

ChipNeMo: Domain-Adapted LLMs for Chip Design

Mingjie Liu[§], Teo Ene[§], Robert Kirby[§], Chris Cheng[§], Nathaniel Pinckney[§], Rongjian Liang[§]
Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, Bonita Bhaskaran
Bryan Catanzaro, Arjun Chaudhuri, Sharon Clay, Bill Dally, Laura Dang, Parikshit Deshpande
Siddhant Dhodhi, Sameer Halepete, Eric Hill, Jiashang Hu, Sumit Jain, Bruce Khailany
Kishor Kunal, Xiaowei Li, Hao Liu, Stuart Oberman, Sujeet Omar, Sreedhar Pratty, Ambar Sarkar
Zhengjiang Shao, Hanfei Sun, Pratik P Suthar, Varun Tej, Kaizhe Xu, Haoxing Ren
NVIDIA

ChipNeMo: Domain-Adapted LLMs for Chip Design

How Nvidia uses OSS LLM to improve engineering productivity

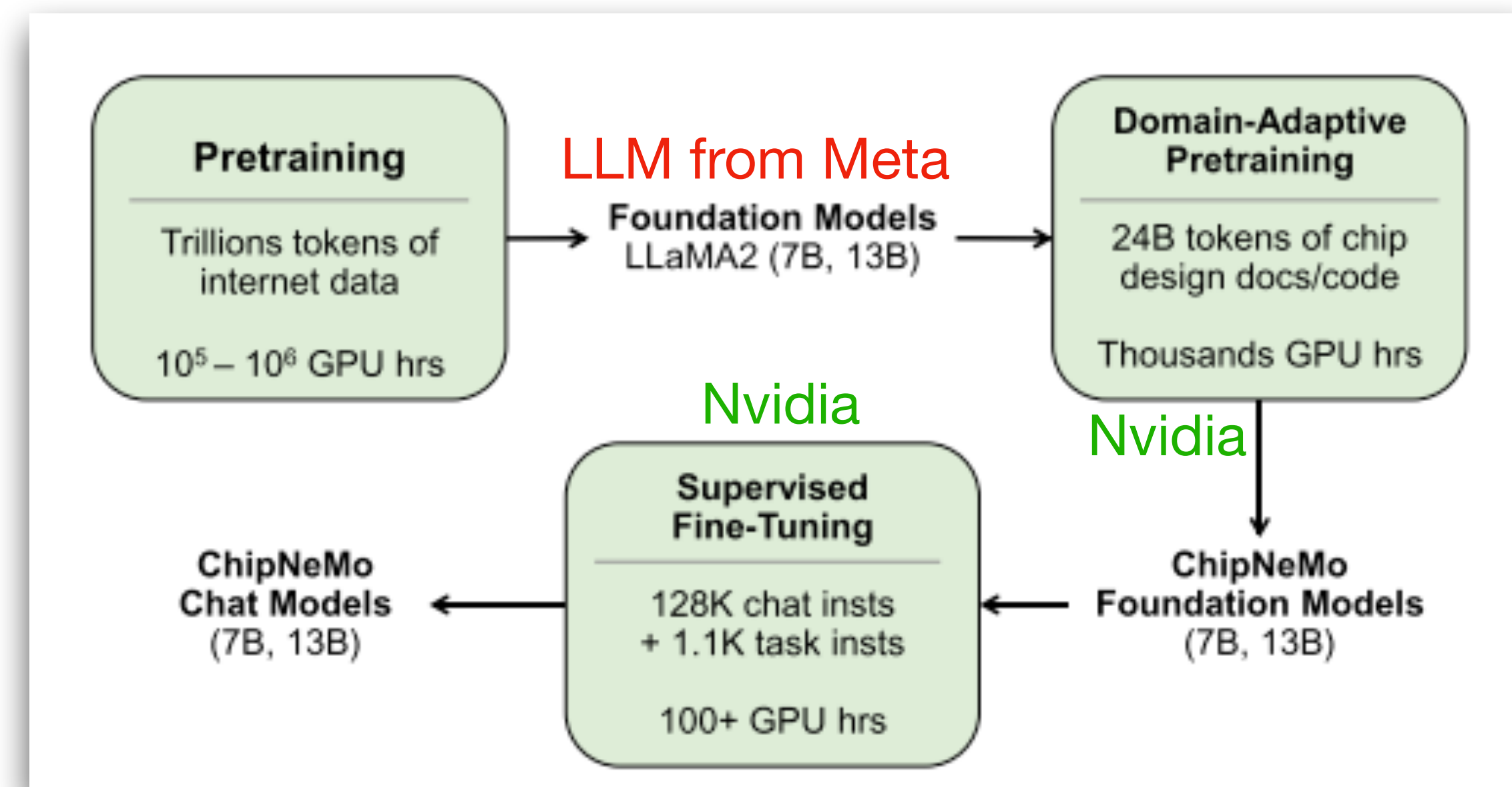
Insop, 1/10/2024, v1

Orange sticky notes are my (ISS's)
opinionated comments.

<https://arxiv.org/abs/2311.00176>

Overview

- Motivation
 - Tool to help chip design
- Opportunity
 - Advances in LLM both closed and open source
- What
 - Trained LLama2 (7B, 13B) with dataset (24B tokens w/ document & code)
- How they use
 - Engineering assistant chatbot
 - EDA script generation
 - BUG summarization and analysis

