ECE382M.20: System-on-Chip (SoC) Design

Lecture 1 – Project Overview

Andreas Gerstlauer
Electrical and Computer Engineering
The University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 1: Outline

- **Marketing requirements**
  - Market focus, product description
  - Cost metrics, product features
- **Product requirements**
  - LLM inference
  - Hardware acceleration
- **Project description**
  - Language models
  - Transformer networks
  - llama.cpp LLM
  - Hardware and software development tasks

Market Focus

- **Natural language processing (NLP)**
  - Human-machine interactions
  - Text generation, reasoning, dialogue
  - Knowledge retrieval / artificial intelligence (AI)
  - Agentic loops to perform larger tasks
  - ➢ Large language models (LLMs)
- ➢ **What problem are we trying to solve?**
  - Edge LLM
    - On device, offline
    - Low-latency responses
  - Support edge tasks like
    - Virtual assistants, machine translation, …
    - Phones, glasses, cars, appliances, etc.

Some Competition

- **NVIDIA Jetson Nano**
  - https://nvidia.github.io/TensorRT-Edge-LLM
  - Arm+GPU based solution
- **Taalas (acquired by AMD)**
  - https://taalas.com/products/
  - Hardcoded Llama models burned into silicon
- **Google Edge TPU aka Coral**
  - https://gweb-coral-full.uc.r.appspot.com/
  - Standalone custom hardware accelerator
- **Plus other small startups…**

Product Description

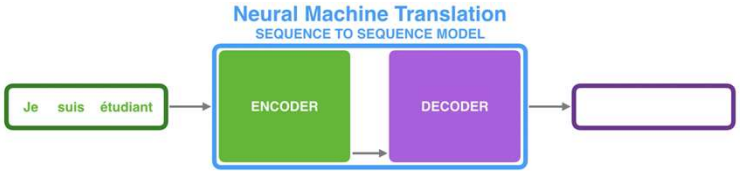
- **Edge LLM SoC**
  - Deliver hardware + software intellectual property (IP)
- **Cost metrics**
  - Real-time: tokens per second, time-to-first-token
  - Task accuracy: % correctness score
  - Power/thermal: W and operating temperature (°C)
  - Cost: $ or die area (mm^2)
- **Product features**
  - Supported tasks and I/O
  - Supported LLMs
  - Flexibility: dynamic, over-the-air reprogramming/updating

Product Requirements

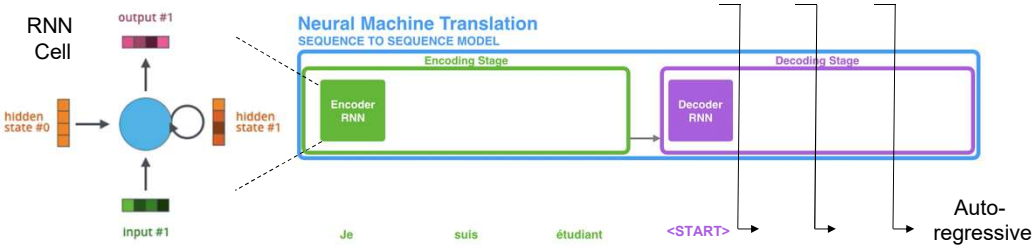
- **High task accuracy → model size**
  - Number of LLM parameters
  - Trained on large data set
  - Computationally intensity
- **High token rate, low power → hardware acceleration**
  - Key/dominating computational kernels
  - Matrix operations
  - General matrix-matrix multiplication (GEMM)
  - General matrix-vector multiplication (GEMV)
- **Flexiblity → software support**
  - Standard embedded Linux environment
  - Software optimizations for performance and power

Natural Language Processing (NLP)

