation figure (Table 3.5) and sums the results; it is a relative proxy.

Table 3.5: Per-operation energy used by the proxy. The 32-bit and 8-bit figures are the multiply and add of Horowitz’s 45 nm survey [65]; the one-bit figure follows the convention of the binary-network literature that one 32-bit operation is worth 64 binary ones [34, 35], and is a convention rather than a measurement.

Operation	Source	pJ
32-bit floating-point multiply-accumulate	$3.7 + 0.9$ [65]	4.6
8-bit integer multiply-accumulate	$0.2 + 0.03$ [65]	0.23
1-bit operation	$4.6/64$, operation-count convention	0.0719

The proxy ignores memory traffic, which can dominate on a Cortex-M, and it treats

a binary multiply-accumulate as one sixty-fourth of a floating-point one, although the Cortex-M7 has no population-count instruction and a one-bit kernel would have to count bits in software. The energy column therefore only orders the nine variants; the phone measurement of Section 5.8 is the only measured energy figure in this thesis. Chapter 4 records that the counter behind these rules was corrected in October 2026 and that every cost in Chapter 5 was recomputed with it from the saved architectures.

3.10 Protocols

The numbers in Chapter 5 come from two experimental programmes, which used two versions of the training loop (Table 3.6). The June 2026 programme ran the search, the nine-variant matrix, the NRMS baseline, the distillation ablation and the recovered binary model in one 8.2-hour batch. Its loop drew the four negatives of each impression once, before training, and its network initialisation was not seeded. The October 2026 programme uses a loop that seeds the initialisation and redraws the negatives at every step from a seeded generator, runs every condition three times and reports the mean and standard deviation over seeds. It covers the reference conditions of Section 3.11 (seeds 42, 12 and 1), a rerun of the nine matrix variants on the June architectures and the recovered one-bit model (seeds 42, 1 and 2), and the encoders of the board-specific searches (seeds 42, 12 and 1). The matrix, the one-bit results and the reference points of Chapter 5 are therefore October values; the June programme contributes the three architectures and the distillation ablation, and its single runs are quoted only for comparison, always marked as June runs.

A single configuration file fixes every hyperparameter; it is reproduced in Appendix A together with the overrides applied by the two programmes. Every run from October onwards appends a record with its settings, duration, metrics, random seed and code commit to a machine-readable log, and the downloaded data are pinned by checksum (Appendix D).

Ranking quality is evaluated impression by impression with AUC, MRR and nDCG at ranks five and ten. AUC is the primary metric because it is threshold-free and the most directly comparable figure between the baseline and the compressed models. Ties, which arise for example when a reader has no history and every candidate receives the same score, count as one half, so a scorer that cannot order the candidates of an impression scores exactly 0.5 on it.

Two further views are reported for the reference conditions. First, impressions are split by the reader’s history length into four buckets (no history, 1 to 5, 6 to 20 and more than 20 clicks), which separates what the model knows about an article from what it knows about a reader. Second, the cross-lingual evaluation keeps the English development impressions and replaces every title, in the history and among the candidates, with its translation into one xMIND language. The pass is repeated for each of the fourteen languages, and the per-language AUC and the mean over the fourteen languages are reported.

Table 3.6: The two training protocols. Both train on the 156,965 training impressions and evaluate on the 73,152 labelled development impressions.

	June 2026 (search, matrix, ablation, binary)	October 2026 (reference conditions, matrix and one-bit reruns, board-search encoders)
Ranking epochs	8 (10 in the distillation ablation)	8
Quantisation-aware epochs	2 after the ranking epochs	2 after the ranking epochs (matrix rerun)
Negatives	four per impression, drawn once	four per impression, redrawn every step from a seeded generator
Initialisation	not seeded	seeded; seeds 42, 12, 1 (reference conditions, board-search encoders) or 42, 1, 2 (matrix and one-bit reruns)
Distillation	15 epochs, learning rate 10^{-3}	15 epochs, learning rate 10^{-3}
Ranking optimiser	AdamW, learning rate 10^{-4} , batch 64	AdamW, learning rate 10^{-4} , batch 64
Runs	single	three per condition, mean $\pm$ standard deviation

3.11 Reference Conditions

A compressed model needs reference points on both sides: larger models and simpler scorers. Besides three scorers that use no network and are computed in a single pass, the October programme defines nine trained conditions, each run three times (Table 3.7). The three scorers are the popularity of an article in the training clicks and a per-user histogram of the reader’s clicks over MIND’s seventeen editorial categories or over its 264 subcategories, applied to the candidates’ labels. Two of the trained conditions are NRMS: the reference configuration distributed with MIND, with the GloVe-initialised 300-dimensional table over 34,304 words and twenty attention heads, and the reduced configuration used as the June baseline, with a randomly initialised 256-dimensional table over the 25,355 words of the training titles and eight heads. Two use the frozen teacher as the news encoder, directly or through a trainable linear map, which bounds what any student distilled from it can reach. The remaining five use the 64-5-384 byte-level encoder in full precision under five training conditions, which separate the effect of the distilled initialisation from the effect of the training signal: a random start or the distilled weights, frozen or fine-tuned, with the clicks shown in English or with each training instance shown in a random xMIND language.

Table 3.7: The reference conditions of the October 2026 programme. Each trained condition is run with three seeds; the scorers without a network are computed in one pass.

Condition	What it measures
Popularity	a floor with no personalisation
Category and subcategory histograms	what MIND’s editorial labels alone give a reader
NRMS, reduced	the June baseline under the October loop
NRMS, GloVe reference	the word-vocabulary model at its published configuration
Frozen teacher, with and without a linear map	the ceiling of what the teacher’s representation offers
Byte-CNN, random start, English clicks	the encoder with no distillation (the matrix models are of this kind)
Byte-CNN, random start, mixed-language clicks	whether translated clicks alone teach cross-lingual ranking
Byte-CNN, distilled, frozen	how much of the teacher’s alignment the student carries
Byte-CNN, distilled start, English clicks	the effect of the distilled initialisation
Byte-CNN, distilled start, mixed-language clicks	the combination that serves many languages

Chapter 4

Implementation

This chapter describes how the methodology of Chapter 3 was implemented and measured. It covers the datasets and their preparation, the software, the export and packaging pipeline, the environment in which each quantity was measured, the audit that separates the two experimental programmes, and the host application written during the internship.

4.1 Datasets

Evaluation uses MIND [3] for the English task and xMIND [14] for the cross-lingual evaluation. MIND is distributed in two configurations. The small one is used throughout; this makes the search-and-precision grid affordable but keeps the absolute ranking numbers below what the large configuration would give. This choice affects every number in Chapter 5: the large split would be expected to raise the figures without changing the comparisons. The small split has no public test set, so evaluation uses the development split, with the number of epochs fixed in advance; for reference, the DIGAT authors report 0.656 AUC for NRMS on this split [66]. Table 4.1 gives the main quantities, computed from the downloaded files.

Table 4.1: The data as used. Counts computed from the downloaded files; the last three rows are preparation parameters.

Quantity	Value
Users (train and dev)	94,057
Training impressions	156,965
Dev impressions (labelled, used for evaluation)	73,152
Articles in the training split	51,282
Articles in the dev split	42,416
Categories / subcategories (editorial labels)	17 / 264
xMIND languages (translated titles)	14
Clicked history kept per user	50
Title length (UTF-8 bytes)	128
Negatives per positive (redrawn each step)	4

Two properties of the development split matter for the later analysis. First, only 3.0% of its impressions belong to readers with no clicked history, 14.7% to readers with one to five clicks, 34.8% to readers with six to twenty and 47.5% to readers with more than twenty. The overall AUC is dominated by readers with long histories, so the per-bucket view of Section 5.6 is needed to see the others. Second, there are 73,152

labelled impressions, which sets the evaluation cost, since a full pass scores every shown candidate of every impression.

xMIND supplies the MIND titles translated into fourteen languages and is released under the terms stated by its authors. Here it is used in three ways: in distillation, where the translations of each title are trained toward the English-title anchor; in the cross-lingual evaluation; and, in the mixed-language conditions, to show the English training impressions in translated form. In click training, the other fourteen languages are either not seen at all (English-click conditions) or seen only as translations of the same English impressions (mixed-click conditions). The resulting transfer is a property of the English-anchored representation and of the training signal, not of native-language behaviour data, so Chapter 5 reports each language separately as well as the mean. The 31 downloaded files are pinned by SHA-256 in a manifest committed to the repository (Appendix D).

4.2 Software

The pipeline is implemented in PyTorch with Compute Unified Device Architecture (CUDA) support and organised as a small library of modules with one script per experimental programme (Table 4.2). The byte-level encoder follows Chapter 3; the teacher is loaded from the Sentence-Transformers model hub, and its embeddings are cached once per split and language. Integer variants use quantisation-aware fine-tuning with per-output-channel scales. Binary variants binarise by sign with a per-channel magnitude scale and keep the embedding, first projection and head in full precision; the recovered variant also uses ReActNet activations and Bi-Real shortcuts. Every script that produces a reported number writes its result file into a versioned results folder, and the figures and tables of this thesis are generated by scripts that read only those files.

4.3 Export

The export and packaging pipeline converts a trained encoder into a form that a device can load, and it is where the research pipeline hands over to deployment. For each arm and precision, the encoder is exported to the Open Neural Network Exchange (ONNX) format with a dynamic batch dimension and named by arm and precision (for example `content_encoder_micro_nas_int8.onnx`). Each file comes with a manifest and a JavaScript Object Notation (JSON) cold-start prior: the normalised click frequency of each MIND category in the training logs, written as a `topicWeights` object in the storage schema of the host application, so that a newly installed client has a population-level ranking before any personal interaction has been recorded. The application already uses this schema for its on-device state, so the model and the host share a single interface. The June full run and the quick re-executions of the notebook produced these files.

The ONNX files of the matrix store the simulated weights in floating point, which is not what a microcontroller runs. A second export, added in October 2026 (`scripts/deploy_boards.py`), produces the deployment files. It exports the trained constrained encoder with a fixed batch of one, at full precision and as an eight-bit quantise–dequantise (QDQ) ONNX model made by ONNX Runtime (ORT) static quantisation, calibrated on 512 training titles. In this eight-bit file every convolution and the output layer run with per-channel symmetric eight-bit weights and eight-bit inputs, while activation functions, pooling and averages run in floating point between them, and the byte table stays a 32-bit lookup. The same two files are produced for the other two architectures with random weights, since their latency and footprint do not depend on the weight values. The same export (`scripts/export_int8.py`) was later applied to every seed of the reference encoder and to the encoders of the board-specific searches. A variant that also quantises the remaining operators kept the accuracy but ran no faster on the boards, so the files described here remain the deployment files.

The files contain the network after the byte-table lookup: the caller gathers the 128 rows of the 257×64 table and passes the embedded sequence and the padding mask, which is the usual arrangement for an embedding lookup on a microcontroller. A first variant with the lookup inside the graph compiled and ran on the boards, but it failed the numerical check against the reference model that the board benchmark performs (relative error 0.43 at full precision), so the lookup was moved to the caller. In the files of the searched encoders, each zero-padding node of the PyTorch export is folded into the convolution that reads it, because the C code that ST Edge AI generates for a separate padding node did not compile. The files are imported by ST Edge AI Core [26], the vendor tool that compiles ONNX or TensorFlow Lite models to C for STM32, and benchmarked on the laboratory’s two boards through the ST Edge AI Developer Cloud, following the procedure the laboratory uses for its other models (`scripts/stedgeai_cloud.py`); Section 5.8 reports the results.

One-bit models also need a binary runtime, either the Larq Compute Engine [27] or the laboratory’s CBin-NN [23], and the trained network has to be rebuilt in a framework that these engines accept. ST Edge AI Core 4.0.0 marks its Larq layers as deprecated, so the vendor tool is not the planned route for one bit, and no one-bit file was measured. ONNX was chosen as the export format because it is portable, can be verified on the laptop and is accepted by the vendor tool, and because it gives the trained encoder a defined interface, which the application now uses (Section 4.6.1).

4.4 Hardware

Training, search and evaluation ran on a single laptop (Table 4.3). The June 2026 programme took 8.2 hours of wall-clock time. In October 2026 the reference conditions took 10.2 hours over 28 runs, the matrix rerun 5.6 hours over 27 runs, the recovered one-bit model 1.9 hours over three runs and the training of the board-search encoders and comparators 6.5 hours over 70 runs. The μ NAS searches ran in the Windows

Subsystem for Linux (WSL) on the same machine, with TensorFlow and Keras. One run of the reference conditions took 240 minutes instead of 21 because another process held most of the GPU memory while it trained; the incident is recorded in the log, and the run is excluded from all timing claims. Later runs capped PyTorch at 85% of the GPU memory, because on Windows the driver otherwise backs the growing allocation cache with system memory and slows training down; the cap does not change any result. The quantities reported alongside the ranking metrics were obtained in different ways, as summarised below.

Measurement status

Ranking quality is measured on the MIND and xMIND development impressions. **Size and RAM** are computed from parameter and buffer counts by the rules of Section 3.9; they are not file sizes. **Operations** are counted per layer. **Energy** is the per-operation proxy of Table 3.5. **Latency** in the matrix is a laptop timing, on the GPU through PyTorch and on the CPU through ONNX Runtime, for the three full-precision architectures, and is reported for relative comparison only. **Board figures** (Sections 5.8 and 5.9) are measurements made by ST Edge AI Core on the STM32H7B3I-DK and NUCLEO-F401RE boards of the ST Edge AI Developer Cloud: inference time on the board, and flash and RAM as allocated by the generated C code. **Phone figures** (Section 5.8) are measurements made on a DOOGEE S98Pro through ONNX Runtime inside the reader application: latency per title, and energy per title read from the phone’s battery gauge with the device unplugged. Energy on the boards is the only on-device quantity that was not measured, because the ST Edge AI Developer Cloud does not report it; it remains a per-operation proxy.

Every run from October 2026 onwards appends one record to a machine-readable log with the run name, the seed, the settings, the wall-clock minutes, the metrics, and the environment in which it ran (Python and PyTorch versions, GPU, code commit and whether the working tree was clean), and a dated logbook records what was run, why, how long it took and what was found. This practice was introduced after a two-process batch on 5 October 2026 saturated the GPU and the laptop had to be restarted. From then on, the programme was run one process at a time at reduced priority, with a sampler recording GPU utilisation, temperature and memory every thirty seconds. Appendix B lists the duration of every run of the October reference conditions.

4.5 Audit

The results of this thesis come from two programmes separated by an audit. The June 2026 programme produced the search, the nine-variant matrix, the NRMS baseline, the distillation ablation, the recovered binary model and the single-run per-language evaluation. In October 2026, before the writing began, the code and the result files

were audited, and the audit found three problems that change how the June numbers should be read.

1. The operation counter skipped the quantised layers. The library counter used in June did not recognise the fake-quantised convolution classes, so the eight-bit and one-bit rows of the matrix reported about a thirteenth of their operations and energy ratios between full and eight-bit precision of 7 to 36 times. A per-layer counter with hooks on every convolution and linear layer, including the quantised subclasses, replaced it; the corrected ratios are 3.4 to 9.3 times, the corrected RAM rule doubles the June buffer estimate, and the recovered binary model is 194 KB rather than the 186 KB first reported. Every cost in Chapter 5 is the corrected one, recomputed from the saved architectures; the ranking metrics themselves were unaffected.
2. The operation bound of the search was not enforced, for the same reason (Section 3.6). The board-specific μ NAS searches of Section 3.7 enforce it with the corrected count.
3. The June runs were single and not seeded, and their loop drew the negatives once. The October programme therefore runs the reference conditions, the nine matrix variants and the recovered one-bit model three times each, under a seeded loop that redraws the negatives.

The June architectures were kept and the nine matrix variants were rerun on them. Chapter 5 reports the rerun, with its corrected costs, and Appendix B lists the June single runs next to it. The October programme also adds the reference points, the distillation effect under the newer loop, the cross-lingual conditions and the board-specific searches.

4.6 Target Application

The deployment target and behavioural testbed is a multilingual RSS reader implemented as a cross-platform mobile application in React Native with Expo. FEEDWELL is a complete, publicly released reader. It provides feed aggregation and reading, article extraction through a readability pass, on-device translation through ML Kit, read-aloud through the platform speech engine, notes, a read-later list, a home-screen widget, an app tour, and an interface localised into thirty languages with right-to-left layout for Arabic, Hebrew and Farsi. All state is held locally and there is no server component, which is the property on which on-device ranking relies. The feed parser removes advertising and tracking domains during ingestion.