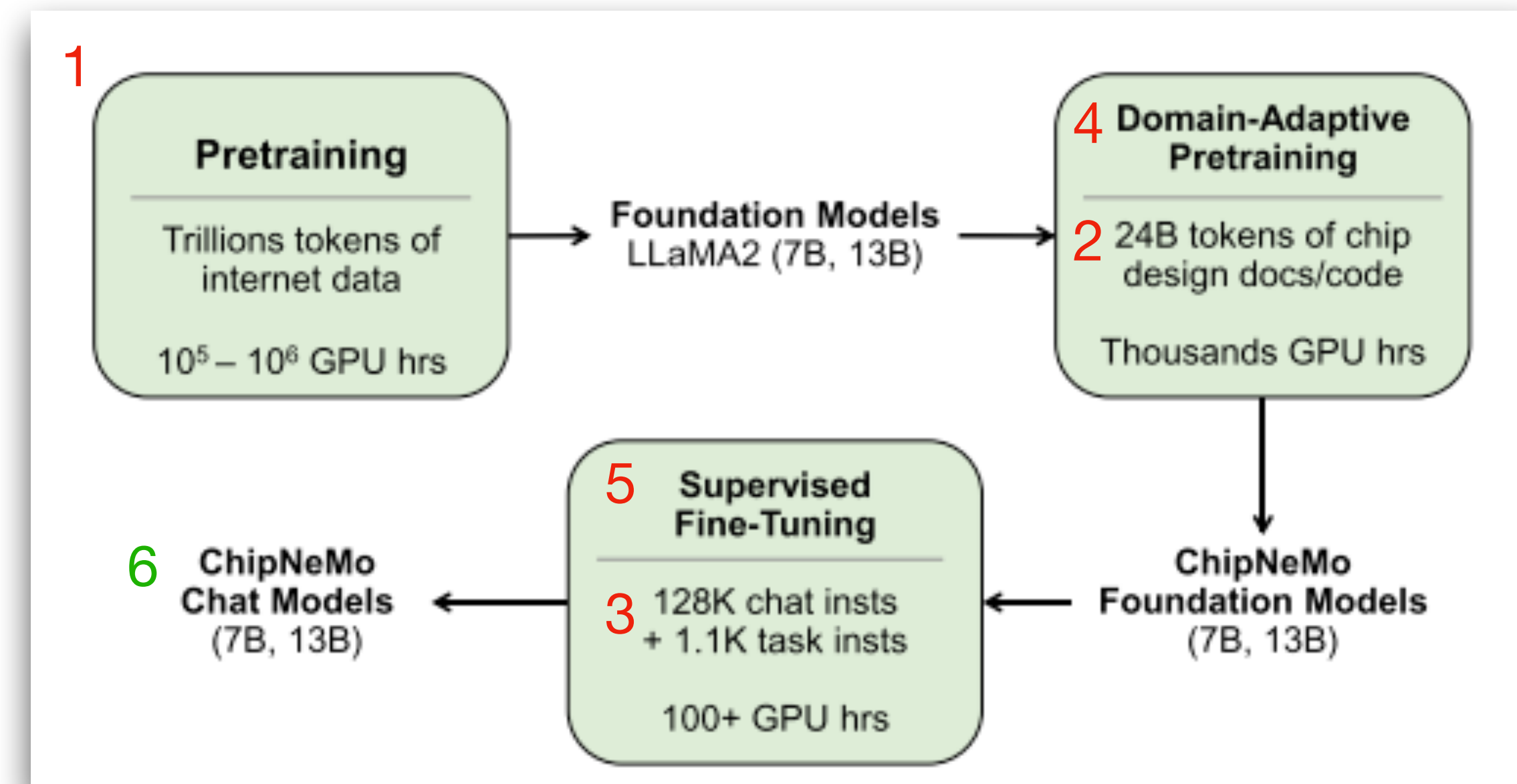


Why OSS LLMs?

- Closed LLMs are highly performant in general domain
- Domain specific LLM could outperform general LLM on domain specific task
 - BloomerGPT, BioMedLLM
- Shows promising result in code generation including hw domain with limited degree (link)

Training flow

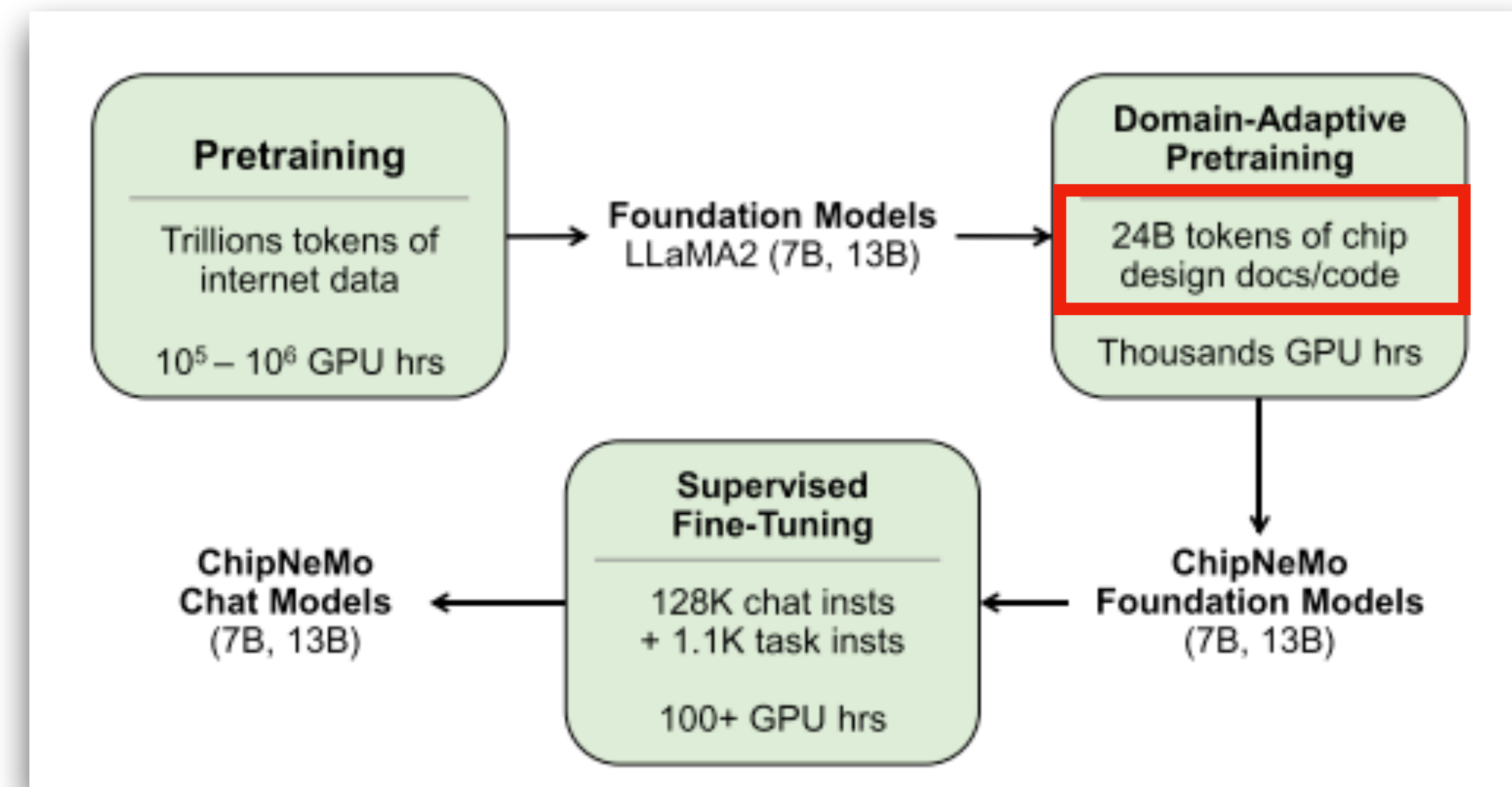
1. Get Llama model checkpoint from Meta
2. Prepare dataset with document/code
3. Prepare chat dataset with Q&A pair
4. Train OSS LLM (Llama) with dataset, called 'Domain-Adaptive Pre-Training (**DAPT**)' dataset from step 2
5. Train the DAPT model with chat dataset, called 'Supervised Fine-Tuning (**SFT**)' with dataset from step 3
6. Use the final model, called **ChipNemo**



Dataset

Domain-specific pre-training

- Domain-specific pre-training dataset
 - Mixed with private and public datasets
 - Code (SW, RTL, verification test bench)
 - Text (HW specifications, documentations)
- Clean and filtering