- **Sequence-to-sequence (seq2seq) models for translation**



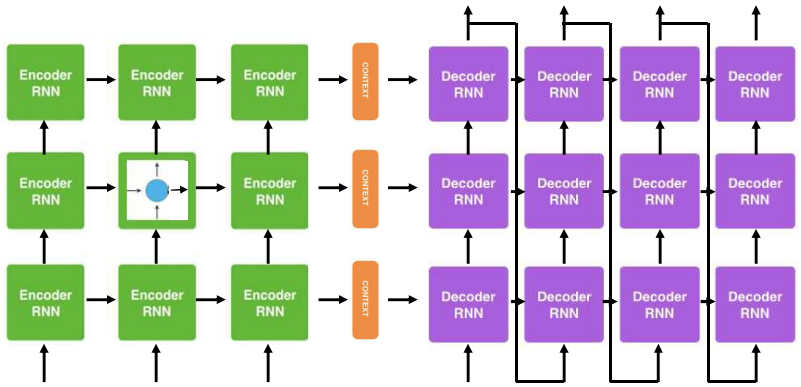
- **Input sequence: learned token embeddings**
  - Map words into input vectors in N-dimensional space
  - Attach meaning (e.g. King – Man + Woman = Queen)
- **Recurrent neural networks (RNNs), e.g. LSTMs**



Source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

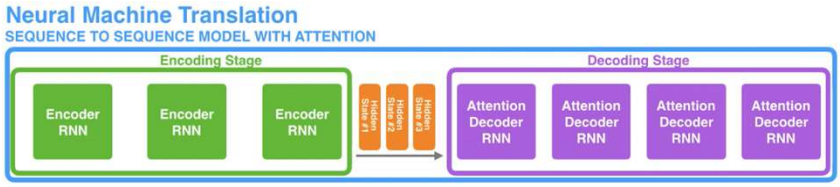
Multi-Layer Deep Encoder-Decoder RNN

- **Unrolled and stacked RNN**
  - Unrolled over time steps (same cell, single set of weights)
  - Multiple stacked layers

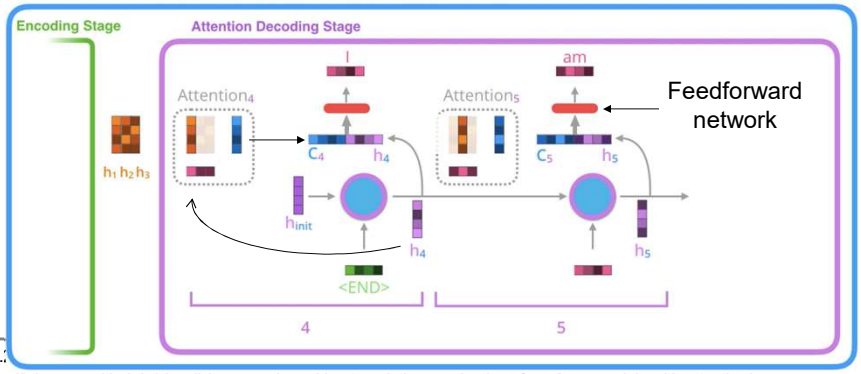


Attention

- **Challenge to capture long sequence in one state (context)**
  - ➢ Pass all intermediate states



- **Decoding with *attention* to retrieve relevant information**

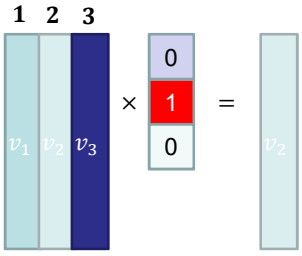


ECE382M.2

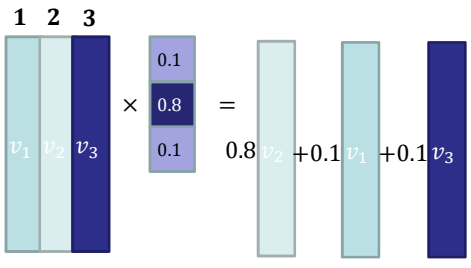
Source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention Is Weighted Averaging Lookup

- **Dictionary: hard indexing**
  - dict = {<key>: <value>}
  - dict[<query>] lookup
  - Matrix-vector product with one-hot vector over key/query matches