FEEDWELL-EDGE is the on-device personalisation engine built on this base during the internship. It was first created as a copy of FEEDWELL 1.6.22 and, in October 2026, rebuilt on FEEDWELL 1.18.2 as version 2.1.0 with its own application identifier (`com.feedwelledge.app`), name and URL scheme, so that it can be installed beside the public application on the same phone; the engine module and its hooks were

carried over unchanged. The release build was installed on the Android test phone. On the branch `edge-encoder`, the application also carries the eight-bit encoder: a local Expo module written in Kotlin computes the byte identifiers, performs the table lookup and runs the encoder with ONNX Runtime 1.27.0, and a research screen in the settings checks its vectors against the laptop and measures latency and battery drain (Section 5.8). Figure 4.1 shows the structure.

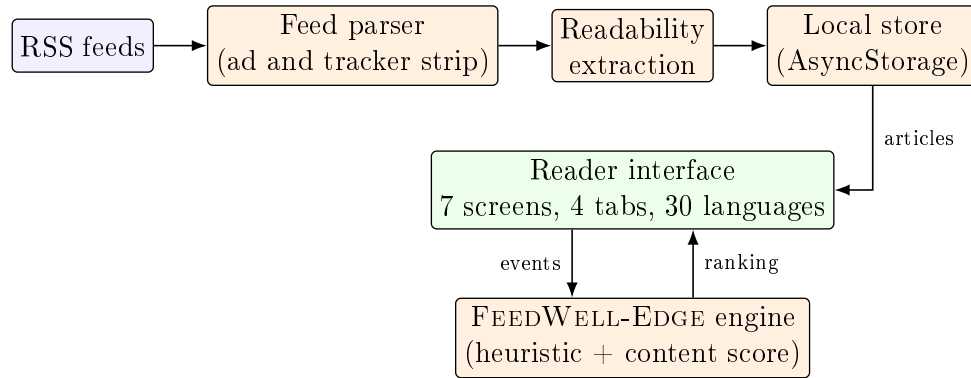


Figure 4.1: Application structure: local-only ingestion, reading and on-device personalisation. Nothing leaves the phone.

4.6.1 Personalisation engine

FEEDWELL-EDGE implements an on-device personalisation engine as a closed, entirely local learning loop (Figure 4.2). The reader emits interaction signals (impressions, opens, read sessions with dwell time and scroll depth, searches and stage transitions) into a capped local event store of 5,000 events. A read is counted complete when the scroll depth reaches 80% or the dwell reaches 45 seconds. From these events the engine maintains an additive per-topic weight, clamped to $[-20, 20]$, with a fast and a slow exponential moving average per topic whose difference is the drift signal, and re-ranks the feed by the sum of the topic weight, a drift boost, a freshness boost that decays over 48 hours and an unread boost; the encoder's content score is added to this sum, as described below. The parameters are listed in Table 4.4, and the full rules, including the exact increments and the telemetry snapshot, are given in Appendix C.

The engine and the neural model of Chapters 3 to 5 began as separate artifacts and have since been combined. FEEDWELL-EDGE keeps its transparent heuristic, but the eight-bit encoder is now bundled into the reader, and its content score is added to the heuristic score through the storage schema of Section 4.3, reusing the existing event pipeline, replay cursor, drift detection and telemetry (Figure 4.3). The content score of a candidate is the dot product of its vector, computed with the application's eight-bit file, and the user vector that the user encoder of the same checkpoint pools from the titles of the reader's last fifty opened or read articles. The scores are converted into z-scores over the loaded articles, and the feed adds the z-score, multiplied by a content weight, to the topic score. Content-aware ranking is on by default with a

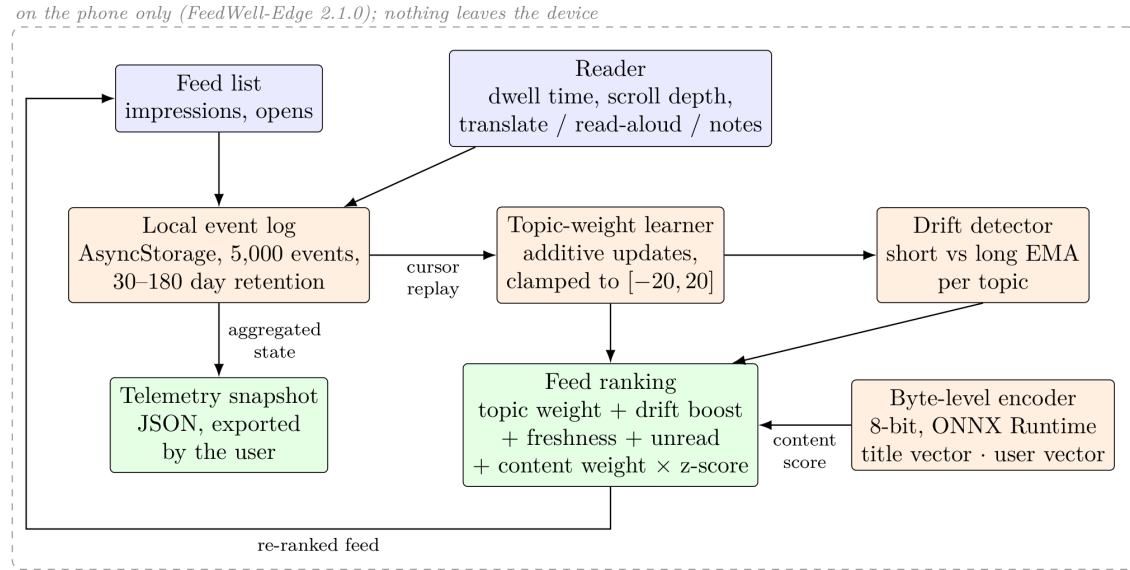


Figure 4.2: The FEEDWELL-EDGE learning loop, closed on the phone. By default the ranking also adds the z-scored content score of the eight-bit byte-level encoder (bottom right), multiplied by a content weight of 2 (Section 5.10).

weight of 2, the value chosen in the replay of Section 5.10; the settings offer 0.5, 1, 2 and 4, and impressions and opens record the content score and the weight in the event log. That section evaluates the combination in an offline replay of MINDsmall through a Python port of the application’s ranking rule: the combined score is 0.024 AUC above the heuristic alone. The engine serves both as the host of the encoder and as the event-capture system from which an integrated recommender can learn. No behavioural data from the application are used in any result of this thesis; at the time of writing no reader had used the content-aware ranking, and a live, on-device comparison of the two rankings remains to be done.

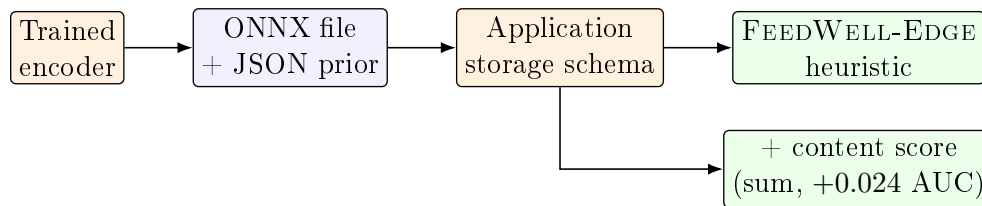


Figure 4.3: The export interface: the trained encoder is written against the application’s storage schema, and its content score is combined with the heuristic ranking through that schema, reusing the same event pipeline. In an offline replay of MINDsmall the sum scores 0.024 AUC above the heuristic alone.

Table 4.2: The main modules and scripts of the MIND-EDGE-RECOMMENDER repository.

File	Role
<code>src/student.py</code>	byte-level encoder, distillation loop, teacher-cosine evaluation
<code>src/teacher.py</code>	frozen teacher, cached anchors per split and language
<code>src/recommender.py</code>	user encoder, ranking loop, impression-level and cross-lingual evaluation
<code>src/baseline_nrms.py</code>	NRMS in the GloVe reference and the reduced configuration
<code>src/quantize.py, src/binary.py</code>	fake-quantised layers; ReActNet and Bi-Real binary encoder
<code>src/nas/search.py</code>	evolutionary search with the hard constraints
<code>src/footprint.py</code>	per-layer counter for size, RAM, operations and energy
<code>src/metrics.py</code>	AUC, MRR, nDCG with midrank ties; popularity and histogram scorers
<code>src/heuristics.py</code>	ONNX export and cold-start prior in the application schema
<code>src/export.py</code>	
<code>src/runlog.py</code>	per-run records with environment, seed, settings, duration and metrics
<code>scripts/run_full.py</code>	the June 2026 programme: search, matrix, baseline, ablation
<code>scripts/run_p1.py</code>	the October 2026 programme: reference conditions over seeds
<code>scripts/run_matrix.py,</code> <code>scripts/run_binary.py</code>	October 2026 reruns of the nine matrix variants and of the recovered one-bit model over three seeds
<code>unas/, src/unas_encoder.py</code>	adapter for the laboratory's μ NAS fork (data export, trainer patch); PyTorch rebuild of the searched networks
<code>scripts/run_unas_full.py,</code> <code>scripts/rechoose_unas.py</code>	full training of the board-search encoders and comparators; choice after full distillation
<code>scripts/recompute_costs.py</code>	corrected costs for the June matrix from the saved architectures
<code>scripts/check_numbers.py</code>	checks every number quoted in the paper draft against the result files
<code>scripts/make_figures.py,</code> <code>scripts/make_tables.py</code>	figures and tables of the paper draft
<code>scripts/deploy_boards.py,</code> <code>scripts/export_int8.py</code>	deployment files: fixed-batch ONNX export, eight-bit QDQ quantisation, accuracy of the exported files
<code>scripts/stedgeai_cloud.py</code>	analysis and board benchmarks through the ST Edge AI Developer Cloud
<code>scripts/export_app_assets.py,</code> <code>scripts/phone_energy.py</code>	encoder files for the application; energy per title from the phone's battery log
<code>src/app_learner.py,</code> <code>scripts/replay_app.py</code>	Python port of the application's ranking rule; replay of MIND through it
<code>scripts/make_thesis_assets.py</code>	figures and tables of this thesis
<code>scripts/check_thesis_numbers.py</code>	checks the numbers of the thesis draft of 7 October 2026 against the result files

Table 4.3: The measurement laptop.

Component	Specification
CPU	Intel Core Ultra 9 275HX (24 cores)
GPU	NVIDIA GeForce RTX 5070 Laptop GPU (8 GB)
Memory	32 GB
Software	Windows 11, Python 3.13.14, PyTorch 2.12.1 with CUDA 13, ONNX Runtime for CPU timings
June programme	494.4 min (8.2 h), one batch
October: reference conditions	612.5 min (10.2 h), 28 runs, one process at a time
October: matrix rerun	337.5 min (5.6 h), 27 runs
October: one-bit rerun	112.9 min (1.9 h), 3 runs
October: board searches	searches in WSL, 1 h 44 min (H7) and 1 h 34 min (F401); training of the chosen encoders and comparators 387.2 min (6.5 h), 70 runs

Table 4.4: FEEDWELL-EDGE learning-loop parameters, as implemented in version 2.1.0.

Parameter	Value	Role
Completion threshold	scroll $\geq 80\%$ or dwell ≥ 45 s	marks a read complete
Weight increments	open: +0.8; read session: $0.5 + \min(d/90, 1.2) + \min(s/100, 1.0)$, plus 0.5 if complete	d dwell in seconds, s scroll depth in percent
Weight clip	$[-20, 20]$	bounds each topic weight
Fast and slow EMA rates	$\alpha_f = 0.3$, $\alpha_s = 0.05$	short- and long-term interest per topic
Drift flag	divergence ≥ 0.35 after at least 5 events on the topic	marks a shifting interest
Ranking score	topic weight + drift boost + freshness boost + unread boost, plus content weight $\times$ content z-score when content-aware ranking is on	freshness decays to zero over 48 h; unread adds 0.35; content weight 2 by default
Event store	5,000 events; purge of events older than 30 days by default	bounded storage

Chapter 5

Results

This chapter reports the reference points, the architecture-and-precision grid, the binarisation experiments, the cross-lingual and history-length views, the cost trade-off, the board and phone measurements, the searches under the board budgets and the in-app ranking, and then discusses them together. All ranking figures are measured on the 73,152 labelled MINDsmall development impressions, in English unless a language is named. The size, RAM, operation and energy figures of the matrix come from the corrected cost model of Section 3.9, and its latency is a laptop timing; Section 5.8 adds the measured eight-bit files and the board and phone figures. Unless a June 2026 single run is named for comparison, every value is a mean, with its standard deviation, over three seeds of the October 2026 loop, which seeds the initialisation and redraws the negatives at every step (Table 3.6).

5.1 Reference Points

Table 5.1 places the reference encoder, the 64-5-384 byte-level network in full precision, against the scorers and models of Section 3.11, and Figure 5.1 draws the English AUC against the number of news-encoder parameters.

Table 5.1: Reference points and the reference byte-level encoder (64-5-384, full precision) on MINDsmall dev, October 2026 programme. Mean $\pm$ standard deviation over three seeds; the last column is the mean AUC over the fourteen xMIND translations; “mixed clicks” means each training instance is shown in a random xMIND language. Histogram scorers use MIND’s editorial labels. Params = news encoder.

Scorer	Params	AUC	MRR	nDCG@10	14-lang. AUC
Popularity	–	0.525	0.255	0.299	–
Category histogram	–	0.598	0.307	0.354	–
Subcategory histogram	–	0.617	0.341	0.384	–
NRMS, reduced (random 256-d table)	7.08 M	0.604 $\pm$ 0.010	0.322 $\pm$ 0.009	0.364 $\pm$ 0.009	–
NRMS, GloVe reference config.	11.29 M	0.664 $\pm$ 0.003	0.369 $\pm$ 0.002	0.415 $\pm$ 0.002	–
Frozen teacher + user encoder	–	0.589 $\pm$ 0.000	0.319 $\pm$ 0.000	0.360 $\pm$ 0.000	0.575 $\pm$ 0.000
+ trainable linear map	–	0.618 $\pm$ 0.002	0.327 $\pm$ 0.001	0.374 $\pm$ 0.001	0.590 $\pm$ 0.002
Byte-CNN, random start, EN clicks	68 k	0.618 $\pm$ 0.002	0.329 $\pm$ 0.007	0.371 $\pm$ 0.004	0.528 $\pm$ 0.002
Byte-CNN, random start, mixed clicks	68 k	0.584 $\pm$ 0.005	0.308 $\pm$ 0.013	0.348 $\pm$ 0.009	0.557 $\pm$ 0.001
Byte-CNN, distilled, frozen	68 k	0.573 $\pm$ 0.004	0.297 $\pm$ 0.002	0.340 $\pm$ 0.003	0.572 $\pm$ 0.001
Byte-CNN, distilled start, EN clicks	68 k	0.629 $\pm$ 0.003	0.338 $\pm$ 0.005	0.383 $\pm$ 0.003	0.557 $\pm$ 0.002
Byte-CNN, distilled start, mixed clicks	68 k	0.609 $\pm$ 0.008	0.312 $\pm$ 0.004	0.360 $\pm$ 0.006	0.589 $\pm$ 0.005

First, the NRMS reproduction in the reduced configuration of the June run, with a randomly initialised 256-dimensional word table that holds 92% of its 7.08 million

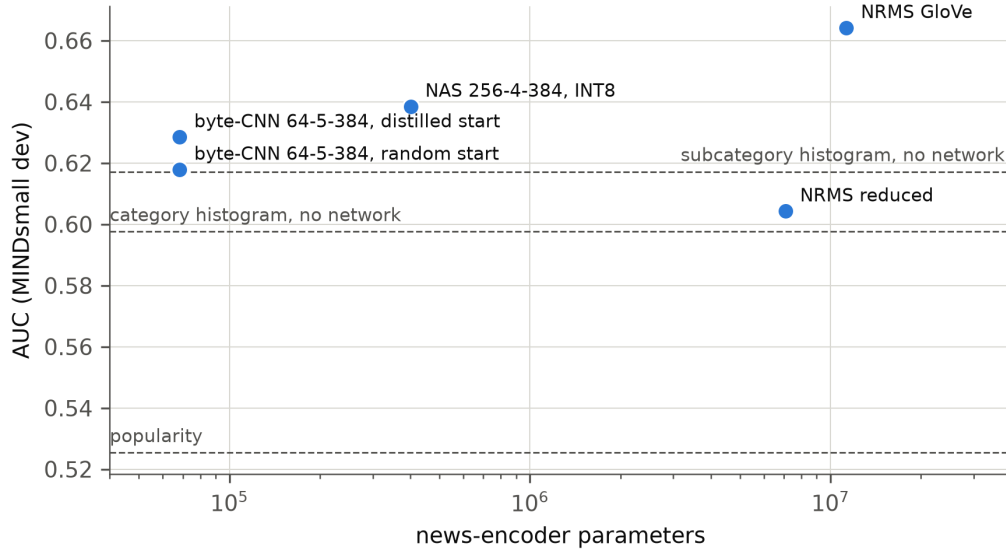


Figure 5.1: English AUC against news-encoder parameters (log scale). Points are three-seed means; dashed lines are the scorers with no network.

parameters, averages 0.604 over three seeds with a spread of 0.010, which covers the 0.607 of the June single run. In the reference configuration distributed with MIND, with the GloVe-initialised 300-dimensional table and twenty heads, NRMS reaches 0.664, in line with the 0.656 reported for this split by the DIGAT authors [66]. The two NRMS figures differ by 0.060. The configurations also differ in table width and number of heads, but the most likely cause is the pre-trained table: the GloVe vectors bring lexical knowledge from a corpus of billions of tokens into the model, whereas neither the reduced configuration nor a byte-level encoder trained on 157 thousand impressions has seen such a corpus.