Dataset

Domain-specific pre-training

- Throw all applicable documents, no limit
- Downsample, code is used for 24.5%
 - Selectively increase code that deemed pertinent
- Upsample, documentation is used 34% during training
- Training over 2 to 4 epochs
- Total token used for training, 24.1B

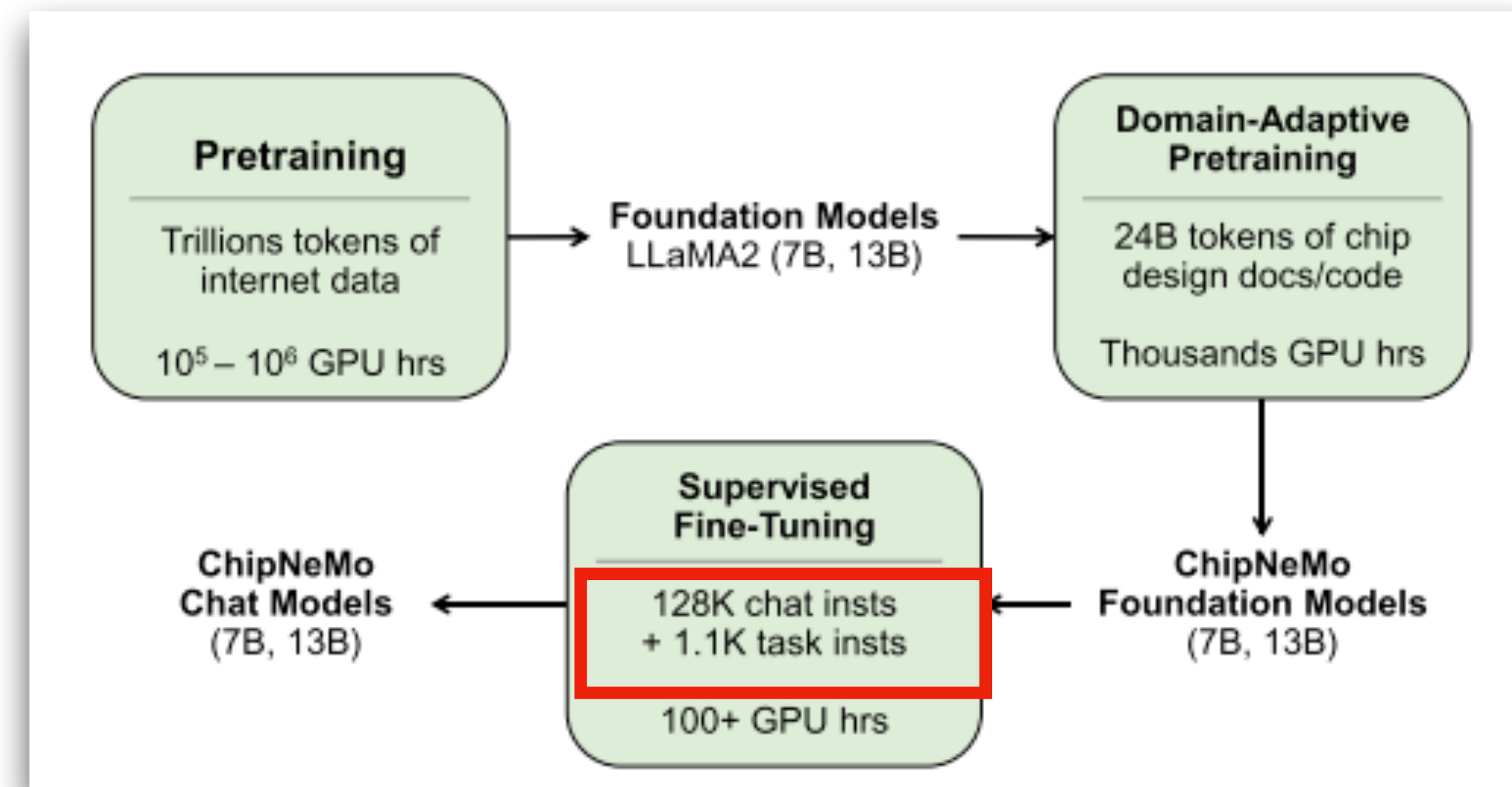
ISS: Pre-train model used public dataset & code (for code LLM) so I am not sure how much gain by adding public dataset

Data Source Type	Data Percentage (%)	Data Tokens (B)	Training Percentage (%)	Training Tokens (B)
Bug Summary	9.5%	2.4	10.0%	2.4
Design Source	47.0%	11.9	24.5%	5.9
Documentation	17.8%	4.5	34.0%	8.2
Verification	9.1%	2.3	10.4%	2.5
Other	7.9%	2.0	12.0%	2.9
Wikipedia	5.9%	1.5	6.2%	1.5
Github	2.8%	0.7	3.0%	0.7
Total	100.0%	25.3	100.0%	24.1

SFT instruction data

SFT (Supervised fine tuning)

- General and domain-specific instruction dataset
 - Large number (128K) general and small number (1K) of domain-specific
- General dataset: 128K
 - Use the general chat dataset, OASST, FLAN, P3
 - Small custom questions
 - Brainstorming, open-ended Q&A, re-writing, summarization
- Domain specific dataset: 1K
 - Human (SME) generated single turn question/answer
 - Design knowledge, EDA script generations, bug summarization & analysis



ISS: If the pre-trained model is instruct model, then 'general dataset' can be skipped