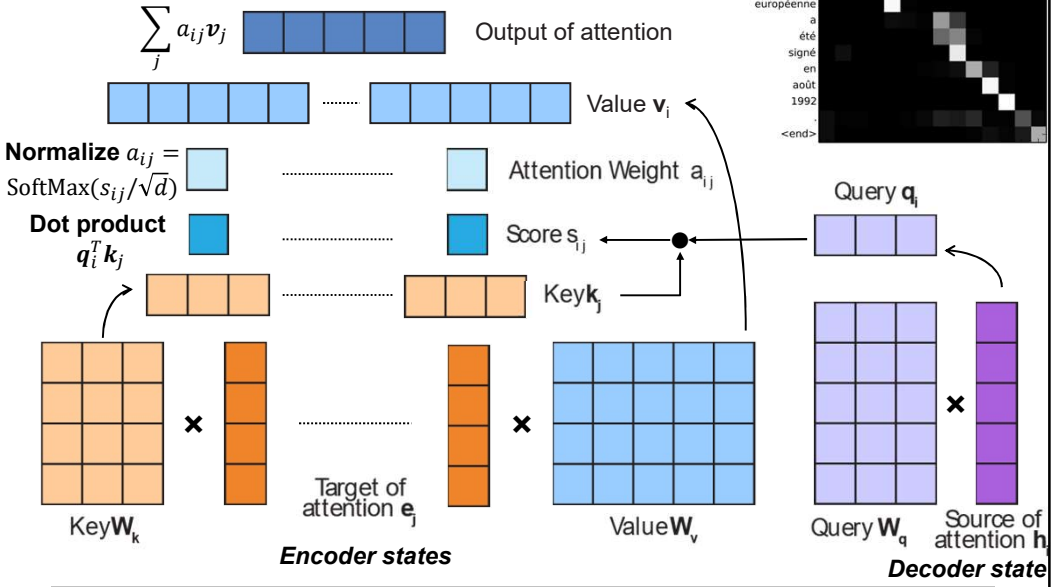
- **Attention: soft indexing**
  - Weighted sum of values
  - Vector of scores/weights
  - Distribution over keys
  - Computed based on similarity of key and query
  - Feature-based attention

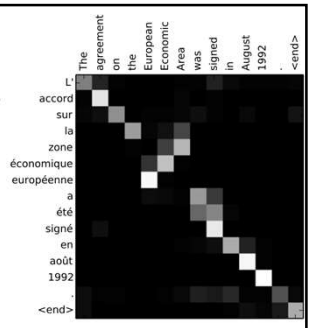


Source: B. Wang, Harvard

QKV Attention

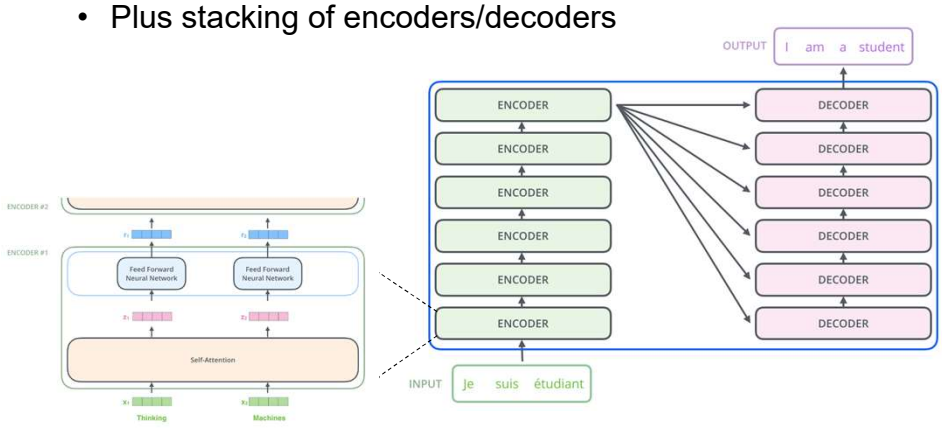
- **Learned linear projections (weights)**