Second, the scorers that use no network set a floor that a content encoder has to clear. Popularity gives 0.525. A histogram of the reader’s clicks over MIND’s seventeen category labels gives 0.598, and over the 264 subcategory labels 0.617, with MRR and nDCG@10 (0.341 and 0.384) level with those of the best byte-level encoder. These scorers use editorial labels that MIND provides but a live feed does not. A content encoder can derive similar information from the title alone, which is the situation the application is built for.

Third, the byte-level encoder trained on English clicks scores 0.618 from a random start and 0.629 from the distilled weights, so the distilled initialisation adds 0.011 under the October loop, less than the 0.033 of the June ablation (0.600 against 0.633, single runs, ten epochs, negatives drawn once). The distilled encoder is 0.036 below GloVe NRMS with 165 times fewer news-encoder parameters, 0.024 above the reduced NRMS and 0.011 above the subcategory histogram. The frozen teacher under a trained user encoder scores 0.589, and 0.618 with a trainable linear map. The student fine-tuned on clicks exceeds both, so once click supervision is available the distilled representation

serves as a starting point that training improves on.

5.2 Search Outcomes

The June search returned a different architecture in each arm (Figure 5.2): a wide, deep 256-channel four-block network unconstrained; a narrow, deep 64-channel five-block network under the budget; and a wider, shallower 96-channel two-block network when searching for binary weights. Without constraints, the search adds width. Under the budget it gives up width first, narrowing to 64 channels while keeping depth; a likely reason is that a block costs C^2 weights, so at this scale depth is the cheaper way to add capacity. For binary weights it does the opposite and trades depth for width, which is consistent with each binary layer losing information, so that a shallower, wider network loses less across the stack.

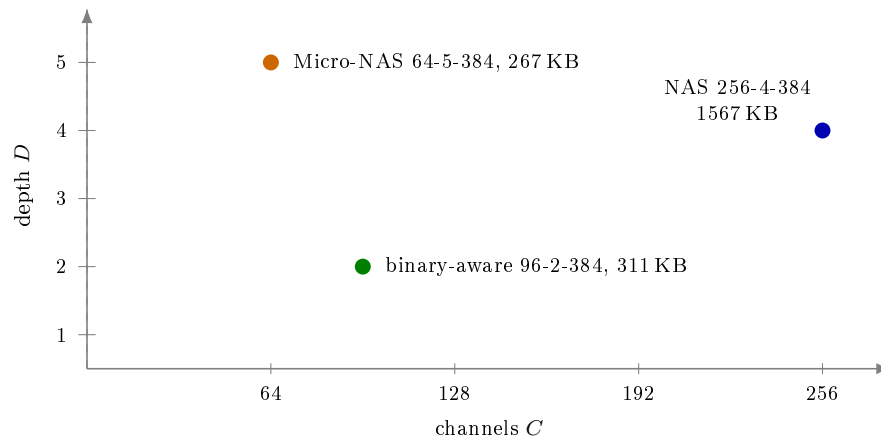


Figure 5.2: The architecture selected by each arm in the channels–depth plane (all three chose a 384-dimensional output). Sizes at full precision. The budget trades width for depth; binary awareness trades depth for width.

The matrix of Section 5.3 gives the cost of the budget. At full precision, moving from the unconstrained to the constrained architecture lowers AUC from 0.628 to 0.619, a difference of 0.009 that lies inside the spread over seeds of the unconstrained arm (0.015), for a model 5.9 times smaller (1567 KB to 267 KB) with 11 times fewer multiply-accumulates (36.1 to 3.29 million). With eight-bit weights the difference is 0.024 (0.639 against 0.614), larger than either standard deviation. The June search, however, enforced only the flash and RAM limits, not the operation limit: both constrained winners exceed the bound of 2 million multiply-accumulates that they were meant to respect (Section 3.6). The June constrained arms should therefore be read as flash-constrained; the searches of Section 5.9 enforce all three limits.

5.3 Precision

Table 5.2 is the complete grid, Figure 5.3 shows its AUC cells and Figure 5.4 plots AUC against size and against the energy proxy.

Table 5.2: Architecture $\times$ weight precision on MINDsmall dev: October 2026 rerun of the June architectures, AUC as mean $\pm$ standard deviation over seeds 42, 1 and 2, MRR and nDCG@10 as means. Each architecture is trained from a random initialisation on English clicks; INT8 and 1-bit are the trained model with simulated quantisation of the inner block weights and two more epochs, so off-diagonal cells are ablations. †: the precision the arm was searched at. Size, RAM and energy are estimates by the rules of Section 3.9. Operations do not change with precision.

Arm	Prec.	AUC	MRR	nDCG@10	Size (KB)	RAM (KB)	MMAC	Energy (μ J)
NAS	FP32†	0.628 $\pm$ 0.015	0.326	0.373	1566.8	256	36.14	166.3
NAS	INT8	0.639 $\pm$ 0.005	0.333	0.381	789.8	64	36.14	17.91
NAS	1-bit	0.598 $\pm$ 0.014	0.302	0.350	563.1	64	36.14	12.54
Micro-NAS	FP32	0.619 $\pm$ 0.001	0.327	0.371	266.8	64	3.29	15.15
Micro-NAS	INT8†	0.614 $\pm$ 0.006	0.323	0.365	203.9	16	3.29	3.16
Micro-NAS	1-bit	0.554 $\pm$ 0.017	0.277	0.320	185.6	16	3.29	2.72
Bin. μ NAS	FP32	0.602 $\pm$ 0.003	0.301	0.350	311.4	96	3.26	14.98
Bin. μ NAS	INT8	0.601 $\pm$ 0.009	0.302	0.351	255.7	24	3.26	4.35
Bin. μ NAS	1-bit†	0.558 $\pm$ 0.007	0.271	0.317	239.5	24	3.26	3.96

Moving the inner weights from full to eight-bit precision changes AUC by +0.011, −0.005 and −0.001 in the three arms (per-seed values in Table B.1). All three differences lie within the spread over seeds, so eight-bit weights have no measurable cost in accuracy. The benefit appears in the cost columns. The energy proxy falls by a factor of 3.4 to 9.3, because the inner blocks perform most of the operations, and the size falls by a factor of 1.2 to 2, because the inner blocks hold between a quarter and two thirds of the parameters (Table 3.3, Figure 5.5). The baseline behaves in the same way: NRMS in the reduced configuration moves from 0.6045 to 0.6047 when every weight is quantised to eight bits and shrinks from 27 to 6.8 MB, but it is still 34 times larger than the constrained encoder at 204 KB, because most of its size is the table.

Run-to-run spread

The standard deviation over seeds reaches 0.017 in Table 5.2 (the one-bit constrained model), so differences below about 0.02 between cells are read as unresolved; this covers every full-precision to eight-bit difference. The June single run of each cell lies within 0.033 of its October mean, and the largest difference is the one-bit constrained model: 0.521 in June against 0.554 ± 0.017 in October.

On the laptop CPU, through ONNX Runtime, a title takes 0.34, 0.19 and 0.10 ms for the three full-precision architectures, and under 0.7 ms on the GPU, where the launch overhead dominates and the smallest network is not the fastest. The estimated working memory is 16 KB for the constrained encoder with eight-bit weights and 64 KB for the unconstrained one, both inside the SRAM of either target board.

	deployment precision →		
	FP32	INT8	1-bit
NAS (unconstrained) winner 256-4-384	0.628 ± 0.015	0.639 ± 0.005	0.598 ± 0.014
Micro-NAS (256 KB, 128 KB) winner 64-5-384	0.619 ± 0.001	0.614 ± 0.006	0.554 ± 0.017
Binary-aware Micro-NAS winner 96-2-384	0.602 ± 0.003	0.601 ± 0.009	0.558 ± 0.007

AUC on MINDsmall dev, mean ± standard deviation over three seeds (October 2026 rerun of the June architectures, seeded loop). Shaded: the precision each arm was searched at. Weight-only simulated quantisation of the inner convolutions. Sizes: 1567 / 790 / 563 KB (NAS), 267 / 204 / 186 KB (Micro-NAS), 311 / 256 / 239 KB (binary-aware).

Figure 5.3: The AUC cells of the matrix, mean and standard deviation over three seeds. Shaded cells are the precision each arm was searched at.

Headline operating point

Micro-NAS with eight-bit inner weights: 0.614 ± 0.006 AUC (three seeds) at an estimated 204KB and 16 KB of working memory, the accuracy of the reduced NRMS (0.604 ± 0.010) with 136 times less storage, and 0.050 below GloVe NRMS (0.664) with 216 times less. The same architecture at full precision, started from the distilled weights, reaches 0.629 at 267KB; exported as an eight-bit file it scores 0.622 (0.624 over three seeds) and runs on the boards (Section 5.8).

5.4 Binarisation

One-bit weights mark the limit of what the compression techniques achieve on this encoder (Figure 5.6). Naive sign binarisation of the inner weights costs the constrained architecture 0.065 AUC (0.554 ± 0.017) and the binary-aware arm’s own winner 0.044 (0.558 ± 0.007); the wide unconstrained network loses least, 0.029 (0.598 ± 0.014), which is consistent with the search’s preference for width under binary weights. The recovery recipe of Section 3.8, full binary convolutions with ReActNet activations and Bi-Real shortcuts, started from the distilled weights, raises the constrained architecture to 0.587 ± 0.012 over three seeds (0.573, 0.585 and 0.601; 0.572 in the single June run) at an estimated 194KB: 109,248 parameters, of which 61,440 are one-bit, and 8.41 million operations per title, 7.86 million of them binary. The gain of 0.032 over the naive model is nearly twice the larger of the two standard deviations.

Even after recovery, the one-bit model stays 0.028 below the eight-bit model of the same architecture, 0.018 below the reduced NRMS (0.604) and 0.042 below the distilled encoder at full precision (0.629), and its size advantage is small. Because the embedding, the first projection and the head stay in full precision, one-bit weights save 18 KB over eight-bit ones at this size (186 against 204KB), and the recovered model,

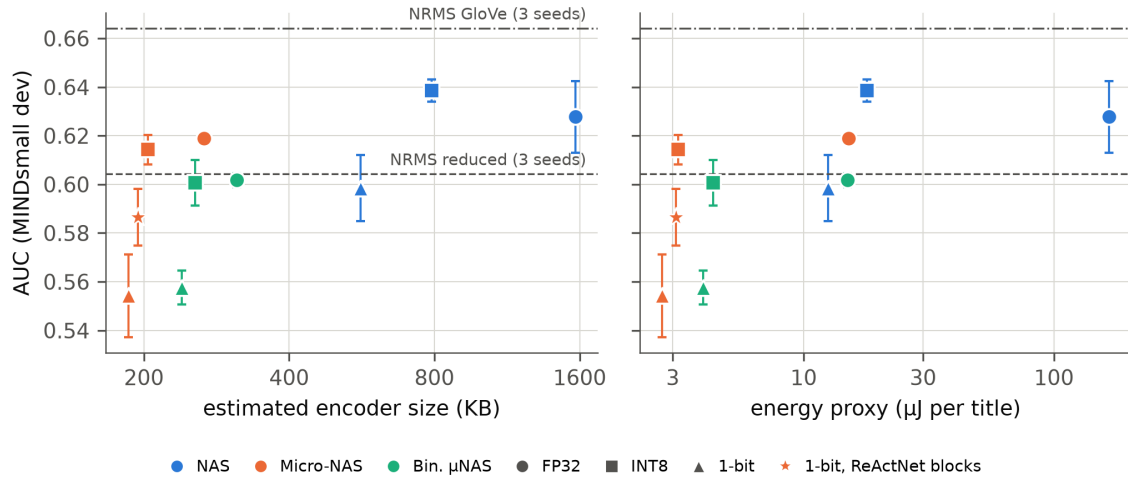


Figure 5.4: AUC against estimated size (left) and against the energy proxy (right), by arm (colour) and precision (marker), mean and standard deviation over three seeds; the star is the recovered one-bit model of Section 5.4, also over three seeds. Dashed line: the reduced NRMS; dash-dotted line: GloVe NRMS (both three seeds). The word models lie far to the right at 27 and 43 MB.

whose full convolutions carry more weights, is only 10 KB smaller than the eight-bit model. In the accuracy-against-size view (Figure 5.4) the binary points lie below and only slightly to the left of their eight-bit counterparts. For this task, eight-bit weights are the recommended operating point and one-bit weights are not recommended. Since the Cortex-M7 has no population-count instruction, no speed advantage can be claimed for one bit without a measurement.

5.5 Cross-Lingual Transfer

Table 5.3 gives the AUC of four encoders on each of the fourteen translated development sets, and Figure 5.7 shows the three byte-level conditions with their English score for reference.

Cross-lingual transfer depends on how the encoder is trained. Trained from a random start on English clicks, the byte-level encoder reads the fourteen translated development sets at 0.528 on average, close to chance; the matrix models are of this kind, and the June single run of the constrained full-precision model gave 0.522 on the same evaluation (Appendix B). The distilled student alone, frozen under a trained user encoder, scores 0.572 across languages against 0.573 in English, so distillation does transfer the teacher’s language alignment into 68 thousand parameters; the teacher itself scores 0.575 against 0.589, and with a trainable linear map 0.590 against 0.618. Fine-tuning the distilled encoder on English clicks raises English to 0.629 but leaves the translations at 0.557, so part of the alignment is lost in fitting the English clicks.

Showing the same clicks in a random xMIND language instead keeps the mean at 0.589 across languages, from 0.538 (Georgian) to 0.609 (Finnish), with 0.609 in English: a cost of 0.020 in English for a gain of 0.032 across languages. The 68-

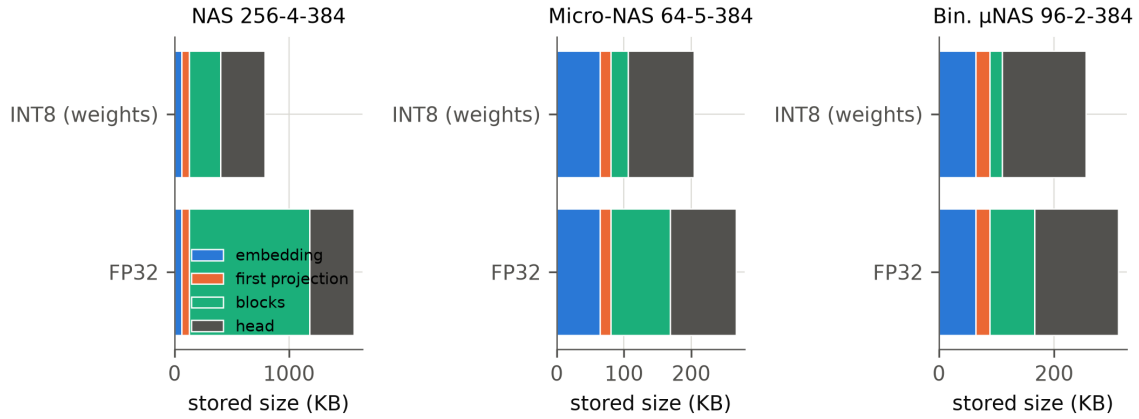


Figure 5.5: Stored size by layer group in the three searched architectures, at full precision and with eight-bit inner weights. The byte embedding (64 KB) and the head are fixed costs that quantisation does not change.

thousand-parameter student comes within 0.001 of the 0.590 that the whole teacher reaches through a linear map. Mixed-language clicks without distillation reach 0.557 across languages but cost 0.034 in English (0.584 against 0.618), so the distilled start keeps the English cost of multilingual training small.

