Vector Floating-Point Architectures for Softmax and Exponential Acceleration

ENS Rennes – M1 SIF research project, 2026–2027

Supervisors: Louis Ledoux and Olivier Sentieys
TARAN team – Inria, Université de Rennes / IRISA

Context and motivation

Matrix multiplication is heavily accelerated in modern AI hardware, making nonlinear functions such as *softmax* potential performance bottlenecks [1, 2]. This project studies a vector floating-point unit (FPU), focusing on its exponential operator.

The work is part of **CAMELIA**, a French research programme developing modular, energy-efficient AI accelerators and their software stack. Within this programme, TARAN's target architecture combines a matrix–matrix multiplication engine with a vector unit that processes its results. The internship explores this vector unit's support for softmax.

For a vector of real-valued scores $x = (x_1, \ldots, x_n)$, softmax produces nonnegative weights that sum to one:

$$y_i = \mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}. \tag{1}$$

Direct evaluation can overflow when the scores are large. To avoid this, implementations subtract the largest score, $m = \max_j x_j$, before evaluating the exponentials:

$$y_i = \frac{\exp(x_i - m)}{\sum_{j=1}^{n} \exp(x_j - m)}. \tag{2}$$

This gives the same weights: the common factor $\exp(-m)$ cancels between numerator and denominator. Each exponential is now at most one, avoiding overflow in its evaluation.

In Transformer models, *attention* lets each token (for example, a word fragment) combine information from other tokens according to their relevance. Matrix multiplication computes relevance scores between learned representations of the tokens; softmax turns these scores into weights for combining their information.

Therefore, keeping pace with the matrix engine also requires efficient softmax hardware. Alongside exponentiation, it must find the maximum, sum results across vector lanes, and normalize the outputs. This motivates studying both the exponential operator and its place within the vector FPU.

Research objective

How should exponential units be organized within a vector FPU to balance softmax throughput, hardware cost, and numerical accuracy? Possible directions include sharing or replicating arithmetic resources across lanes, choosing approximation methods, and adapting numerical precision [10, 11]. The study may consider reduced-precision formats such as FP16 or BF16, or mixed precision with wider accumulation.

For an exponential input z (here, $z = x_i - m$), one possible approach is range reduction [9]:

$$z = k \ln 2 + r, \quad k \in \mathbb{Z}, \quad |r| \le \tfrac{1}{2} \ln 2, \qquad \exp(z) = 2^k \exp(r) \approx 2^k P_d(r), \tag{3}$$

where k is chosen by rounding $z/\ln 2$ to the nearest integer, and P_d is a degree-d polynomial approximating exp on the reduced interval. Lookup tables, piecewise polynomials, and combinations of tables and arithmetic offer different area, latency, and accuracy trade-offs [3]. For softmax, specialization to $z \le 0$ and approximation of very small outputs are also worth investigating, while measuring the resulting error in the normalized weights.

Possible work

The internship will begin with a review of vector FPUs, exponential approximations, and softmax accelerators. Several existing architectures provide complementary starting points: **FPnew** [4] demonstrates configurable floating-point arithmetic with SIMD support; **Softermax** [1] explores low precision, base replacement, and online normalization; **SoftEx** [5] combines BF16 exponential lanes with FP32 accumulation; and **VEXP** [2] integrates approximate exponentiation into a RISC-V FPU.

Building on this survey, the student and supervisors will select questions to investigate in greater depth. Possible directions include:

- **Arithmetic choices:** compare exponential approximations and number formats, and study how their errors affect the softmax weights. Software experiments can help identify useful accuracy–cost trade-offs.
- **Vector architecture:** explore how exponential units can be combined with the other operations needed by softmax. Resource sharing, parallelism, pipelining, and data buffering are among the choices to examine.
- **Implementation and evaluation:** prototype selected ideas using simulation or hardware generation, and assess their numerical accuracy, performance, and hardware cost. FPGA or ASIC synthesis may help evaluate their suitability for the target accelerator.

Tools and research environment

Arithmetic core generators may support this exploration. **FloPoCo** [6] provides parameterized arithmetic operators, including exponential architectures [3]. **HAriCo** (Hardware Arithmetic Cores), co-developed by Florent de Dinechin, Pierre Cochard, and Louis Ledoux, is an upcoming generator based on intermediate representations (IRs). This approach enables arithmetic transformations and connections with **MLIR and CIRCT**, the compiler infrastructure for hardware design in the LLVM ecosystem [7, 8]. These tools offer opportunities to (faster) explore arithmetic generation and its integration into the CAMELIA design flow.

Student profile

An interest in computer architecture or computer arithmetic, and programming experience in Python or C/C++, are expected. Familiarity with digital design, VHDL/Verilog, ASIC, or compiler infrastructure is useful and can be developed during the internship.

References

[1] J. R. Stevens et al. Softermax: Hardware/Software Co-Design of an Efficient Softmax for Transformers. DAC, 2021.

[2] R. Wang et al. VEXP: A Low-Cost RISC-V ISA Extension for Accelerated Softmax Computation in Transformers. arXiv:2504.11227, 2025.

[3] F. de Dinechin and B. Pasca. Floating-point exponential functions for DSP-enabled FPGAs. FPT, pp. 110–117, 2010.

[4] S. Mach, F. Schuiki, F. Zaruba, and L. Benini. FPnew: An Open-Source Multi-Format Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing. arXiv:2007.01530, 2020.

[5] A. Belano et al. SoftEx: A Low Power and Flexible Softmax Accelerator with Fast Approximate Exponentiation. DATE, 2025. Open-source implementation.

[6] F. de Dinechin and B. Pasca. Designing Custom Arithmetic Data Paths with FloPoCo. *IEEE Design & Test of Computers*, 28(4):18–27, 2011. Project website.

[7] L. Ledoux. Digital Design with MLIR and CIRCT. ACM Europe School on MLIR, 2026. Tutorial introducing HAriCo and arithmetic lowering to CIRCT.

[8] CIRCT project. Circuit IR Compilers and Tools. Project documentation, `https://circt.llvm.org/`.

[9] J.-M. Muller. Elementary Functions: Algorithms and Implementation. 3rd edition, Birkhäuser, 2016.

[10] F. de Dinechin and M. Kumm. Application-Specific Arithmetic: Computing Just Right for the Reconfigurable Computer and the Dark Silicon Era. Springer, 2024.

[11] J.-M. Muller et al. Handbook of Floating-Point Arithmetic. 2nd edition, Birkhäuser, 2018.