Evaluation dataset

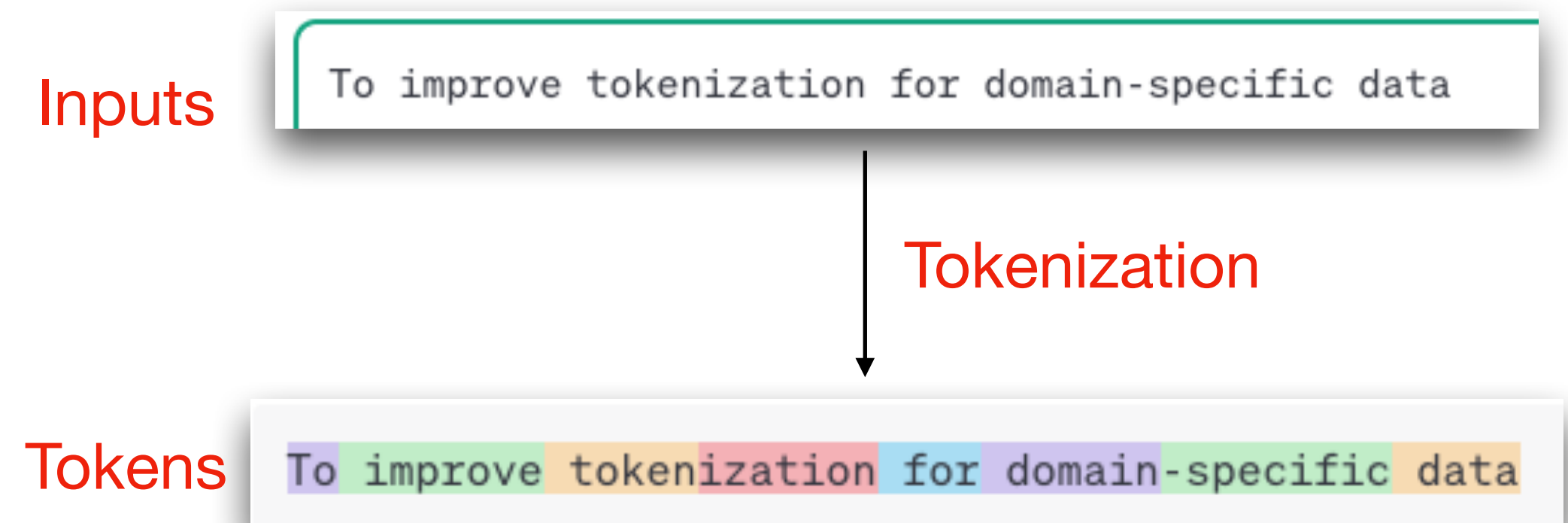
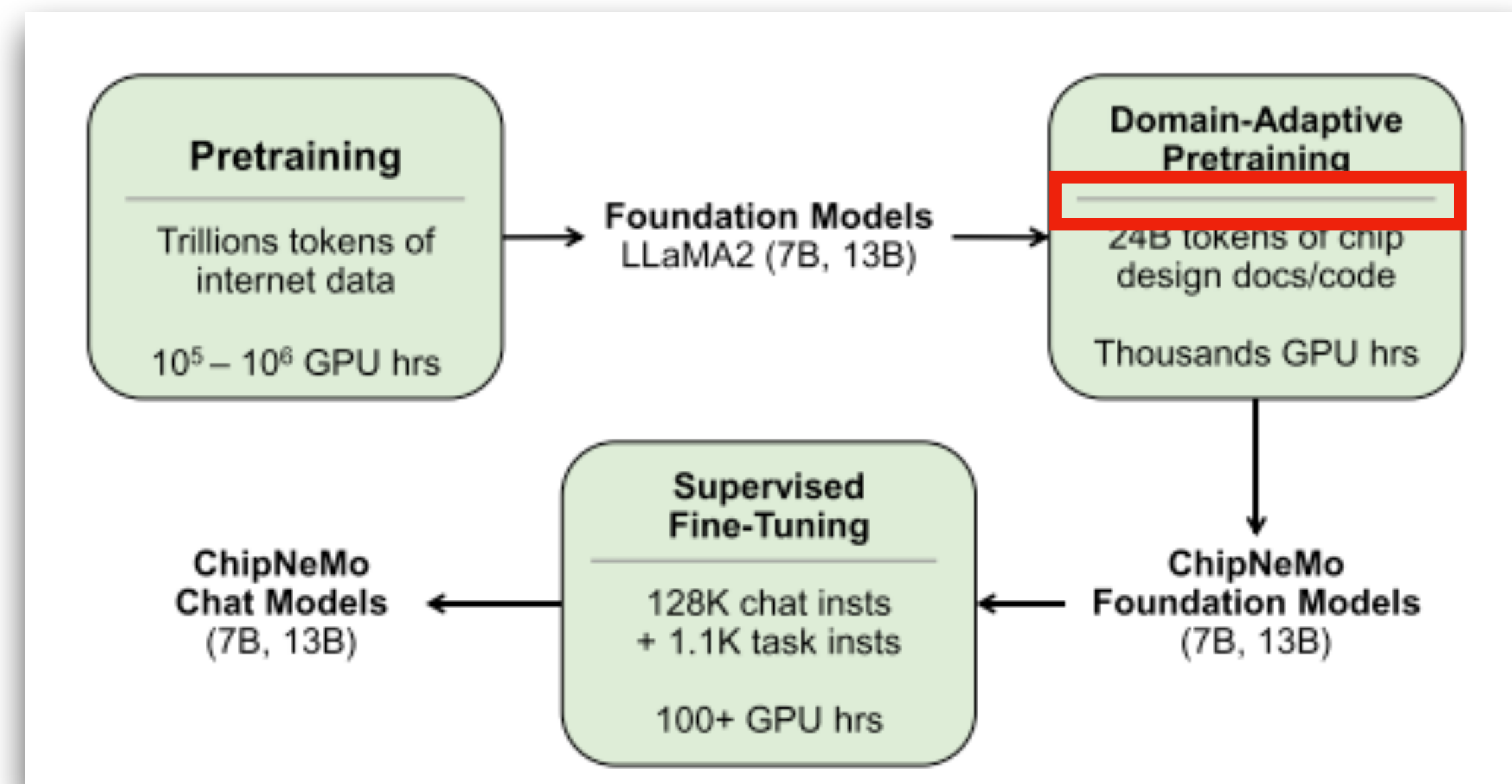
AutoEval

- To evaluate, create multi-choice QA, similar to MMLU
 - At least one complex answer choice to be challenging and realistic
- Generation was done together with SME
- Be aware of containment from the instruction dataset
- Average results from 5 runs to provide stable statistics
- Iteration was done with a set of 5-shot examples
- They also used academic benchmarks:
 - HumanEval for python
 - VerilogEval for verilog

Domain Source	Number of Questions
Design Knowledge (Design)	94
EDA Script Generation (Scripting)	74
Bug Summarization and Analysis (Bugs)	70
Open Domain Circuit Design (Circuits)	227