The per-language results do not follow a simple pattern. Transfer does not track the writing system cleanly: under mixed clicks the Latin-script languages span 0.596 to 0.609, Chinese and Japanese, whose characters take three bytes each, reach 0.592 and 0.586, and the three weakest languages, Thai (0.554), Tamil (0.544) and Georgian (0.538), also use three-byte scripts. For Tamil the teacher is weak as well (0.524 through the linear map), so the student is limited by its anchor. For Thai and Georgian the teacher scores 0.592 and 0.582, so the loss arises in the student, which is consistent with a 128-byte window holding only about forty characters of a three-byte script. Somali shows the opposite case: the teacher gives 0.539, yet the student trained on mixed clicks reaches 0.606, so click supervision compensates where the anchor is poor. These are interpretations of a single table, not tested hypotheses; they indicate where multilingual anchors and a longer window should be tried first.

One set of weights, with no per-language table, ranks all fourteen additional languages above 0.53 AUC and eleven of them above 0.58. This answers RQ1: the distilled byte-level encoder carries a partly language-agnostic representation, and the training signal determines how much of it is preserved.

5.6 History Length

Table 5.4 and Figure 5.8 split the development impressions by the length of the reader’s history. For the 3.0% of impressions whose reader has no history, a history-based scorer ties every candidate and scores 0.500 by construction; popularity gives 0.529 there and the population’s category distribution 0.533, which is what the exported

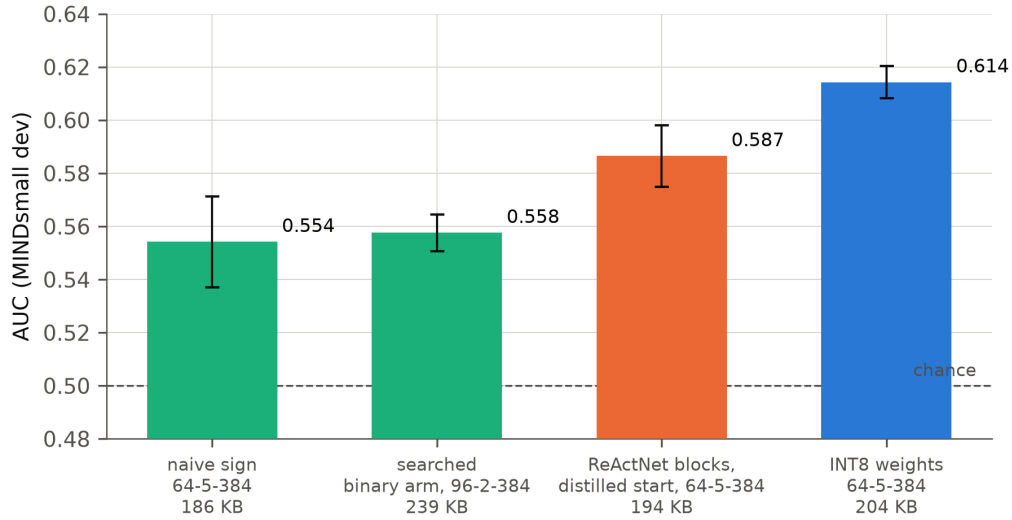


Figure 5.6: One-bit inner weights on the constrained architecture: the naive sign model, the binary-aware arm’s winner, the recovered ReActNet model and the eight-bit reference, with estimated sizes. October 2026 runs, mean and standard deviation over three seeds.

cold-start prior of Section 4.3 offers a new user. In the other buckets the distilled byte-level encoder scores 0.615 for readers with one to five clicks, 0.633 for six to twenty and 0.638 for more than twenty, against 0.622, 0.664 and 0.688 for GloVe NRMS, so the gap widens with history length, from 0.007 to 0.050. The subcategory histogram, which uses no network, scores 0.570, 0.613 and 0.640 in the same buckets: 0.044 below the byte-level encoder for short histories and level with it for long ones. The encoder is therefore relatively strongest for readers about whom the system knows little, which is the situation of a newly installed on-device application in its first days. With long histories, the lexical knowledge in the GloVe table, and even the editorial labels, are worth more.

5.7 Cost Trade-off

Table 5.5 collects the cost of every variant, and Figure 5.4 plots accuracy against size and against energy. In the accuracy-against-size view the eight-bit models lie on or next to the frontier: the unconstrained eight-bit model gives the highest quality (0.639 at 790 KB), and the constrained eight-bit model sits at the knee, at 204 KB with 16 KB of working memory; the same architecture at full precision is 0.005 higher, within the spread, at 267 KB. The energy proxy follows operations and not storage, so it orders the variants differently: the eight-bit models cost 3 to 18 μ J per title, while the full-precision versions of the same architectures cost 3.4 to 9.3 times more. The one-bit models save little over eight-bit on either axis, for the reasons given in Section 5.4.

The table also shows the cost of reading bytes. The constrained encoder spends 3.29 million multiply-accumulates on a title because it processes 128 positions where

Table 5.3: AUC per language, October 2026 programme, means over three seeds. The byte-level encoder is 64-5-384 in full precision; “random, EN” is trained from a random start on English clicks, “distilled, EN” from the distilled weights on English clicks, “distilled, mixed” from the distilled weights on clicks shown in a random xMIND language; “teacher + linear” is the frozen teacher with a trainable linear map.

Code	Language	random, EN	distilled, EN	distilled, mixed	teacher + linear
en	English	0.618	0.629	0.609	0.618
zho	Chinese	0.500	0.532	0.592	0.602
fin	Finnish	0.535	0.572	0.609	0.618
kat	Georgian	0.511	0.523	0.538	0.582
grn	Guarani	0.546	0.568	0.597	0.588
hat	Haitian Creole	0.535	0.585	0.606	0.577
ind	Indonesian	0.543	0.587	0.605	0.620
jpn	Japanese	0.518	0.522	0.586	0.604
ron	Romanian	0.543	0.576	0.599	0.616
som	Somali	0.525	0.564	0.606	0.539
swh	Swahili	0.559	0.587	0.606	0.574
tam	Tamil	0.513	0.518	0.544	0.524
tha	Thai	0.502	0.525	0.554	0.592
tur	Turkish	0.535	0.567	0.608	0.617
vie	Vietnamese	0.521	0.571	0.596	0.612
mean of 14		0.528	0.557	0.589	0.590

Table 5.4: AUC by history length, October 2026 programme (byte-level encoder and NRMS: three-seed means). Buckets hold 3.0%, 14.7%, 34.8% and 47.5% of the 73,152 development impressions.

Scorer	Impressions	no history	1–5 clicks	6–20 clicks	>20 clicks
Popularity	73,152	0.529	0.545	0.535	0.512
Category histogram	73,152	0.533	0.563	0.597	0.613
Subcategory histogram	73,152	0.533	0.570	0.613	0.640
NRMS, GloVe	73,152	0.500	0.622	0.664	0.688
Byte-CNN, distilled start, EN clicks	73,152	0.500	0.615	0.633	0.638

a word model processes about twenty, so removing the vocabulary table is paid for in arithmetic. On a microcontroller this trade is favourable, since flash is fixed at manufacture, while a few million integer operations can be executed in tens of milliseconds on a Cortex-M7 running at hundreds of megahertz; Section 5.8 measures the actual time.

5.8 Board and Phone Measurements

The deployment files of Section 4.3 were measured in three ways: their ranking quality on the development impressions, computed on the laptop from the exported files; their latency, flash and RAM on the two boards of the ST Edge AI Developer Cloud (Tables 5.6 and 5.7, Figure 5.9); and their latency and energy on the smartphone that

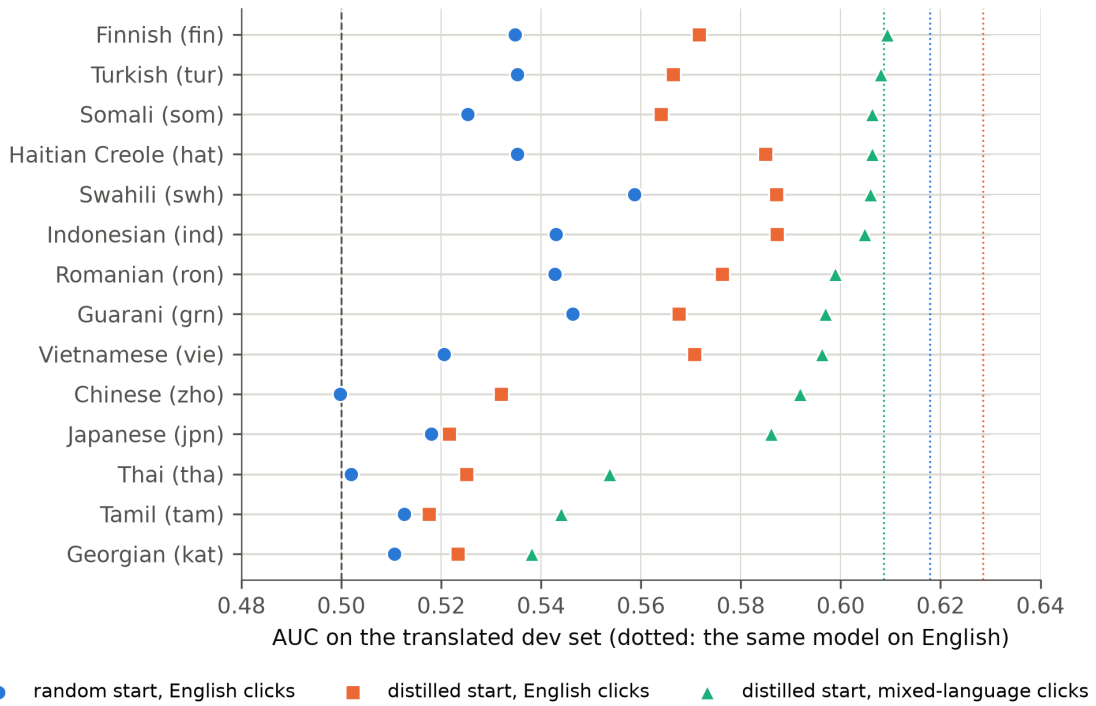


Figure 5.7: AUC on each translated development set for the three byte-level training conditions (three-seed means), languages ordered by the mixed-click result. Dotted vertical lines: the same model on English. Dashed line: chance.

hosts the application. The board and phone figures are the first in this thesis that come neither from the cost model nor from the laptop.

Exported as an eight-bit file, with eight-bit weights and eight-bit inputs to every convolution and to the output layer while activation functions and averages stay in floating point, the constrained encoder scores 0.622 AUC, against 0.628 for the same weights at full precision. The loss of 0.006 is the first measured cost of an eight-bit file in this thesis. Unlike the differences in the matrix, it compares one trained model with itself, so run-to-run variation does not affect it; over the three seeds of the reference encoder the loss is 0.004 on average (0.624 ± 0.004 against 0.629 ± 0.003). The news vectors of the two files agree closely: their cosine similarity over the 42,416 development titles has a mean of 0.99 and a minimum of 0.937.

On the STM32H7B3I-DK the eight-bit constrained encoder takes 31.19 ms per title, 8.73 million cycles, against 76.99 ms at full precision. Its weights take 53.2 KB, the flash image of the network with its runtime library 77.7 KB, and its working memory 49.4 KB, all of it in the board’s tightly coupled memory; with the 64.3 KB byte table the encoder needs 142.0 KB of flash. On the NUCLEO-F401RE the same file takes 449.64 ms and 44.5 KB of RAM, so the constrained encoder also fits the smaller board. The check that the Developer Cloud runs on the target, which executes the generated C code on random inputs and compares its outputs with the reference model, gave a relative error of zero for every eight-bit file and below 10^{-5} for every full-precision one.

Eight-bit kernels make the constrained encoder 2.5 times faster on the Cortex-M7,

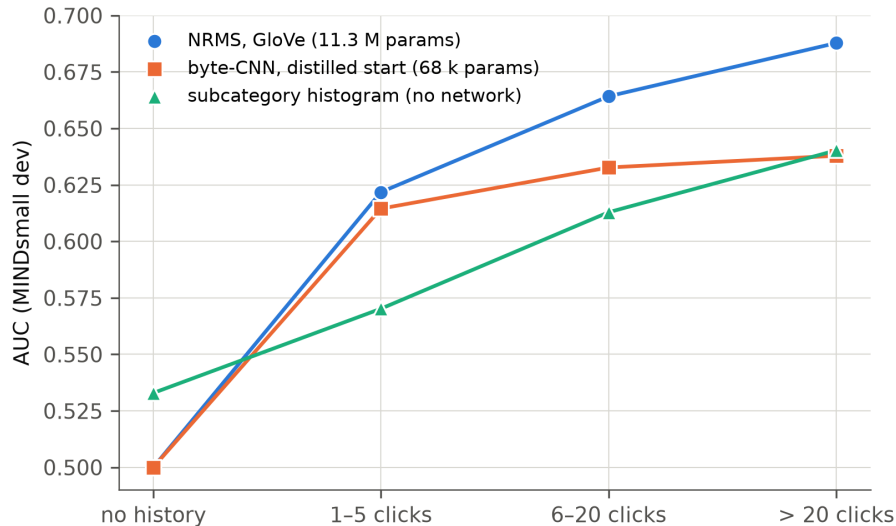


Figure 5.8: AUC by history length for GloVe NRMS, the distilled byte-level encoder and the subcategory histogram.

Table 5.5: Cost per title inference. Size, RAM and energy are estimates (Section 3.9); operations are counted; latency is a laptop timing per architecture (ONNX Runtime on the CPU, PyTorch on the GPU) and does not depend on the simulated precision.

Arm	Precision	Size (KB)	RAM (KB)	MMAC	Energy (μ J)	CPU (ms)	GPU (ms)
NAS	FP32	1566.8	256	36.14	166.3	0.34	0.57
NAS	INT8	789.8	64	36.14	17.91		
NAS	1-bit	563.1	64	36.14	12.54		
Micro-NAS	FP32	266.8	64	3.29	15.15	0.19	0.65
Micro-NAS	INT8	203.9	16	3.29	3.16		
Micro-NAS	1-bit	185.6	16	3.29	2.72		
Micro-NAS	1-bit, ReActNet blocks	194.3	16	8.41	3.09		
Binary-aware	FP32	311.4	96	3.26	14.98	0.10	0.43
Binary-aware	INT8	255.7	24	3.26	4.35		
Binary-aware	1-bit	239.5	24	3.26	3.96		

the binary-aware architecture 1.4 times (84.14 to 58.57 ms) and the unconstrained one 1.3 times (837.37 to 649.41 ms); part of the latter’s 188.5 KB of working memory lies outside the tightly coupled memory, in the board’s general RAM. On the Cortex-M4 the eight-bit files are no faster than the full-precision ones (449.64 against 444.08 ms for the constrained encoder, 431.30 against 397.22 ms for the binary-aware one), so on that board the benefit of eight bits lies only in flash and RAM. Encoding the thirty titles of a typical feed refresh would take about 0.9 s on the H7 and 13.5 s on the F401; each title is encoded once, since news vectors are cached.

The smartphone that hosts the application was measured with ONNX Runtime inside the reader instead of the vendor tool. The phone is a DOOGEE S98Pro (MediaTek Helio G96, two Cortex-A76 and six Cortex-A55 cores, Android 12). The eight-bit file built into the reader contains the encoder trained on mixed-language clicks. It computes the same vectors as on the laptop, with a largest difference of 4.9×10^{-7} over

Table 5.6: Accuracy of the deployed files of the constrained encoder (64-5-384, distilled start, English clicks, seed 42) on the 73,152 development impressions, scored through the saved user encoder. Sizes are those of the ONNX files of the network; the 257×64 byte table (64.3 KB at 32 bits) is applied by the caller and not included.