Transformers

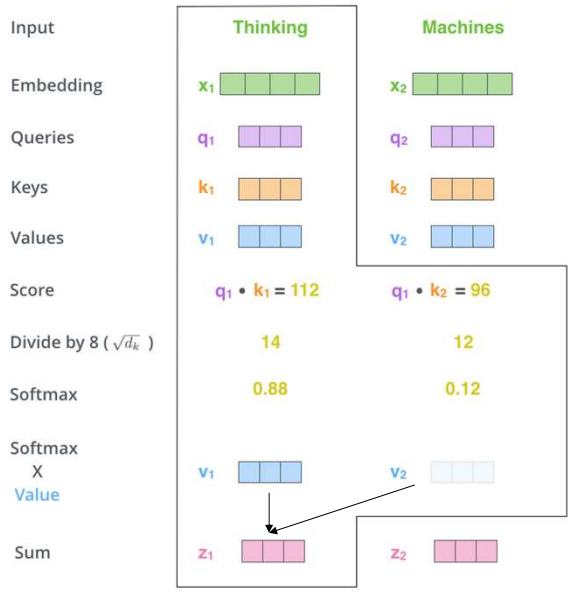
- **Attention is all you need [Vaswani'17]**
  - *Self-attention* to replace (unrolled) recurrence
  - Apply attention to encoder (self- vs. cross-attention)
  - Avoid issues like vanishing gradients during training
  - Plus stacking of encoders/decoders



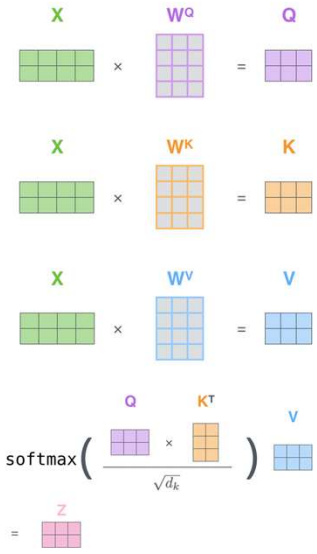
Source: https://jalammar.github.io/illustrated-transformer/

Self-Attention

- **Attend each word to each other word in sequence**



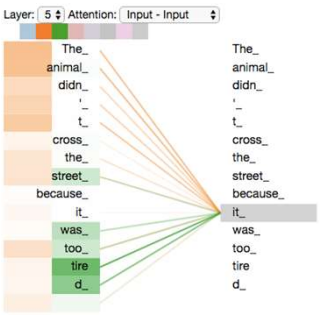
Matrix form:

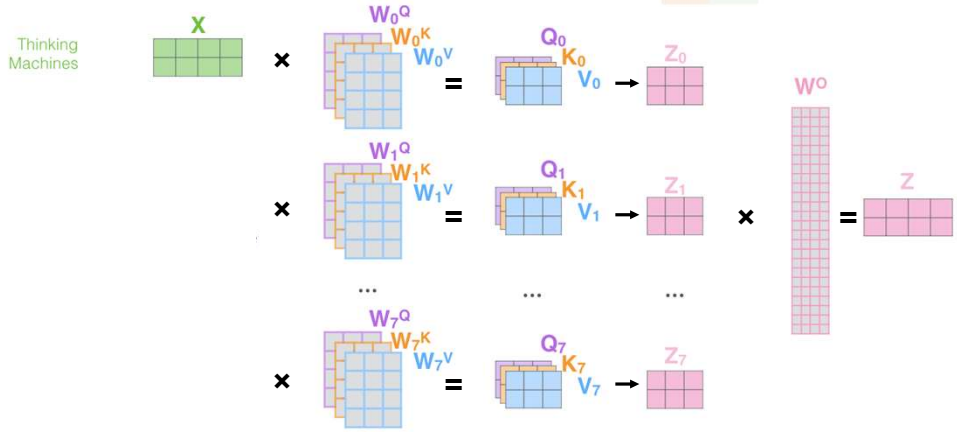


Source: https://jalammar.github.io/illustrated-transformer/

Multi-Headed Attention

- **Compute multiple attention matrices**
  - Multiple QKV weights
  - Concatenate and combine using another O weight matrix

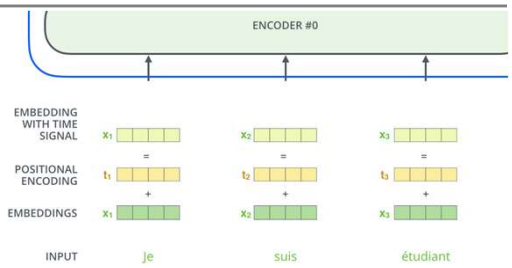




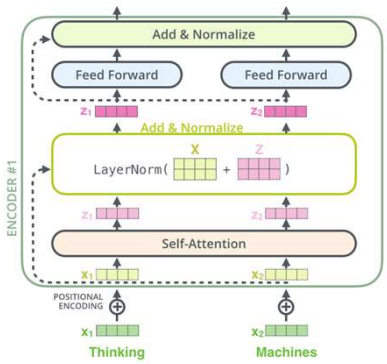
Source: https://jalammar.github.io/illustrated-transformer/