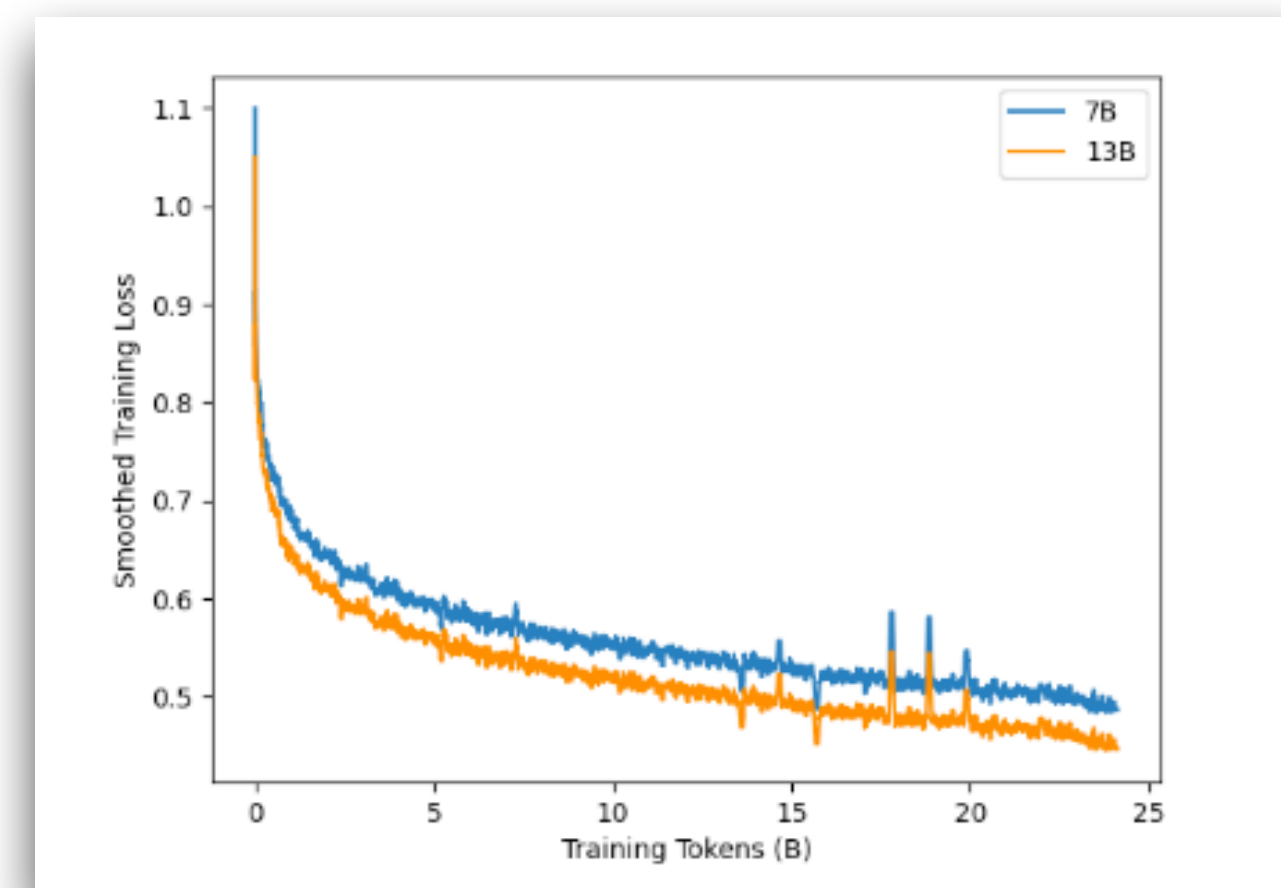
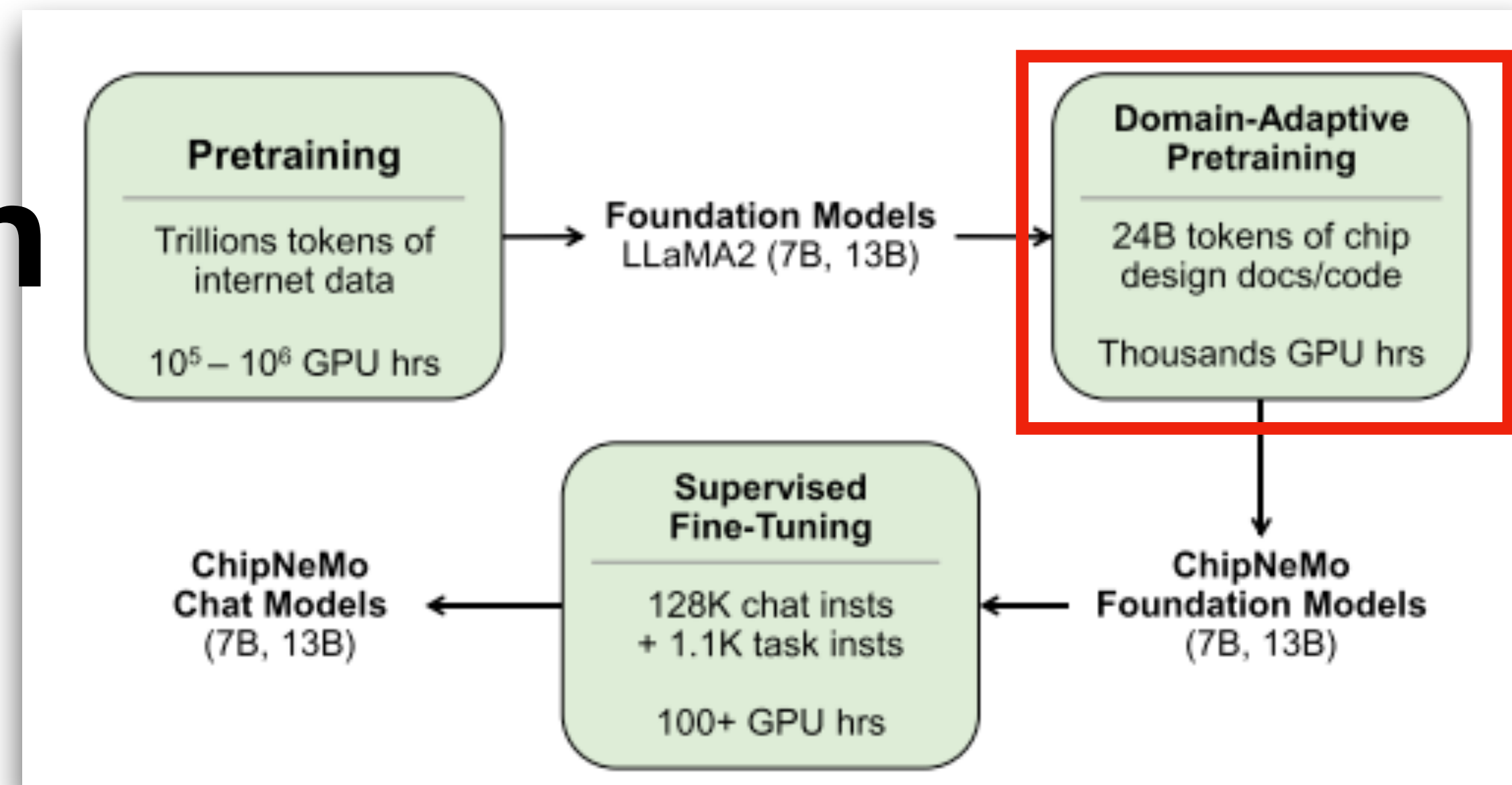
Custom Tokenizer

- *Token* is a smallest unit for model
 - Tokenizer is a property of the model
 - See example on the left
- To improve tokenization for domain-specific data
- Since Tokenizer is a property of the model, it needs to follow the steps
 1. Train tokenizer with DAPT dataset
 2. Identify tokens that are absent and rare in the original tokens
 3. Expand the original tokens with tokens from step 2
 4. Initialized the embeddings with new tokens by utilizing the old tokenizer



Domain adaptive pretrain

- LLaMA2 7/13 billion size model
- Train w/ Nvidia NeMo sw framework
 - Tensor parallelism: use multiple gpu to train
 - Flash attention: minimize memory move so improve training times
- Small learning rate, 5e-6
- Adam optimizer
- Effective batch size of 1M tokens
 - Global batch size of 256 and Context window 4096 token
- Training step of 23,200 (1 epoch of the data blend)
- Found that LoRA degrade in accuracy compared to full fine tuning