Model file	Size (KB)	AUC	MRR	nDCG@5	nDCG@10
PyTorch, full precision (reference)	–	0.628	0.344	0.323	0.387
ONNX, full precision	204.1	0.628	0.344	0.323	0.387
ONNX, 8-bit QDQ (integer kernels)	88.8	0.622	0.341	0.321	0.385

Table 5.7: Measured on the STM32H7B3I-DK (Cortex-M7, 280 MHz) and NUCLEO-F401RE (Cortex-M4, 84 MHz) boards of the ST Edge AI Developer Cloud, ST Edge AI Core 4.0.1, one title per inference. Weights and flash total are those of the generated code, the total including the runtime library; the 64.3 KB byte table is held by the caller and not included. RAM is the activations plus the runtime’s own memory. MAC as counted by ST Edge AI. The Micro-NAS rows use the trained weights; the other two architectures use random weights, which change neither latency nor footprint. The unconstrained architecture does not fit the 96 KB of RAM of the F401RE.

Architecture	Precision	Board	Weights (KB)	Flash total (KB)	RAM (KB)	MAC	Latency (ms)
Micro-NAS 64-5-384	FP32	H7B3I-DK	200.0	210.0	70.9	3.36 M	76.99
Micro-NAS 64-5-384	FP32	F401RE	200.0	210.2	65.7	3.36 M	444.08
Micro-NAS 64-5-384	INT8	H7B3I-DK	53.2	77.7	49.4	3.37 M	31.19
Micro-NAS 64-5-384	INT8	F401RE	53.2	78.0	44.5	3.37 M	449.64
Binary-aware 96-2-384	FP32	H7B3I-DK	245.6	254.6	84.9	3.31 M	84.14
Binary-aware 96-2-384	FP32	F401RE	245.6	254.8	65.7	3.31 M	397.22
Binary-aware 96-2-384	INT8	H7B3I-DK	63.9	84.7	67.6	3.34 M	58.57
Binary-aware 96-2-384	INT8	F401RE	63.9	84.8	59.3	3.34 M	431.30
NAS 256-4-384	FP32	H7B3I-DK	1494.5	1504.2	180.7	36.35 M	837.37
NAS 256-4-384	INT8	H7B3I-DK	381.5	411.3	188.5	36.41 M	649.41

titles drawn from five scripts, and scores 0.607 AUC in English and 0.592 over the fourteen translations.

Encoding a title, from its bytes to its vector, takes 0.53 ms on one core (median over 2,000 calls) and 0.48 ms with two threads. The full-precision file takes 0.59 ms on one core and 0.46 ms with two threads, so at this model size eight bits buy no speed on the phone, only a smaller file.

Energy was read from the phone’s battery gauge with the device unplugged and the screen on, over fifteen minutes with the encoder idle and fifteen minutes encoding in a loop at 1,806 titles per second. The phone drew 2.86 W while encoding and 1.15 W when idle, which corresponds to about 0.95 mJ per title. The gauge reports only an averaged current and a charge counter that moves in one-percent steps; these two coarser cross-checks give 0.83 and 1.6 mJ, so the result is best read as roughly one millijoule per title.

The measurements also test the cost model of Section 3.9. ST Edge AI counts 3.37 million multiply-accumulates for the eight-bit constrained encoder, against the 3.29 million of the per-layer counter, which leaves out biases, batch normalisation and pooling. The RAM rule, which counts only the buffers of the widest block, gave

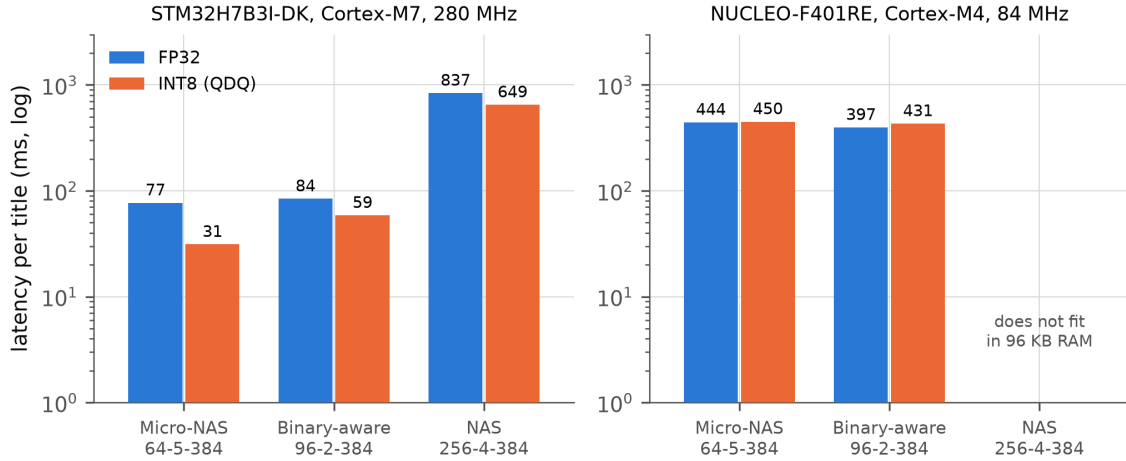


Figure 5.9: Latency per title on the two boards, full precision against eight-bit, for the three June architectures (Table 5.7). Logarithmic scale.

16 KB, whereas the generated code needs 49.4 KB; as Section 3.9 warned, the rule underestimates working memory, here by a factor of three. The flash estimate of 203.9 KB kept the first projection and the head at 32 bits; the eight-bit file quantises them as well, and the measured 142.0 KB is below the estimate. No board energy was measured, because the Developer Cloud reports only time and memory, so the energy column of Table 5.5 remains a proxy; the phone measurement above is the only energy figure measured on a target device.

5.9 Searches Under the Board Budgets

Table 5.8 gives the outcome of the board-specific searches of Section 3.7. Of the 150 candidates of each search, 110 fitted the STM32H7B3I-DK budget and 67 the NUCLEO-F401RE budget; among the hand-designed shapes, the best within the H7 budget is 64-2-384 and the best within the F401 budget 32-5-384. The recipe choices are, within the H7 budget, three convolutions (98, 26 and 94 channels, with one max-pooling) at 2.00 million multiply-accumulates and, within the F401 budget, a strided depthwise and a strided five-wide convolution at 0.65 million. Both lead the best hand-designed shape of their budget in teacher cosine under the search recipe (0.331 against 0.303 and 0.298 against 0.296), but in ranking they only match it: 0.595 against 0.599 AUC within the H7 budget and 0.591 against 0.589 within the F401 budget, differences smaller than the spread over seeds.

Full distillation reorders the eight finalists of each search (Table B.4 in Appendix B): the recipe choices fall to fifth and fourth place, and the final choices are candidate 140 for the H7 (1.93 million multiply-accumulates) and candidate 142 for the F401 (0.84 million, with one residual branch). These reach 0.605 and 0.608 AUC, ahead of the hand-designed shapes by 0.006, which is inside the spread over seeds, and by 0.019. All in-budget encoders stay below 64-5-384, which fits neither operation budget and

Table 5.8: Encoders under the budgets of the two boards. Recipe choice: best of the eight retrained finalists under the short search recipe; final choice: best of the same eight after full distillation. MMAC, weights (eight-bit) and peak activations from the search’s resource model. Search cos: validation cosine to the teacher under the search recipe, mean of three seeds ($\ddagger$: one seed). Teacher cos: English development titles after distillation on all training rows. AUC: MINDsmall dev after English clicks, at full precision and as the eight-bit file, mean $\pm$ standard deviation over seeds 42, 12 and 1. Last two columns: milliseconds per title of the eight-bit file on the STM32H7B3I-DK and the NUCLEO-F401RE. All rows read the bytes through the same fixed table with a padding-mask channel and average over all positions.

Encoder	MMAC	Weights (KB)	Act. (KB)	Search cos	Teacher cos	AUC	8-bit file AUC	H7 (ms)	F401 (ms)
<i>STM32H7B3I-DK budget: 256 KB weights, 128 KB activations, 2M MACs</i>									
μ NAS 144, recipe choice	2.00	51.9	15.5	0.331	0.392 $\pm$ 0.001	0.595 $\pm$ 0.003	0.596 $\pm$ 0.003	18.7	252.3
μ NAS 140, final choice	1.93	65.8	14.0	0.318	0.398 $\pm$ 0.002	0.605 $\pm$ 0.010	0.602 $\pm$ 0.009	28.4	226.6
Hand-designed 64-2-384	1.66	37.1	16.0	0.303 $\ddagger$	0.371 $\pm$ 0.001	0.599 $\pm$ 0.005	0.598 $\pm$ 0.006	16.6	191.7
<i>NUCLEO-F401RE budget: 128 KB weights, 48 KB activations, 1M MACs</i>									
μ NAS 134, recipe choice	0.65	41.5	5.9	0.298	0.349 $\pm$ 0.002	0.591 $\pm$ 0.003	0.590 $\pm$ 0.003	15.1	32.1
μ NAS 142, final choice	0.84	60.0	6.5	0.287	0.367 $\pm$ 0.001	0.608 $\pm$ 0.007	0.605 $\pm$ 0.006	12.1	60.6
Hand-designed 32-2-384	1.00	20.2	8.0	0.296 $\ddagger$	0.333 $\pm$ 0.019	0.589 $\pm$ 0.008	0.589 $\pm$ 0.003	21.2	88.7
<i>Outside both MAC budgets</i>									
Hand-designed 64-5-384	3.31	50.1	16.0	0.343	0.459 $\pm$ 0.000	0.618 $\pm$ 0.006	0.614 $\pm$ 0.010	31.5	426.8

scores 0.618 in this form and 0.629 in its original one. Under the search recipe the H7 recipe choice was 0.012 below 64-5-384 in teacher cosine; after full distillation the gap is 0.067, so a short distillation is a weak predictor of how far a network gets with full training. Across the fourteen translations the final choices score 0.553 and 0.552 after English clicks, against 0.557 for the reference encoder. Trained on clicks shown in a random language, the two recipe choices read the translations at 0.562 and 0.556 and English at 0.571 and 0.549.

On the NUCLEO-F401RE the F401 recipe choice is the fastest of the seven encoders: 32.1ms per title, against 88.7ms for 32-5-384 and 426.8ms for 64-5-384. It has a third fewer multiply-accumulates than 32-5-384 but runs 2.8 times faster, at 4.1 cycles per multiply-accumulate against 7.2; in 32-5-384 the depthwise layers take 22 cycles per multiply-accumulate and the first pointwise layer, over 65 input channels, 8.8. The final F401 choice, larger and more accurate, takes 60.6ms there, 1.5 times less than 32-5-384, and is the fastest of the seven on the STM32H7B3I-DK (12.1ms). Within the H7 budget, the hand-designed 64-2-384 is faster on that board (16.6ms) than both searched choices (18.7 and 28.4ms). All in-budget encoders keep their ranking accuracy as eight-bit files, losing at most 0.003. Quantising every operator, instead of only the convolutions and the output layer, kept the accuracy but made no file faster; on the H7 the code generator then fused one convolution with its max-pooling into an integer kernel that took 20.2ms alone. For the eight-bit files of the seven encoders, the on-target check of the generated code gave zero error except for 32-5-384 (0.008) and candidate 142 (0.019), the only one of the seven with a residual branch.

5.10 In-App Ranking

Besides running on the phone, the deployed encoder now contributes to the ranking of the feed. FEEDWELL-EDGE orders articles by a rule that it learns on the device: a weight per topic that grows with every opened or read article, together with terms for drift, freshness and unread items. To test what the encoder adds to this rule, the rule was ported to Python and checked against the application on 600 events and 200 articles, on which the scores match. The MINDsmall clicks were then replayed through it in time order, with each impression scored before the rule learns from its clicks; the MIND category stands in for the topic and an article's first appearance for its publication time. The content score of a candidate is the dot product of its vector, computed with the application's eight-bit file, and the user vector that the same checkpoint's user encoder pools from the last fifty clicked titles. It enters the rule standardised within the impression and multiplied by a weight. The weight was chosen on the earlier half of the development impressions, and the results are reported on the later half (36,576 impressions), so the weight is not fitted to the reported numbers.

On the held-out half the rule alone scores 0.600 AUC and the content score alone 0.594. Added together with the chosen weight of four, they score 0.624, which is 0.024 above the rule, with a 95% interval of 0.021 to 0.027 over 1,000 resamples of the 28,445 users. With the finer subcategory as the topic, the rule scores 0.606 and the sum, at a weight of two, 0.632, a gain of 0.026 (95% interval 0.023 to 0.029). The gain is small but consistent across both topic definitions, and it holds on the time-ordered split, which is the setting closest to how the application would use it. On this basis the application now ranks with the combined score by default, with a weight of 2, the best weight on the earlier half averaged over both topic definitions. The replay is an offline study on MINDsmall, with the MIND category as a proxy for the reader's own topics, and not a live user trial. MIND records clicks only, without the reading time and scroll depth from which the application also learns, and without publication times, and at the time of writing no reader had used the content-aware ranking; a controlled comparison on real reading behaviour is left to future work (Section 6.2). The replay shows that the byte-level encoder, trained and deployed under the budget of this thesis, carries ranking information that the transparent on-device rule does not already capture.

5.11 Discussion

Table 5.9 summarises the answers to the five research questions. The byte-level representation removes the memory wall as intended: a model that fits a microcontroller budget reaches the quality of the reduced word-vocabulary model, stays within 0.036 to 0.050 AUC of the GloVe model and serves many languages with one set of weights. On the remaining arithmetic, the compression techniques developed for vision transfer

to different degrees. Architecture search under a flash budget costs little quality at full precision, and a search under the full budget of a board finds encoders that match or beat the best hand-designed shapes of that budget once the choice is made after full training. Eight-bit weights have no measurable cost. Binarisation is the limit: naive one-bit weights lose 0.029 to 0.065 AUC, and recovery training wins back only part of this, to a level still below eight bits.

Table 5.9: The research questions and what the results answer.

RQ	Theme	Finding
RQ1	Representation	Yes, partly: the distilled 68k-parameter encoder, frozen, ranks fourteen translations at 0.572 against 0.573 in English; trained on mixed-language clicks it reaches 0.589 across languages (0.538 to 0.609) for 0.609 in English, within 0.001 of the whole teacher through a linear map.
RQ2	Architecture	The June flash and RAM budget costs 0.009 AUC at full precision (0.628 to 0.619, inside the spread over seeds) and 0.024 with eight-bit weights, for a model 5.9 times smaller with 11 times fewer operations; the operation bound was not enforced in the June search. Searched with μ NAS under the full budget of each board and chosen after full training, the encoders reach 0.605 (H7) and 0.608 (F401), 0.006 and 0.019 above the best hand-designed shapes of the same budgets.
RQ3	Precision	Eight-bit inner weights change AUC by -0.005 to $+0.011$, inside the spread over seeds, at 1.2 to 2 times less storage and 3.4 to 9.3 times less energy proxy; one-bit weights lose 0.029 to 0.065 against full precision. The eight-bit file of the distilled encoder loses 0.004 on average over three seeds and runs 2.5 times faster than full precision on the STM32H7, in 31.19 ms per title.
RQ4	Recovery	Naive binarisation costs the constrained model 0.065 (0.554); Re-ActNet blocks started from distilled weights reach 0.587 ± 0.012 at 194 KB, still 0.028 below eight bits on the same architecture.
RQ5	Reference	GloVe NRMS 0.664, reduced NRMS 0.604, subcategory histogram 0.617; the distilled byte-level encoder scores 0.629 with 165 times fewer parameters than GloVe NRMS, and distillation adds 0.011 under the seeded loop.

Three types of threat to validity limit these conclusions. Threats to internal validity come from the small number of seeds and from the June search. Every condition was run with three seeds, which put the spread at up to 0.017, and every smaller difference is treated as unresolved; the effects claimed from the reference conditions (distillation, mixed-language clicks, the NRMS configurations) are larger than the standard deviations in Table 5.1. The June search that chose the three matrix architectures was itself a single, unseeded run, and the board-specific searches scored their candidates with a short distillation whose ranking did not survive full training. Threats to external validity concern scale and language. The numbers are small-split MIND figures and would be expected to shift, though not to change order, on the large split. The

cross-lingual results come from translated impressions of English readers and not from native behaviour data, so they measure transfer in the content encoder and not multilingual personalisation, and the translations are machine-made. Threats to construct validity concern what is measured. In the matrix, quantisation is simulated on the weights and latency is a laptop timing; the eight-bit files and the board and phone runs of Sections 5.8 and 5.9 correct both, but only for the deployed encoders. Energy is a per-operation proxy everywhere except on the phone, and the in-app gain comes from an offline replay with MIND categories as topics, not from live use. None of these threats undermines the comparative claims, that is, the ordering of arms and precisions, the position of the encoder among its reference points and the one-bit limit, but each limits how far the absolute value of the corresponding number can be trusted. Figure 5.10 summarises the chapter.