Additional Transformer Pieces

- **Position encoding**
  - Represent order in the sequence
  - Usually added



- **Layer normalization**
  - Residual connections
  - Normalization

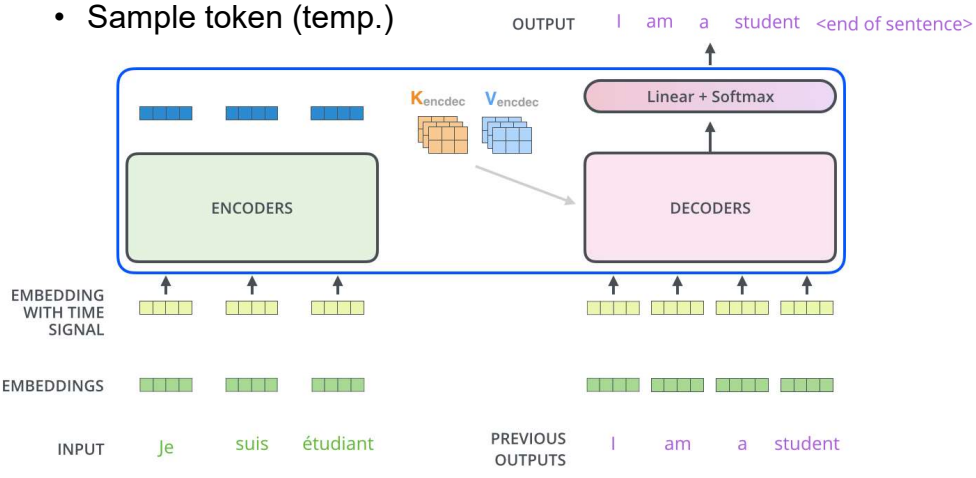


ECE382M.20: SoC Design, Lecture 1 © 2026 A. Gerstlauer 15

Source: https://jalammar.github.io/illustrated-transformer/

Encoder-Decoder

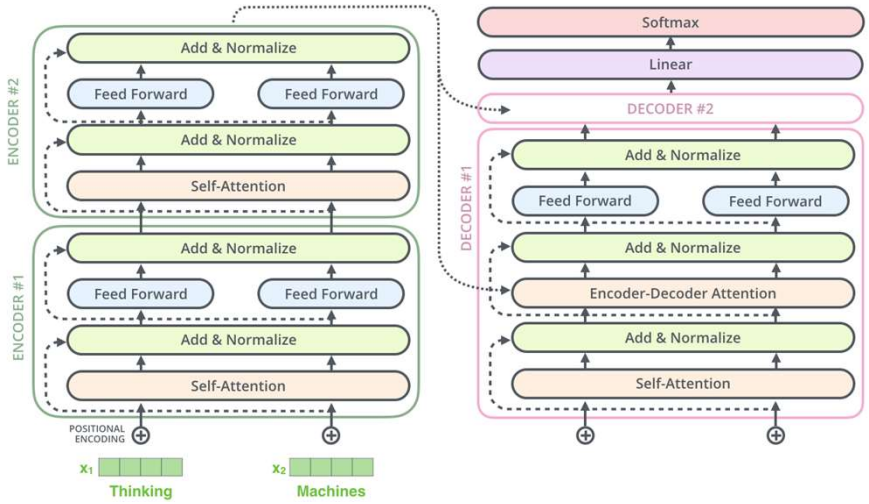
- **Passing top encoder's output transformed to KV matrices**
  - Cross-attention, in addition to self-attention in decoder
- **Output linear (fully-connected) & softmax layer**
  - Word probabilities (logits) & distribution
  - Sample token (temp.)