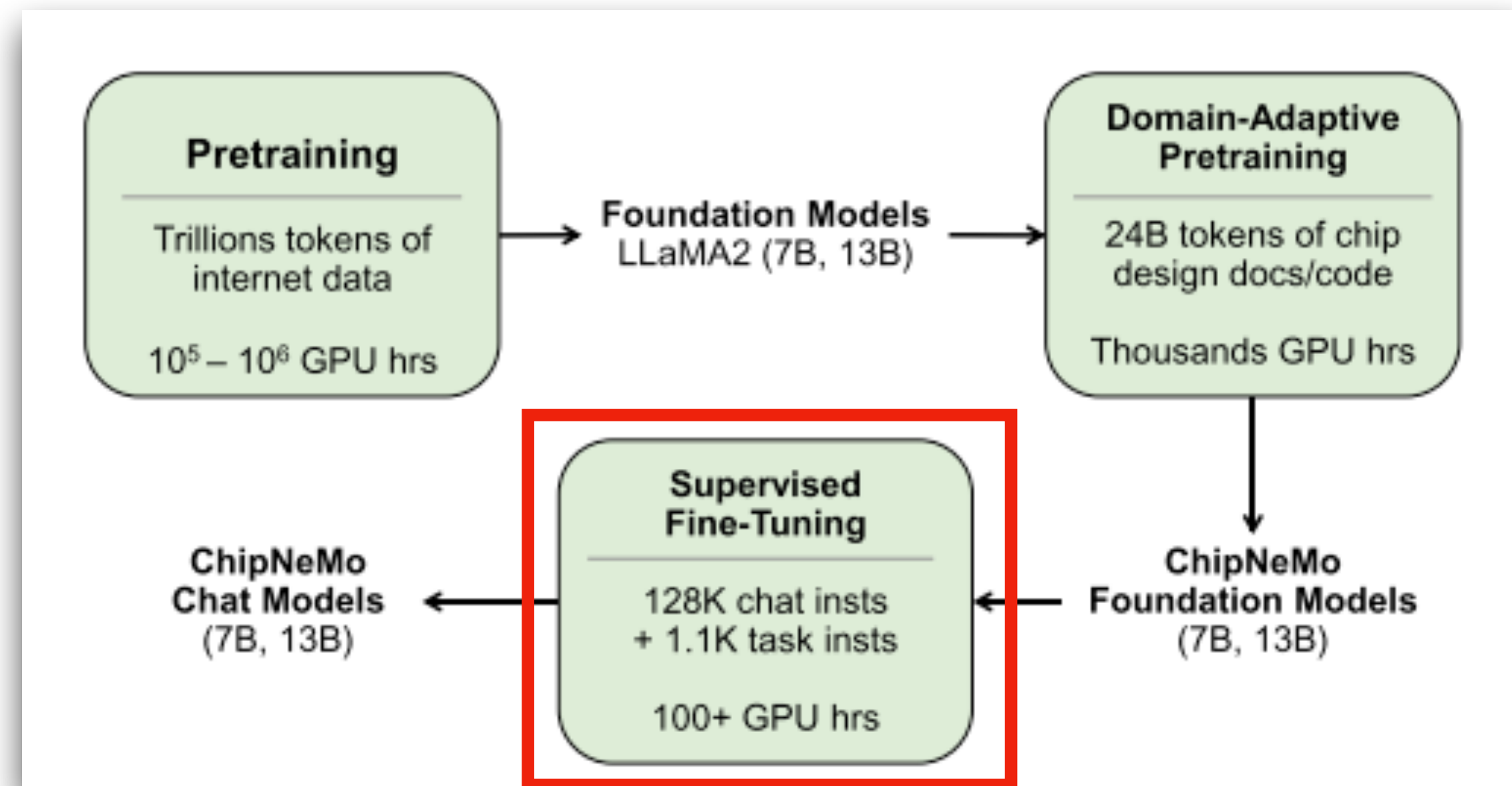


ISS: LoRA may not be effective choice when tasks are complex, which may requires more parameters to be trained.

SFT

Supervised fine-tuning

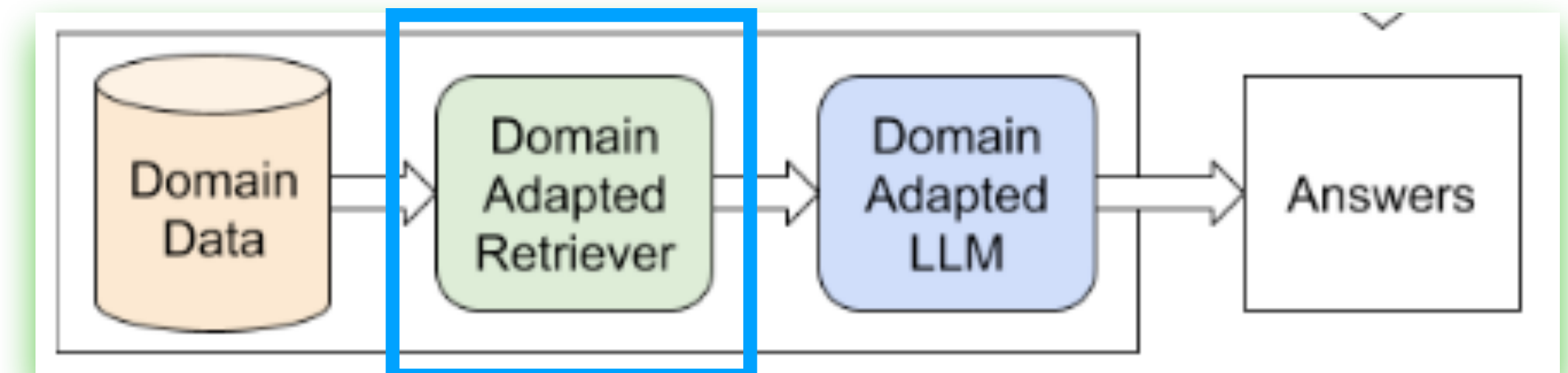
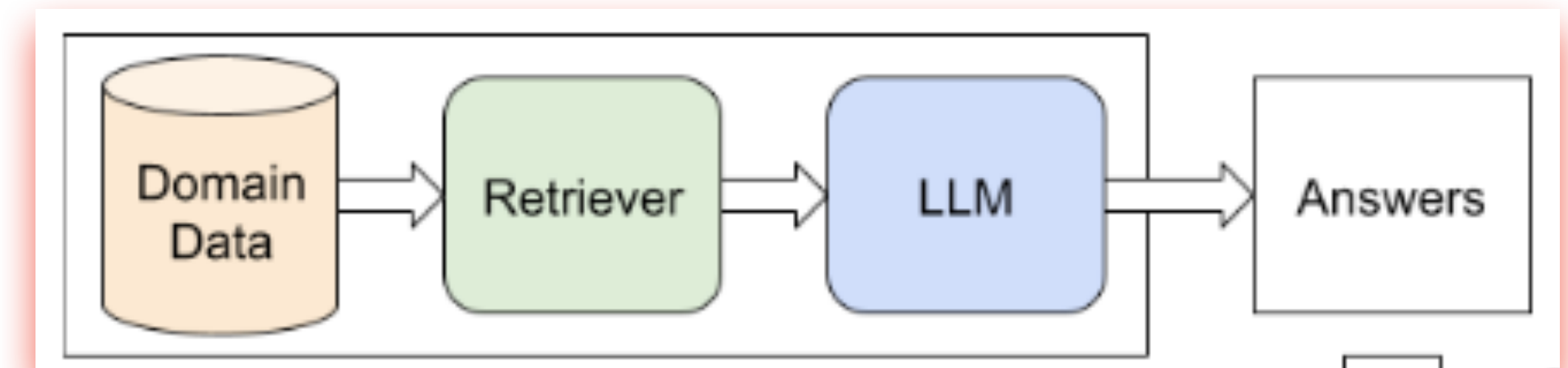
- Same parameters as DAPT except global batch size of 128
- Apply a single epoch of SFT dataset (128K+1.1K) with random sample
 - More than a single epoch shows overfitting
- When they apply DAPT dataset to Llama-chat model
 - Significant degradation is observed