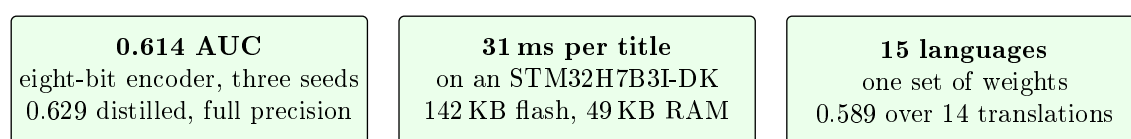


Figure 5.10: The headline results. The AUC figures are three-seed means (eight-bit weights, simulated; distilled encoder at full precision); the board figure is the measured eight-bit file of the distilled encoder (Section 5.8); the multilingual figure is the mixed-click condition.

5.12 Limitations

Several limitations follow from the scope set in Chapter 1. The memory wall is removed for the input representation, but the comparison baselines still have it: this work does not make a word-vocabulary model smaller but replaces it with a byte-level encoder. The encoder is bundled into the reader and its content score is part of the ranking, but the evidence for that step is an offline replay of MINDsmall through the application’s rule (Section 5.10), with the MIND category as a proxy for the reader’s topics; a live, on-device comparison of the two rankings on real reading behaviour has not yet been made. The boards ran the three June architectures at eight bits and full precision and the seven encoders of the board-specific searches as eight-bit files, and the phone ran the deployed constrained encoder, but the nine-variant matrix itself remains simulated, and energy on the boards was not measured. The June search enforced flash and RAM but not operations. The board-specific searches enforce all three limits, but they score a candidate by a short distillation, whose ranking did not survive full training: the choice made after full distillation of the eight finalists changed both picks, and a longer fitness inside the search would cost more search time. The one-bit model has no runtime on the target boards yet and would need validation on hardware before any deployment claim. A cross-lingual loss remains: even after mixed-language training, the translations stay 0.020 below English and 0.039 below the English-only encoder, and three languages remain below 0.56. Finally, a byte sequence is about five times

longer than the corresponding word sequence, which trades flash for computation, and all experiments use the small MIND split. Chapter 6 takes up the most important of these limitations as future work.

Chapter 6

Conclusion

6.1 Summary

This thesis set out to build a multilingual news recommender that fits microcontroller-class memory, stays close to the ranking quality of a much larger word-vocabulary model and covers many languages with one set of weights, and, once the embedding table was removed, to measure the trade-offs produced by architecture search and precision reduction. The order of the steps mattered: the change of representation had to come first, and the TinyML techniques could be applied to the remaining network only afterwards.

Main result

Reading titles as raw UTF-8 bytes fixes the vocabulary at 256 symbols shared by every language and removes the per-language table, whose cost would otherwise grow with the number of languages. With this representation, a distilled byte-level encoder, a flash-constrained search and eight-bit weights compress the recommender to an estimated 204 KB at the quality of a 27 MB word-vocabulary model and within 0.050 AUC of a 43 MB GloVe model. As an eight-bit file it runs in 31.19 ms per title on an STM32H7 board with 142.0 KB of flash, byte table included, and 49.4 KB of RAM, and on the host phone it encodes a title in 0.53 ms for about one millijoule. Searched with μ NAS under the budget of a NUCLEO-F401RE and chosen after full training, an encoder of 0.84 million multiply-accumulates reaches 0.608 AUC at 60.6 ms per title on that board. For this task, one-bit weights do not give a usable gain.

Measured against the objectives and research questions of Chapter 1, the work makes the following contributions. It reformulates the problem so that model size no longer depends on the number of languages, and it provides a distilled byte-level multilingual encoder of 68 thousand parameters. It reports a search-and-precision study of nine variants, each trained with three seeds, on one corrected cost model and places the encoder among reference points on both sides: the eight-bit constrained encoder reaches 0.614 AUC in 204 KB, the distilled full-precision encoder 0.629, GloVe NRMS 0.664 and the subcategory histogram 0.617, while binarisation is reported as a bounded negative result at 0.587. A second search with μ NAS, under the full budget of each of two STM32 boards, gives encoders of 0.605 and 0.608 AUC, 0.006 and 0.019 above the best hand-designed shapes of the same budgets. The eight-bit files were measured on the two boards, with 0.622 AUC and 31.19 ms per title on the Cortex-M7

for the reference encoder, and on the host phone. Multilingual transfer ranges from 0.528 to 0.589 over fourteen languages depending on how the encoder is trained, and the three languages it serves worst are identified. Finally, the host application, rebuilt on FEEDWELL 1.18.2, records reading events on the device and now ranks with the encoder’s content score added to its heuristic, a combination that gains 0.024 AUC in an offline replay. Table 5.9 restates the answers to the five questions.

6.2 Future Work

Future work follows a staged path from the results of this thesis to a fully measured and evaluated on-device system (Figure 6.1).

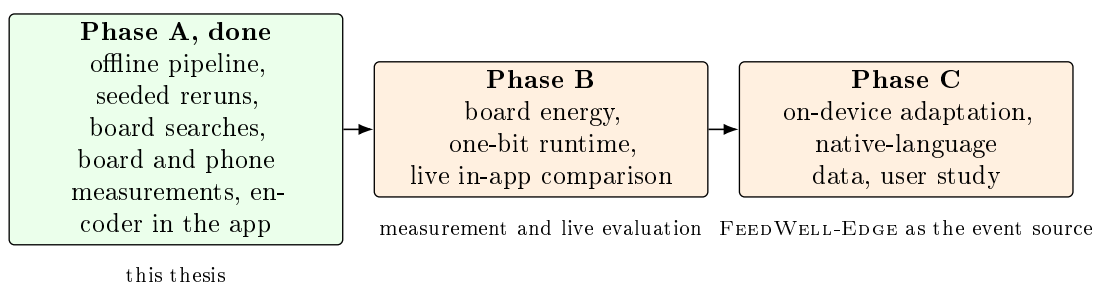


Figure 6.1: Staged path from the results of this thesis to a fully measured and evaluated system.

6.2.1 Remaining measurements

Two measurements remain open: energy on the boards and a runtime for the one-bit model. For the deployed files, the board and phone measurements of Section 5.8 replace the simulated quantisation and the laptop timing, and the phone’s battery gauge replaces the per-operation energy estimate on one target device. On the boards, energy is still a per-operation estimate. Measuring it requires the laboratory’s power probe on the STM32H7B3I-DK, which is a short measurement once the board is instrumented. The one-bit model would have to be rebuilt in a framework that the Larq Compute Engine [27] or CBin-NN [23] accepts, and this should be attempted only if the eight-bit deployment leaves flash to spare.

6.2.2 Search fitness

The nine-variant matrix has been rerun with three seeds, and the board-specific searches enforce all three limits with the corrected operation count. The fitness of these searches is still an open problem. A short distillation ranked the finalists differently from full training, and the choice made after full distillation changed the pick in both searches. A longer fitness inside the search, or a full-distillation stage for more than eight finalists, would make the choice more reliable at the cost of search time.

6.2.3 Live application evaluation

The eight-bit encoder is bundled into the host application, and its content score is added to the ranking through the engine’s storage schema, reusing the existing event pipeline, replay cursor, drift detection and telemetry (Section 4.6.1). An offline replay of the MINDsmall clicks through the application’s own rule gives a first measurement of what the encoder adds: the content score raises the heuristic from 0.600 to 0.624 AUC with the category as topic and from 0.606 to 0.632 with the subcategory, and the application now ranks with the combined score by default, with a weight of 2. No reader has used it yet. The next step is a live comparison of the two rankings on the device, based on real reading behaviour instead of replayed impressions.

6.2.4 On-device adaptation

With the encoder in place, a further step is on-device adaptation: fine-tuning a small head on top of the frozen byte-level embeddings with the locally recorded interaction events, so that personalisation is learned on the device instead of being applied through a fixed heuristic. The event-capture engine already produces the signals that such a scheme would use.

6.2.5 Multilingual coverage

The three weakest languages suggest two concrete improvements. The first is to distil the student against multilingual anchors, that is, against the teacher’s embedding of each translated title as well as the English anchor. The second is to test a longer byte window for three-byte scripts. Both could raise the weakest languages without enlarging the table, since the byte alphabet is already shared. In addition, native-language behaviour data and a wider set of languages would allow multilingual personalisation to be measured directly, instead of through translated impressions.

6.2.6 User study

The offline ranking metrics do not measure whether readers find the recommendations useful; that requires an evaluation with real users. A controlled user study on the integrated application, comparing the combined ranking with the heuristic alone, would connect the offline evidence of this thesis with the practical claim that the system makes.

6.3 Closing Remarks

In this work, the main obstacle to on-device multilingual news recommendation was the vocabulary table. Removing it by changing the input representation, and only then applying architecture search and quantisation, gives a model that fits a microcontroller budget and matches the quality of a word model more than a hundred times its size. For this task, one-bit precision did not give a usable gain. A search under the budget

of the smaller board finds an encoder that runs there in about 60 ms per title. The model runs on the boards and on the phone, where it encodes a title in about half a millisecond for roughly one millijoule, and its content score already improves the application's ranking in an offline replay. The next steps are to measure its energy on the boards and to evaluate the combined ranking with live reading behaviour.

Appendix A

Configuration

The experiments are controlled by one configuration file, which fixes the hyperparameters and the search space and is reproduced verbatim below from the MIND-EDGE-RECOMMENDER repository, and by the overrides that the two experimental programmes apply on top of it (Table A.1). Every value used in Chapters 4 and 5 comes from this file or from that table.

Listing A.1: The configuration file `config.yaml` of the MIND-EDGE-RECOMMENDER repository.

```
# =====
# FeedWell-Edge :: central experiment configuration
# Every knob lives here and is overridable from the notebook, e.g.:
#   from src.config import load_config
#   cfg = load_config(); cfg["train"]["epochs"] = 5
# =====

seed: 42

paths:
  data_dir: data          # downloaded datasets
  artifacts_dir: artifacts # run outputs: checkpoints, exports, scratch tables (not versioned)
  cache_dir: artifacts/cache # caches independent of the run protocol (teacher anchors)
  results_dir: paper/results # frozen result files quoted by the paper (versioned)
  logs_dir: logs

# -----
# Datasets
# -----
data:
  mind_size: small      # small | large (start small for fast iteration)
  # All 14 xMIND languages, using the HF on-disk codes (HF uses 'zho' for
  # Mandarin where the GitHub download.py uses 'cmn'). Files live at
  # data/xmind/<size>/data/<lang>/<split>.parquet.gz
  xmind_langs: [zho, fin, grn, hat, ind, jpn, kat, ron, som, swi, tam, tha, tur, vie]
  max_history: 50        # clicked-news kept per user history
  max_title_bytes: 128    # byte-level student input length (UTF-8 bytes)
  neg_ratio: 4           # negative samples per positive (NRMS-style training)

# -----
# Frozen multilingual TEACHER (runs offline only, never shipped on-device)
# -----
teacher:
  model: sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2
  embed_dim: 384
  batch_size: 256

# -----
# Deployed STUDENT content encoder (byte-level char-CNN)
# -----
student:
  type: byte_cnn          # byte_cnn | hashed_ngram (fallback)
```

```

byte_embed_dim: 64      # 256-symbol UTF-8 alphabet -> shared by all 14 langs
channels: 128
depth: 3                # depthwise-separable 1D conv blocks
out_dim: 384            # distillation target dim (matches teacher)
matryoshka_dim: 64      # truncated dim usable on-device

# -----
# NAS (architecture axis: nas -> micro_nas -> binarized_micro_nas)
# -----
nas:
  method: evolutionary  # evolutionary supernet search (NAS-BNN style)
  search_space:
    channels: [32, 64, 96, 128, 192, 256]
    depth: [1, 2, 3, 4, 5]
    out_dim: [64, 128, 256, 384]
  population: 50
  generations: 20
  # Hard constraints for the Micro-NAS arms. Deliberately TIGHTER than the
  # STM32H7 ceiling (2 MB / 1 MB) so the constraints actually bind at this tiny
  # text-encoder scale and the architecture axis (NAS vs Micro-NAS) separates.
  constraints:
    flash_kb: 256        # aggressive on-device budget
    ram_kb: 128
    max_macs: 2000000    # 2 MMac latency proxy

# -----
# Precision axis
# -----
quant:
  int8: {method: qat}    # ptq | qat
  binary: {keep_first_last_fp: true} # XNOR-Net practice: don't binarize first/last

# -----
# Training
# -----
train:
  device: cuda           # falls back to cpu automatically
  epochs: 10
  batch_size: 64
  lr: 0.0001
  distill_epochs: 15
  distill_lr: 0.001

# -----
# Evaluation
# -----
eval:
  metrics: [auc, mrr, ndcg@5, ndcg@10]
  langs: all             # evaluate all 14 xMIND langs (cross-lingual transfer)

# -----
# Deployment targets (for the footprint/energy matrix)
# -----
targets:
  laptop: {flash_kb: null, ram_kb: null, energy: nvml}
  pi5:    {flash_kb: null, ram_kb: null, energy: ina219}
  stm32h7: {flash_kb: 2048, ram_kb: 1024, energy: analytical}

# Per-operation energy in pJ (relative proxy; label as such).
# fp32_mac and int8_mac are multiply + add from Horowitz, ISSCC 2014 (45 nm):
# 3.7 + 0.9 and 0.2 + 0.03. Horowitz gives no 1-bit figure; binary_op follows the
# operation-count convention of XNOR-Net / Bi-Real Net (64 binary operations per
# 32-bit one), i.e. fp32_mac / 64. It is a convention, not a measurement.

```

```
energy_pj:  
  fp32_mac: 4.6  
  int8_mac: 0.23  
  binary_op: 0.0719  
  sram_read_32b: 5.0
```

Table A.1: Overrides applied by the run scripts on top of the configuration file.

Script	Settings that differ from the file
<code>scripts/run_full.py</code> (June 2026)	ranking epochs 8 (file: 10); distillation epochs 15; search with 15 generations and a population of 30 (file: 20 and 50), each candidate scored by a two-epoch distillation on 20,000 titles with 4,000 for validation; two quantisation-aware epochs after the ranking epochs for the eight-bit and one-bit cells; NRMS baseline 8 epochs; negatives drawn once per impression; initialisation not seeded
distillation ablation (June 2026)	10 ranking epochs, from a random start and from the distilled weights
recovered binary model (June 2026)	ReActNet blocks with Bi-Real shortcuts on the 64-5-384 architecture, initialised from the distilled encoder
<code>scripts/run_p1.py</code> (October 2026)	ranking epochs 8; distillation epochs 15 with learning rate 10^{-3} ; negatives redrawn at every step from a generator seeded with the run seed; seeds 42, 12 and 1; teacher conditions 8 epochs; NRMS in the GloVe reference configuration (300-d table, 20 heads of 20, attention 200, dropout 0.2) and the reduced configuration (256-d table, 8 heads, attention 128)
<code>scripts/run_matrix.py</code> (October 2026)	the three June architectures from a random start on English clicks; ranking epochs 8; two quantisation-aware epochs for the eight-bit and one-bit cells; negatives redrawn at every step; seeds 42, 1 and 2
<code>scripts/run_binary.py</code> (October 2026)	ReActNet blocks with Bi-Real shortcuts on 64-5-384; distillation 15 epochs, then 8 epochs of English clicks; seeds 42, 1 and 2
<code>scripts/run_unas_full.py</code> (October 2026)	board-search encoders and comparators: distillation 15 epochs on all training rows in 15 languages, then 8 epochs of English or mixed-language clicks; seeds 42, 12 and 1
μ NAS searches (<code>unas/</code> , October 2026)	150 candidates, population 40, sample 12; fitness: distillation on 96,000 rows in 15 languages, at most 15 epochs; budgets of 256 KB of weights, 128 KB of activations and 2 M multiply-accumulates (STM32H7B3I-DK), and 128 KB, 48 KB and 1 M (NUCLEO-F401RE)

Appendix B

Complete Results

This appendix collects the full figures behind Chapter 5: the per-seed values of the October 2026 matrix rerun next to the June single runs, the June ablations and the one-bit rerun, the June single-run per-language evaluation, the finalists of the board-specific searches, and the per-seed values and durations of the October reference conditions. All ranking metrics are measured on the 73,152 labelled MINDsmall development impressions.