ECE382M.20: SoC Design, Lecture 1 © 2026 A. Gerstlauer 16

Full Transformer Architecture

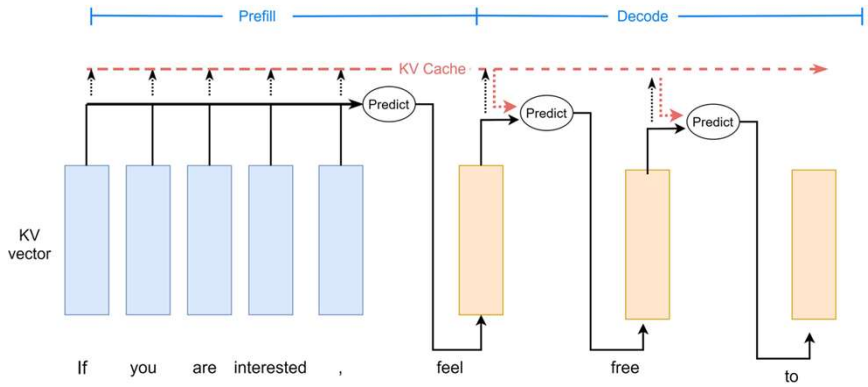
- **Example with 2 stacked encoders and decoders**
  - Each decoder w/ self-attention followed by cross-attention
  - Decoder self-attention is masking future words (set to -∞)



Transformer Variants

- **Encoder-decoder models**
  - Text translation, transformation or summarization
  - E.g. T5, BART
- **Encoder-only models**
  - Text understanding, classification and extraction
  - E.g. BERT
- **Decoder-only models**
  - Text generation, language modeling, Q&A
  - E.g. GPT, Llama

Decoder-Only LLMs



- **Prefill**
  - Process prompt, compute KV matrices, fill KV cache
  - GEMM operations, size depends on prompt length
  - Compute-bound
  - Time-to-first-token (TFT)
- **Decode**
  - Predict next token, append to KV cache
  - GEMV operations, size depends on KV cache size
  - Memory- and compute-bound
  - Inter-token latency (ITL)

Llama and llama.cpp

- **Llama: Meta's family of decoder-only LLMs** LLaMA
  - Specific activations, positional encodings, normalizations
  - Different hyperparameters (layers, dimensions)
  - Open-source inference code (and leaked models)
- **llama.cpp: LLM inference engine in C/C++** llama.cpp
  - Built on top of GGML tensor library
  - Load pre-trained models in GGUF format (Hugging Face)
  - Different quantization levels
  - Various backends incl. optimizations (GPU, Arm)

Project Description

- **HW/SW co-design of an embedded SoC**
  - Low-power llama.cpp implementation
    - One Arm A53 @ 300MHz + 100mW max.
  - ARM-based target platform
    - Arm Cortex-A53 processor, memory components, I/O devices
    - Custom hardware accelerators
    - Interconnected via standard system busses or memory/cache interfaces
  - Virtual and physical prototyping
    - SystemC TLM-based virtual platform model (QEMU ARM simulator)
    - ARM- and Xilinx FPGA-based prototyping board (Zynq Ultrascale+)

  ➢ Lab and project in teams
    ➢ 3 per team, 24 teams for 24 boards

Project Objectives and Activities

- **Project objective:**
  - Implement the llama.cpp code on a ARM based SoC while meeting the performance, area and power metrics.
- **Project activities:**
  - Profile the software implementation to determine performance bottlenecks
  - Optimize the software (quantization, …)
  - Partition the software into components which will run on the Arm processor and on hardware accelerators
  - Synthesize accelerators into Verilog for gate level implementation
  - Co-simulate and prototype the HW/SW implementation
  - Estimate timing, area and power metrics and validate against product requirements

Development Tasks

- **ARM software development**
  - Compile and profile llama.cpp on Arm board
  - Explore models and application-level optimizations
  - Explore software optimizations on Arm platform
  - Develop hardware abstraction layer (HAL) and I/O handler
  - Develop interrupt handler & driver (Linux kernel module)
- **Hardware development on FPGA**
  - Hardware accelerators (synthesize kernel code)
  - Interface to ARM board and on-chip bus
  - Memory/cache interfaces (optional DRAM controller)
  - Interrupt logic, clocking & reset
  - Debug, diagnostics