ISS: This needs to be double checked. When we train code-llama-instruct with our DAPT dataset, the resulting model was performant

Retrieval-augmented generation (RAG)

- A general RAG helps to use domain text
 - Improve accuracy, avoid hallucination
- They claim that
 - Domain adapted model for RAG boost the performance significantly
 - Domain adapted retriever (customized embedding model) also improve retrieve accuracy
- Even with the retriever improvement, retrieval still struggles map some queries to relevant text

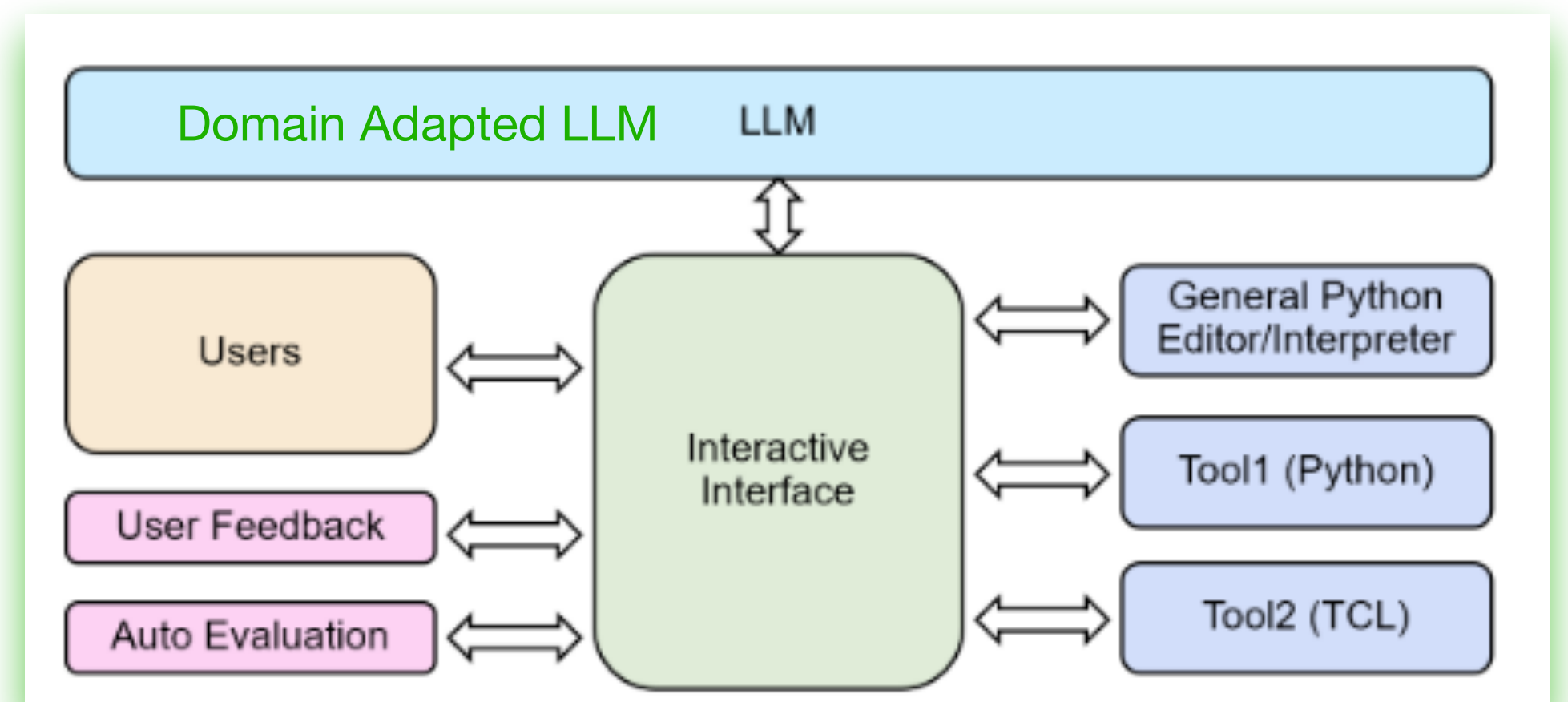
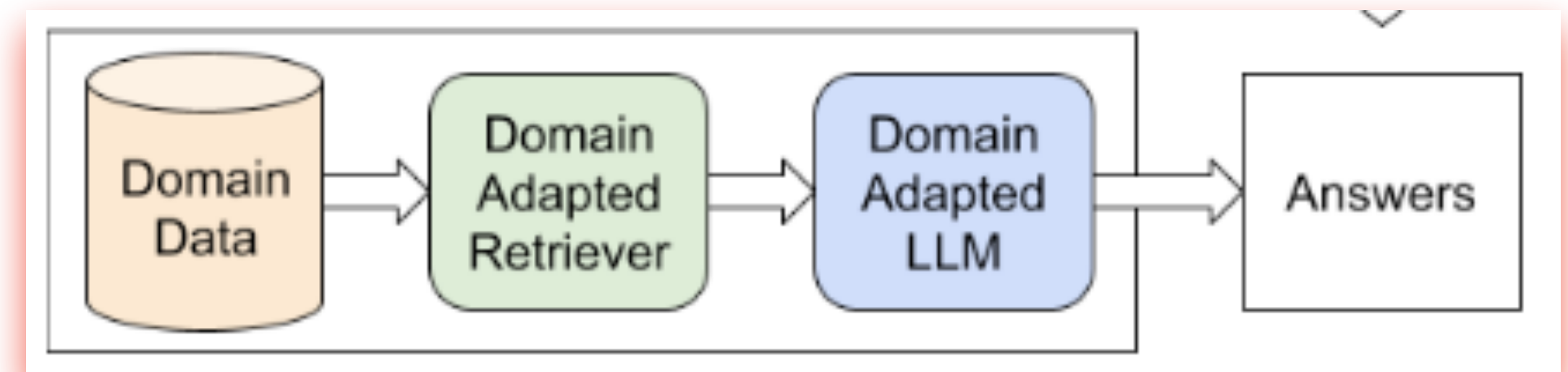


ISS: Their claim of ‘using domain specific embedding and LLM together improves the performance’ sounds interesting and needs to be checked

How chipnemo is used?