B.1 Matrix rerun

Table B.1: AUC of the nine variants per seed in the October 2026 rerun (seeds 42, 1 and 2), with the mean and population standard deviation, next to the single June 2026 run of the same cell. Costs are those of Table 5.2.

Arm	Prec.	seed 42	seed 1	seed 2	mean $\pm$ SD	June run
NAS	FP32	0.6122	0.6475	0.6238	0.6278 $\pm$ 0.0147	0.6327
NAS	INT8	0.6320	0.6420	0.6416	0.6385 $\pm$ 0.0046	0.6471
NAS	1-bit	0.6088	0.5792	0.6072	0.5984 $\pm$ 0.0136	0.5878
Micro-NAS	FP32	0.6203	0.6181	0.6187	0.6190 $\pm$ 0.0009	0.6045
Micro-NAS	INT8	0.6219	0.6073	0.6136	0.6143 $\pm$ 0.0060	0.6100
Micro-NAS	1-bit	0.5319	0.5573	0.5734	0.5542 $\pm$ 0.0171	0.5210
Bin. μ NAS	FP32	0.6046	0.5977	0.6028	0.6017 $\pm$ 0.0029	0.5934
Bin. μ NAS	INT8	0.6085	0.5875	0.6061	0.6007 $\pm$ 0.0094	0.6009
Bin. μ NAS	1-bit	0.5666	0.5564	0.5497	0.5576 $\pm$ 0.0069	0.5460

The 27 runs of the rerun took 337.5 minutes in total. Laptop latency per architecture, for the full-precision models: ONNX Runtime on the CPU 0.3405, 0.1856 and 0.0992 ms per title for the NAS, Micro-NAS and binary-aware architectures; PyTorch on the GPU 0.5721, 0.6508 and 0.4333 ms.

B.2 June ablations and one-bit rerun

The recovered binary model has 109,248 parameters, of which 61,440 are one-bit weights in five full binary convolutions of kernel three and 47,808 are full-precision parameters (byte embedding, first projection, head, batch normalisation, RSign and RPreLU); it executes 8,413,184 operations per title, 7,864,320 of them binary, for an estimated 194.25 KB, 16 KB of working memory and 3.09 μ J. Rerun with three seeds

Table B.2: June 2026 ablations, single runs. The distillation ablation trained the 64-5-384 encoder for ten epochs; the NRMS comparison quantised every weight of the reduced model to eight bits after eight epochs.

Experiment	Setting	AUC	MRR	nDCG@5	nDCG@10
Distillation ablation	random start	0.5999	0.2933	0.2728	0.3426
Distillation ablation	distilled initialisation	0.6332	0.3449	0.3311	0.3903
NRMS, reduced	FP32 (27.0 MB)	0.6045	0.3243	0.3049	0.3672
NRMS, reduced	INT8, all weights (6.8 MB)	0.6047	0.3244	0.3049	0.3672
Recovered binary, 64-5-384	ReActNet blocks, distilled start	0.5723	0.2910	0.2723	0.3365

under the October loop, it scores 0.5732, 0.5848 and 0.6014 (seeds 42, 1 and 2; mean 0.5865, standard deviation 0.0116), and its distillation reaches a teacher cosine of 0.351, 0.342 and 0.350. The fifteen-epoch distilled student of the October programme reaches a mean cosine of 0.477, 0.477, 0.474 and 0.473 with the teacher’s anchors on the development titles at prefix dimensions 64, 128, 256 and 384.

B.3 June languages

Table B.3: Per-language metrics of the constrained full-precision encoder (64-5-384, random start, English clicks), June 2026 programme, single run. The October three-seed mean of the same condition is 0.528 (Table 5.3).

Code	Language	AUC	MRR	nDCG@5	nDCG@10
en	English	0.605	0.307	0.286	0.352
zho	Chinese	0.521	0.251	0.235	0.298
fin	Finnish	0.495	0.225	0.208	0.272
grn	Guarani	0.533	0.259	0.247	0.306
hat	Haitian Creole	0.556	0.263	0.250	0.312
ind	Indonesian	0.537	0.293	0.272	0.329
jpn	Japanese	0.551	0.259	0.245	0.309
kat	Georgian	0.491	0.264	0.235	0.296
ron	Romanian	0.532	0.254	0.237	0.300
som	Somali	0.514	0.235	0.215	0.281
swl	Swahili	0.521	0.243	0.230	0.289
tam	Tamil	0.504	0.244	0.229	0.288
tha	Thai	0.504	0.237	0.216	0.281
tur	Turkish	0.530	0.267	0.254	0.311
vie	Vietnamese	0.523	0.254	0.240	0.298
mean of 14		0.522	0.253	0.236	0.298

B.4 Board-search finalists

Table B.4: The eight finalists of each board-specific search: cosine to the teacher on the search’s validation rows under the short search recipe (mean of three seeds) and after full distillation (seed 42). The final choice is the best after full distillation over three seeds (STM32H7B3I-DK: 0.3815; NUCLEO-F401RE: 0.3451).

Board	Candidate	MMAC	Cosine to the teacher	
			search recipe	full distillation
H7B3I-DK	140 (final choice)	1.93	0.318	0.382
	108	1.85	0.317	0.374
	126	1.78	0.315	0.374
	112	1.74	0.318	0.368
	144 (recipe choice)	2.00	0.331	0.367
	142	1.78	0.322	0.363
	129	1.73	0.321	0.360
	146	1.78	0.321	0.356
F401RE	142 (final choice)	0.84	0.287	0.345
	133	0.78	0.287	0.344
	101	0.42	0.283	0.334
	134 (recipe choice)	0.65	0.298	0.333
	135	0.86	0.298	0.331
	119	0.42	0.282	0.331
	100	0.43	0.281	0.319
	36	0.72	0.272	0.308

B.5 October reference conditions

Table B.5 gives the values per seed for every reference condition of the October 2026 programme, and Table B.6 the wall-clock duration of each run.

Table B.5: English AUC and mean AUC over the fourteen translations for every October reference condition and seed. Means and standard deviations are in Table 5.1.

Condition	English AUC			mean AUC, 14 languages		
	seed 42	seed 12	seed 1	seed 42	seed 12	seed 1
Frozen teacher + user encoder	0.589	0.589	0.589	0.575	0.575	0.575
Frozen teacher + linear map	0.620	0.617	0.617	0.592	0.591	0.588
NRMS, GloVe reference config.	0.660	0.666	0.666	–	–	–
NRMS, reduced (random 256-d table)	0.594	0.619	0.600	–	–	–
Byte-CNN, random start, EN clicks	0.620	0.615	0.618	0.525	0.528	0.529
Byte-CNN, random start, mixed clicks	0.579	0.581	0.591	0.557	0.556	0.558
Byte-CNN, distilled, frozen	0.568	0.577	0.576	0.571	0.571	0.574
Byte-CNN, distilled start, EN clicks	0.628	0.625	0.632	0.557	0.560	0.554
Byte-CNN, distilled start, mixed clicks	0.614	0.598	0.614	0.596	0.587	0.585

Table B.6: Wall-clock minutes of every run of the October reference conditions on the measurement laptop. The 239.9-minute run is the one stretched by another process holding the GPU; the same condition took 20.6 minutes with the other two seeds.

Condition	seed 42	seed 12	seed 1	mean (min)
Frozen teacher + user encoder	7.2	3.9	5.1	5.4
Frozen teacher + linear map	5.1	3.4	4.8	4.4
NRMS, GloVe reference config.	23.9	22.1	24.2	23.4
NRMS, reduced (random 256-d table)	13.8	12.1	13.7	13.2
Byte-CNN, random start, EN clicks	17.4	16.8	16.8	17.0
Byte-CNN, random start, mixed clicks	20.6	239.9	20.6	93.7
Byte-CNN, distilled, frozen	7.2	7.7	7.7	7.5
Byte-CNN, distilled start, EN clicks	16.7	17.4	16.9	17.0
Byte-CNN, distilled start, mixed clicks	21.5	23.4	21.5	22.1
Heuristic scorers (one pass)	1.05	–	–	1.05
Total	612.5 min (10.2 h)			

Appendix C

FeedWell-Edge Engine

This appendix documents the on-device engine of FEEDWELL-EDGE version 2.1.0: its event schema, scoring, drift detection and telemetry. The engine is a heuristic personalisation system implemented in `src/edgectl/localLearningService.js` of the FEEDWELL-EDGE repository. It is separate from the neural encoder of Chapters 3 to 5, whose content score the application adds to the engine’s ranking (Section 5.10). The heuristic rules below are taken from that file at the commit given in Table D.1.

Storage and controls. The engine keeps four records in the application’s local key-value store (AsyncStorage): the event log, the active read sessions, the model state and a consumption cursor that marks how far the learner has processed the log. The event log is capped at 5,000 events; older events are dropped when the cap is reached, and a purge removes events older than the retention period chosen in the settings screen, which cycles through 30, 90 and 180 days (30 by default). The same screen holds the switch that enables on-device learning, a seven-day summary of the local events, the telemetry export and the deletion of all learning data. Nothing is transmitted.

Event schema. Six kinds of event are recorded, each with a timestamp and the article’s topic (its first category): an *impression* when an article is shown in a list; an *open* when it is opened; a *read session*, started on entering the reader and finished on leaving it, carrying the dwell time in seconds, the maximum scroll depth in percent and a completion flag; a *search* with its query; a *stage transition* between the list, preview and reader stages with the dwell on the stage left; and a *feature usage* record when translation, read-aloud or notes are used on an article. A read is counted complete when the scroll depth reaches 80% or the dwell reaches 45 seconds. Since the encoder was added, the snapshot of the article stored with each event keeps its title (at most 300 characters), and impressions and opens also record the content score and the content weight in force.

Learning step. The learner replays the events from the cursor and updates one additive weight per topic: an open adds 0.8; a read session adds $0.5 + \min(d/90, 1.2) + \min(s/100, 1.0)$, where d is the dwell in seconds and s the scroll depth in percent, plus 0.5 if the session was complete; other events carry no weight. Each topic weight is clamped to $[-20, 20]$. For every topic the learner also keeps two exponential moving averages of the increments, a short one with rate 0.3 and a long one with rate 0.05, their absolute difference as the divergence, the number of events seen and the time of

the last update. A topic is flagged as drifting when its divergence reaches 0.35 after at least five events; the largest divergence seen is kept in the state. The cursor then advances to the end of the log.

Ranking. An article's score is the sum of four terms: the weight of its topic; a drift boost of $\max(0.4, 3 \times \text{divergence})$ if the topic is flagged, else 0; a freshness boost of $\max(0, 1.5 - \min(h/48, 1.5))$, where h is the number of hours since publication (72 if the date is unknown), so that freshness decays to zero over 48 hours; and an unread boost of 0.35 if the article has not been read. The heuristic sorts the feed by this score, to which the application adds the encoder's content score.

Content-aware ranking. With content-aware ranking switched on, which is the default, the application computes a content score for every article in the list (`src/edgectl/contentRanker.js` and the local module `modules/edge-encoder`). The eight-bit encoder turns each title into a 384-dimensional vector, on one thread and with a cache of at most 5,000 vectors keyed by title, and the user encoder of the same checkpoint pools the vectors of the titles of the 50 most recently opened or read articles into one user vector. An article's content score is the dot product of the two vectors; the scores are standardised over the loaded articles, and the feed adds the standardised score, multiplied by the content weight, to the heuristic score. The settings offer content weights of 0.5, 1, 2 and 4, with 2 as the default, the value chosen in the replay of Section 5.10.

Telemetry and controls. A summary of the local state can be produced on request and exported by the user as a JSON snapshot. It contains the generation time and schema version, the retention policy (event cap, cursor position and number of unprocessed events), totals (event count, active sessions, time of the last model update and number of topics with a weight), counts by event type, daily counts over the last fourteen days and the serialised sizes of the log and the state. A separate function deletes all stored learning data. All processing is local and no event leaves the device.

Cold-start prior. The export stage of Section 4.3 writes a `topicWeights` object with one entry per MIND category, the normalised click frequency of that category in the training logs, in the same shape as the model state's topic weights, so that a client can start from the population prior before any personal event has been recorded.

Development data. During development the application was exercised on the test phone with demonstration feeds and sample interactions, which served only to check that the engine records, learns and re-ranks as described. No such data appear in any result of this thesis.

Appendix D

Reproducibility

Reproducing the results requires the repositories and their commits, the software environment, the deployment files and the data manifest given below.

D.1 Repositories

Three public repositories hold the artifacts, at the commits listed in Table D.1. The research pipeline, the result files, the figures and tables of this thesis and the diagram sources live in MIND-EDGE-RECOMMENDER; the application lives in FEEDWELL-EDGE, whose `source` remote is the public FEEDWELL repository it was rebuilt from.

Table D.1: Repositories and the commits that correspond to the reported results.

Repository	Content	Commit
https://github.com/SepehrMohammady/MIND-Edge-Recommender	pipeline, scripts, result files (<code>paper/results</code>), logbook, diagrams, thesis assets	4fa1e51 on <code>main</code> ; each run record carries the commit of its run (Section D.2)
https://github.com/SepehrMohammady/FeedWell-Edge	the application with the on-device engine (<code>src/edgeml/localLearningService.js</code>), the eight-bit encoder and content-aware ranking, version 2.1.0	76f8bf5 on branch <code>edge-encoder</code> , which starts from 2a7a7e6 on <code>rebuild-on-feedwell-1.18</code>
https://github.com/SepehrMohammady/FeedWell	the base application, version 1.18.2	38a8b67 on <code>main</code>

D.2 Environment

Experiments ran on the laptop of Table 4.3 under Windows 11 with Python 3.13.14 and PyTorch 2.12.1 built for CUDA 13, on an NVIDIA GeForce RTX 5070 Laptop GPU; the CPU timings used ONNX Runtime. The exact package versions are pinned in `requirements-lock.txt` in the repository. The μ NAS searches ran in a copy of the laboratory’s fork in the Windows Subsystem for Linux on the same laptop, with TensorFlow 2.21 and Keras 3.12. Every October run appended a record to `paper/results/experiments.jsonl` with the run name, the seed, the settings, the duration in minutes, the metrics and the environment (Python and PyTorch versions, GPU,

code commit and whether the working tree was clean). The 136 records of October 2026 (28 runs of the reference conditions, 27 of the matrix rerun, 3 of the one-bit rerun, 70 of the board-search encoders and 8 of the replay) carry the commit at which each run was made, from `f5b7a0a` (5 October) to `fd7d5e0` (9 October), and every one was made from a clean working tree. The dated `LOGBOOK.md` records what was run, when, for how long and what was found, including the GPU-sharing incident and the restart of 5 October 2026.

D.3 Board, phone and replay runs

The deployment files of Section 5.8 are produced by `scripts/deploy_boards.py` from the checkpoint `artifacts/runs/p1/distill_ft_en_seed42.pt` and written to `artifacts/stedgeai/models`; their accuracy on the development impressions is in `paper/results/deploy_int8.json`. The eight-bit files of the other seeds and of the board-search encoders are produced by `scripts/export_int8.py`, with their accuracy in `paper/results/int8_export.json`; the board-search encoders are summarised in `paper/results/unas_summary.json`, and the searches, the grid and the choice after full distillation in `paper/results/unas/`. The board runs go through `scripts/stedgeai_cloud.py`, which uses the Python client of ST’s model-zoo services against the ST Edge AI Developer Cloud (ST Edge AI Core 4.0.1); every run writes the server’s full report to `paper/results/stedgeai_cloud/` and one summary line to `runs.jsonl` in the same folder, from which the board tables of this thesis are generated. The raw files of the phone measurements are in `paper/results/phone/edge_bench/`, the energy per title in `paper/results/phone_energy.json`, the accuracy of the application’s encoder files in `paper/results/app_encoder.json`, and the replay of Section 5.10 in `paper/results/app_replay.json`.

D.4 Figures and tables

The scripts `scripts/make_figures.py`, `scripts/make_tables.py` and `scripts/make_thesis_assets.py` read only the frozen result files in `paper/results` and the downloaded data, and regenerate every figure and table of the paper draft and of this thesis; `scripts/check_numbers.py` checks the numbers quoted in the paper draft against the same files. The diagrams of Chapters 3 to 5 are TikZ sources in `paper/diagrams`, rendered to PNG at 300 dpi. The assets that rest on the October matrix rerun, the one-bit rerun and the board-specific searches (Figures 5.1, 5.3, 5.4 and 5.6; Tables 5.2, 5.8, B.1 and B.4) were regenerated for this version of the thesis from the result files at commit `4fa1e51`, in the plotting style of `scripts/make_thesis_assets.py`. The notebook in `notebooks/` re-executes the pipeline in a quick mode on the corrected code; its headline cell reads the frozen result files and not the output of its own quick run.

D.5 Data manifest

The datasets are MIND in its small configuration, with the utilities archive that holds the GloVe table and the word dictionary of the reference NRMS, and the fourteen-language xMIND release. The 31 downloaded files are pinned by SHA-256 in `docs/data_manifest.json`; the manifest is reproduced here.

File	SHA-256
<code>mind_small_dev.zip</code>	<code>d6ce515dcaa6b6d47ddf0a326eebc8a31b84735ae410285c9882ca2a06eec669</code>
<code>mind_small_train.zip</code>	<code>6ef97a271580b98ccfc4301ada55cc639423cb0576a78b8dcfcf74a4dbcc3194</code>
<code>mind_small_utils.zip</code>	<code>756cea1306c97fef62adf786814514f8c32c7b4955e093a3eea857f3286d2be</code>
<code>xmind/data/fin/dev.parquet.gzip</code>	<code>c0e6bad6af8a4f54a36b5edfac6367a273571a987d83a39d6a0385dbbfcee198</code>
<code>xmind/data/fin/train.parquet.gzip</code>	<code>483ed81bee86c215a696f601f59c43e38a607cc720d2d500e946773eb54946d1</code>
<code>xmind/data/grn/dev.parquet.gzip</code>	<code>42a7ec00e8b55b177ac04faff60c39076ef29c78c7e9d11836f032a8cfbb610c</code>
<code>xmind/data/grn/train.parquet.gzip</code>	<code>ad213233e94d60795f5910864003da2ea980a3c7319b584af4535fd9e09ee019</code>
<code>xmind/data/hat/dev.parquet.gzip</code>	<code>73396519e0a78933586401649a94bbc28eb1930ac196c5a5c9686ef49bd3d26e</code>
<code>xmind/data/hat/train.parquet.gzip</code>	<code>2639ee0b58df43f93d1a632a11663313f508d97ec0d80dbf5a2be7c5758d7e3a</code>
<code>xmind/data/ind/dev.parquet.gzip</code>	<code>8db6107f8acf3f61047cdb0a7052cd81df2839c970ce2fdb36098895a2cf98be</code>
<code>xmind/data/ind/train.parquet.gzip</code>	<code>96f4ce4135caa290f34c72dded911f62383f5cade7750b028fbaf6e98de5969fc</code>
<code>xmind/data/jpn/dev.parquet.gzip</code>	<code>aae47671302faf2aa37dd6d4ecf48c09c5ec981ecd3b5dfffb3ff088dc0529d0b</code>
<code>xmind/data/jpn/train.parquet.gzip</code>	<code>c34c606a181b5aab42fcbf494c7a796c398af45816142fb6ddd0ddfd4ec42cef</code>
<code>xmind/data/kat/dev.parquet.gzip</code>	<code>346e51776b3135285e67750a95c72ab869be95c7d42cd57261989f33dd656696</code>
<code>xmind/data/kat/train.parquet.gzip</code>	<code>5a5b42590f2613f17dc6f15ef6f6c27b77d19fb752877a851f45cc31dffe7291</code>
<code>xmind/data/ron/dev.parquet.gzip</code>	<code>7fa63e72d0f3d55c83f0e37b3b76291e573c21a5f4a85c871092b5db8ed06963</code>
<code>xmind/data/ron/train.parquet.gzip</code>	<code>eea64eee939a404a22104226c2031bc690a72a55f89fadfdbba2d68c3caf03b8</code>
<code>xmind/data/som/dev.parquet.gzip</code>	<code>d5539a6bb4d7e880126bb85cd0001d9f8668d23037fe570d9ad0be62d222e20a</code>

File	SHA-256
xmind/data/som/train.parquet.zip	112d85623882c1acec795ceeafd2659d c7ba30865b1bcba31f65225641bd4b30
xmind/data/swh/dev.parquet.zip	a80f704c22d9f25d955f98c6b4d4b397 5edfe37d47183b240d1ed7d792c1e577
xmind/data/swh/train.parquet.zip	9afa647a21721bb814f60a2fb204ce0f 00ffebd31c81a3d2f83368c875e586f0
xmind/data/tam/dev.parquet.zip	4d22065d3e34a4e80982cf4f91d62e29 0c540b449ec015a5454ebce337fbbf38
xmind/data/tam/train.parquet.zip	77ac0159b0409947dd9b01aacb7033e0 3a7956cc34fa841a5918a9a18a8f0c07
xmind/data/tha/dev.parquet.zip	4c9fe2d180949ba1421645a58a987389 a4f1a0f95fe0db68a03ea534a3643d65
xmind/data/tha/train.parquet.zip	747fa87a9fa9fd53f84b1e62eec9e57d eb13f949d8a5b8330a3d8ca9f1ffab75
xmind/data/tur/dev.parquet.zip	317c48bb6af8101d24f930323a8af2d5 e74aa1424511af446ac3ed5a89bf1b8e
xmind/data/tur/train.parquet.zip	7eb0f50ef65a38720721e7163ee7ea9b c6028a00b2de71c7eece3a28577cab2e
xmind/data/vie/dev.parquet.zip	6e421e19b9b2bc82e74c8222f0550d7b cd951e6abd68f7986b1df03bad394239
xmind/data/vie/train.parquet.zip	2fe1634286be321bb2cab99cd6dd0c1e 3df5705f80f42b9f6ae0e44a2e08553c
xmind/data/zho/dev.parquet.zip	f3ce6fef9e0f3212e7a625257af61123 9505287ff65a2a30f7dfce5c2ecff52
xmind/data/zho/train.parquet.zip	0c5abecaa19f746127919e09632b919f 6d6da72f40c073403c2af8e9507ed68a

Bibliography

- [1] European Parliament and Council of the European Union, “Regulation (EU) 2016/679 of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation).” Official Journal of the European Union, L 119, pp. 1–88, 2016.
- [2] L. Lazzaroni, A. Dabbous, M. Nardello, M. Abu Lebdeh, R. Berta, K. S. Yildirim, F. Bellotti, and D. Brunelli, “Green machine learning for the IoT (GEMINI): A position paper,” in *Applications in Electronics Pervading Industry, Environment and Society (ApplePies 2024)*, vol. 1369 of *Lecture Notes in Electrical Engineering*, Springer, 2025. ISBN 978-3-031-84099-9; University of Genoa IRIS handle 11567/1274017.
- [3] F. Wu *et al.*, “MIND: A large-scale dataset for news recommendation,” in *Proceedings of ACL*, pp. 3597–3606, 2020.
- [4] H. Wang, F. Zhang, X. Xie, and M. Guo, “DKN: Deep knowledge-aware network for news recommendation,” in *Proceedings of WWW*, 2018.
- [5] C. Wu, F. Wu, M. An, J. Huang, Y. Huang, and X. Xie, “Neural news recommendation with attentive multi-view learning,” in *Proceedings of IJCAI*, 2019.
- [6] C. Wu, F. Wu, M. An, J. Huang, Y. Huang, and X. Xie, “NPA: Neural news recommendation with personalized attention,” in *Proceedings of KDD*, 2019.
- [7] M. An, F. Wu, C. Wu, K. Zhang, Z. Liu, and X. Xie, “Neural news recommendation with long- and short-term user representations,” in *Proceedings of ACL*, 2019.
- [8] C. Wu, F. Wu, S. Ge, T. Qi, Y. Huang, and X. Xie, “Neural news recommendation with multi-head self-attention,” in *Proceedings of EMNLP-IJCNLP*, pp. 6389–6394, 2019.
- [9] C. Wu, F. Wu, T. Qi, and Y. Huang, “Empowering news recommendation with pre-trained language models,” in *Proceedings of SIGIR*, 2021. arXiv:2104.07413.
- [10] C. Wu, F. Wu, Y. Huang, and X. Xie, “Personalized news recommendation: Methods and challenges,” *ACM Transactions on Information Systems*, 2023. arXiv:2106.08934.
- [11] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global vectors for word representation,” in *Proceedings of EMNLP*, 2014.

- [12] Y. Yu, F. Wu, C. Wu, J. Yi, and Q. Liu, “Tiny-NewsRec: Effective and efficient plm-based news recommendation,” *Proceedings of EMNLP*, 2022. arXiv:2112.00944.
- [13] H. Yin, L. Qu, T. Chen, W. Yuan, R. Zheng, J. Long, X. Xia, Y. Shi, and C. Zhang, “On-device recommender systems: A comprehensive survey,” *Data Science and Engineering*, vol. 10, no. 4, pp. 591–620, 2025.
- [14] A. Iana, G. Glavaš, and H. Paulheim, “MIND your language: A multilingual dataset for cross-lingual news recommendation,” in *Proceedings of SIGIR*, 2024.
- [15] NLLB Team, “No language left behind: Scaling human-centered machine translation,” 2022. arXiv:2207.04672.
- [16] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “MCUNet: Tiny deep learning on IoT devices,” in *Proceedings of NeurIPS*, 2020.
- [17] E. Liberis, Ł. Dudziak, and N. D. Lane, “ μ NAS: Constrained neural architecture search for microcontrollers,” *Proceedings of EuroMLSys*, 2021. arXiv:2010.14246.
- [18] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, “MCUNetV2: Memory-efficient patch-based inference for tiny deep learning,” in *Proceedings of NeurIPS*, 2021.
- [19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” *arXiv:1704.04861*, 2017.
- [20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proceedings of CVPR*, pp. 4510–4520, 2018.
- [21] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in *Proceedings of ICML*, 2019.
- [22] A. Pighetti, F. Bellotti, S. Mohammady, L. Lazzaroni, H. Ballout, and R. Berta, “Exploring neural architecture search (NAS) for binary neural networks (BNNs) in the new WakeVision,” in *Applications in Electronics Pervading Industry, Environment and Society (ApplePies)*, vol. 1553 of *Lecture Notes in Electrical Engineering*, pp. 85–90, 2026.
- [23] F. Sakr, R. Berta, J. Doyle, A. Capello, A. Dabbous, L. Lazzaroni, and F. Bellotti, “CBin-NN: An inference engine for binarized neural networks,” *Electronics*, vol. 13, no. 9, p. 1624, 2024.
- [24] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Napier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, “TensorFlow Lite Micro:

- Embedded machine learning for TinyML systems,” in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 3, pp. 800–811, 2021.
- [25] L. Lai, N. Suda, and V. Chandra, “CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs,” *arXiv:1801.06601*, 2018.
- [26] STMicroelectronics, “ST Edge AI Developer Cloud and ST Edge AI Core documentation.” <https://stedgeai-dc.st.com>, 2026. accessed October 2026.
- [27] T. Bannink, A. Bakhtiari, A. Hillier, *et al.*, “Larq compute engine: Design, benchmark, and deploy state-of-the-art binarized neural networks,” in *Proceedings of MLSys*, 2021. *arXiv:2011.09398*.
- [28] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, “DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *arXiv:1606.06160*, 2016.
- [29] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, “PACT: Parameterized clipping activation for quantized neural networks,” *arXiv:1805.06085*, 2018.
- [30] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, “Learned step size quantization,” in *Proceedings of ICLR*, 2020.
- [31] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in *Proceedings of ICLR*, 2016.
- [32] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” in *Proceedings of NeurIPS*, 2015.
- [33] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1,” *arXiv:1602.02830*, 2016.
- [34] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks,” in *Proceedings of ECCV*, 2016.
- [35] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, “Bi-real net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm,” in *Proceedings of ECCV*, 2018.
- [36] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, “ReActNet: Towards precise binary neural network with generalized activation functions,” in *Proceedings of ECCV*, 2020.

- [37] Z. Lin, Y. Wang, J. Zhang, X. Chu, and H. Ling, “NAS-BNN: Neural architecture search for binary neural networks,” *Pattern Recognition*, vol. 159, p. 111086, 2025.
- [38] S. Mohammady, “Exploring neural architecture search on binary neural networks for edge computing applications,” Master’s thesis, DITEN, University of Genoa, 2025.
- [39] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in *Proceedings of ICLR*, 2019.
- [40] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct neural architecture search on target task and hardware,” in *Proceedings of ICLR*, 2019.
- [41] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” in *Proceedings of ICLR*, 2020.
- [42] A. Capello, R. Berta, H. Ballout, M. Fresta, V. Soltanmuradov, and F. Bellotti, “Enhancing μ NAS for 1D CNNs on microcontrollers,” in *Proceedings of SIE 2024*, vol. 1263 of *Lecture Notes in Electrical Engineering*, pp. 481–486, Springer, 2025.
- [43] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *NeurIPS Deep Learning and Representation Learning Workshop*, 2015. arXiv:1503.02531.
- [44] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using siamese BERT-networks,” in *Proceedings of EMNLP-IJCNLP*, 2019.
- [45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proceedings of NeurIPS*, 2017.
- [46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.
- [47] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” in *Proceedings of NeurIPS*, 2020.
- [48] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter,” *arXiv:1910.01108*, 2019.
- [49] N. Reimers and I. Gurevych, “Making monolingual sentence embeddings multilingual using knowledge distillation,” in *Proceedings of EMNLP*, 2020.
- [50] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain, and A. Farhadi, “Matryoshka representation learning,” in *Proceedings of NeurIPS*, 2022.

- [51] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in *Advances in Neural Information Processing Systems 28*, 2015. arXiv:1509.01626.
- [52] D. Gillick, C. Brunk, O. Vinyals, and A. Subramanya, “Multilingual language processing from bytes,” *arXiv:1512.00103*, 2016.
- [53] J. H. Clark, D. Garrette, I. Turc, and J. Wieting, “CANINE: Pre-training an efficient tokenization-free encoder for language representation,” in *Transactions of the ACL*, 2022.
- [54] L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel, “ByT5: Towards a token-free future with pre-trained byte-to-byte models,” *Transactions of the ACL*, 2022. arXiv:2105.13626.
- [55] J. Wang, T. Gangavarapu, J. N. Yan, and A. M. Rush, “MambaByte: Token-free selective state space model,” *arXiv:2401.13660*, 2024.
- [56] F. Feng, Y. Yang, D. Cer, N. Arivazhagan, and W. Wang, “Language-agnostic BERT sentence embedding,” in *Proceedings of ACL*, pp. 878–891, 2022.
- [57] B. Edizel, A. Mantrach, and X. Bai, “Deep character-level click-through rate prediction for sponsored search,” in *Proceedings of SIGIR*, pp. 305–314, 2017.
- [58] P. Kaliamoorthi, S. Ravi, and Z. Kozareva, “PRADO: Projection attention networks for document classification on-device,” in *Proceedings of EMNLP-IJCNLP*, 2019.
- [59] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” *Proceedings of EACL*, 2017. arXiv:1607.01759.
- [60] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, “Feature hashing for large scale multitask learning,” in *Proceedings of ICML*, pp. 1113–1120, 2009.
- [61] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” in *Proceedings of ICML*, pp. 2285–2294, 2015.
- [62] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.
- [63] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang, “Compositional embeddings using complementary partitions for memory-efficient recommendation systems,” in *Proceedings of KDD*, pp. 165–175, 2020.

- [64] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *Proceedings of ICLR*, 2019. arXiv:1711.05101.
- [65] M. Horowitz, “Computing’s energy problem (and what we can do about it),” in *IEEE ISSCC*, pp. 10–14, 2014.
- [66] Z. Mao, J. Li, H. Wang, X. Zeng, and K.-F. Wong, “DIGAT: Modeling news recommendation with dual-graph interaction,” in *Findings of EMNLP*, 2022. arXiv:2210.05196.