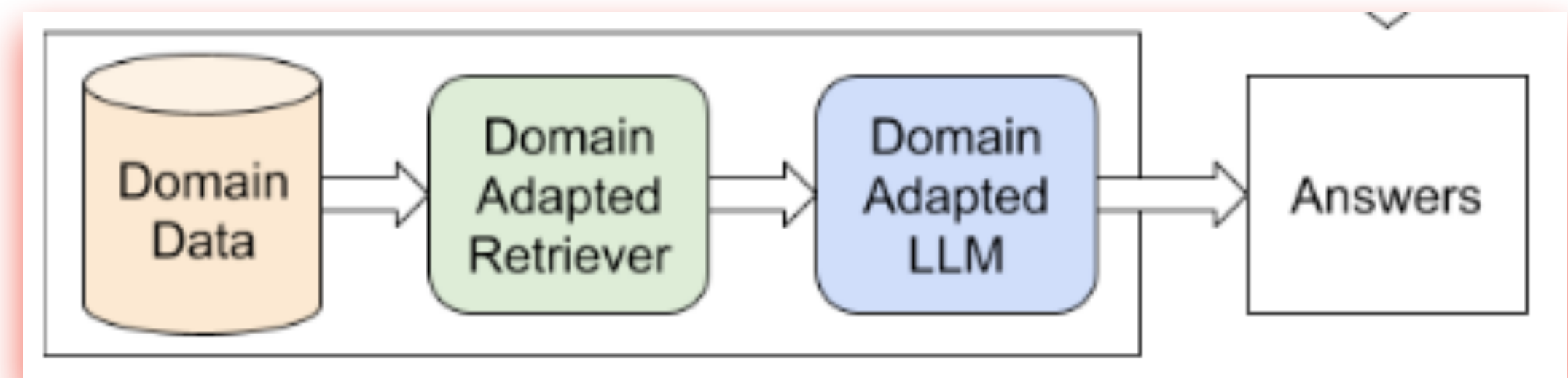
Areas are from survey

- Engineering assistant chatbot
 - RAG w/ customized embedding model
 - ChipNemo model
- EDA script generation
 - ChipNemo model
- BUG summarization and analysis
 - ChipNemo model



Engineering assistant chatbot

- This RAG tool based on the knowledge based on **any recorded data** about designs and communications could improve design productivity.
- Answer to architecture, design, verification, build questions

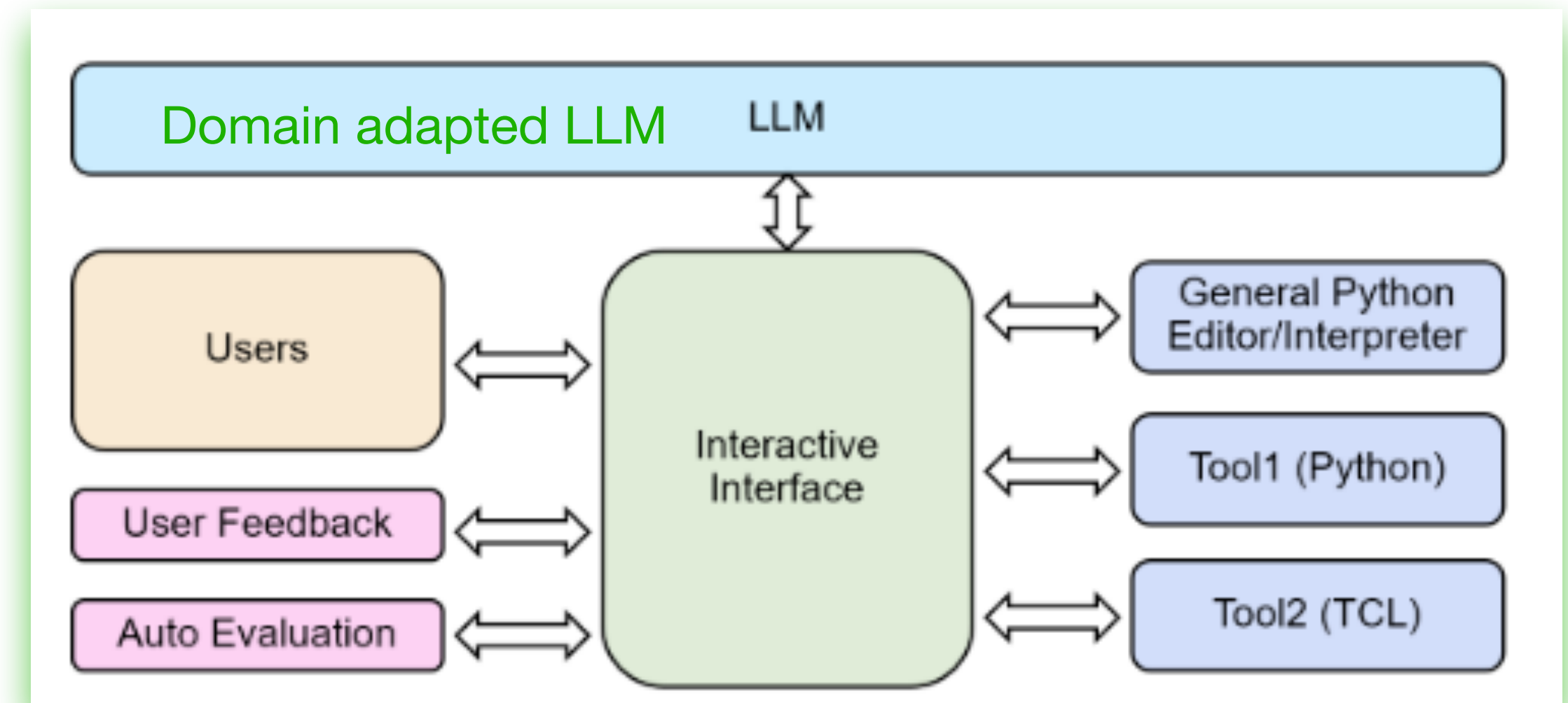


without impacting the productivity of others. It is observed that design engineers often enjoy brainstorming, designing hardware, and writing code, but can be slowed down waiting for answers on design knowledge they lack. Design productivity

unfamiliar with. Internal studies have shown that up to 60% of a typical chip designer's time is spent in debug or checklist related tasks across a range of topics including design specifications, testbench construction, architecture definition, and tools or infrastructure. Experts on these issues are often spread around

EDA script generation

- Learning/navigating/debugging custom scripting library is hard
- Generating small scale scripting from natural language task description
 - Tool1: internal python library for design editing and analysis
 - Tool2: Tcl scripts cli for industry static timing analysis tool
- They put *significant* effort to build the tool flow which includes feedback



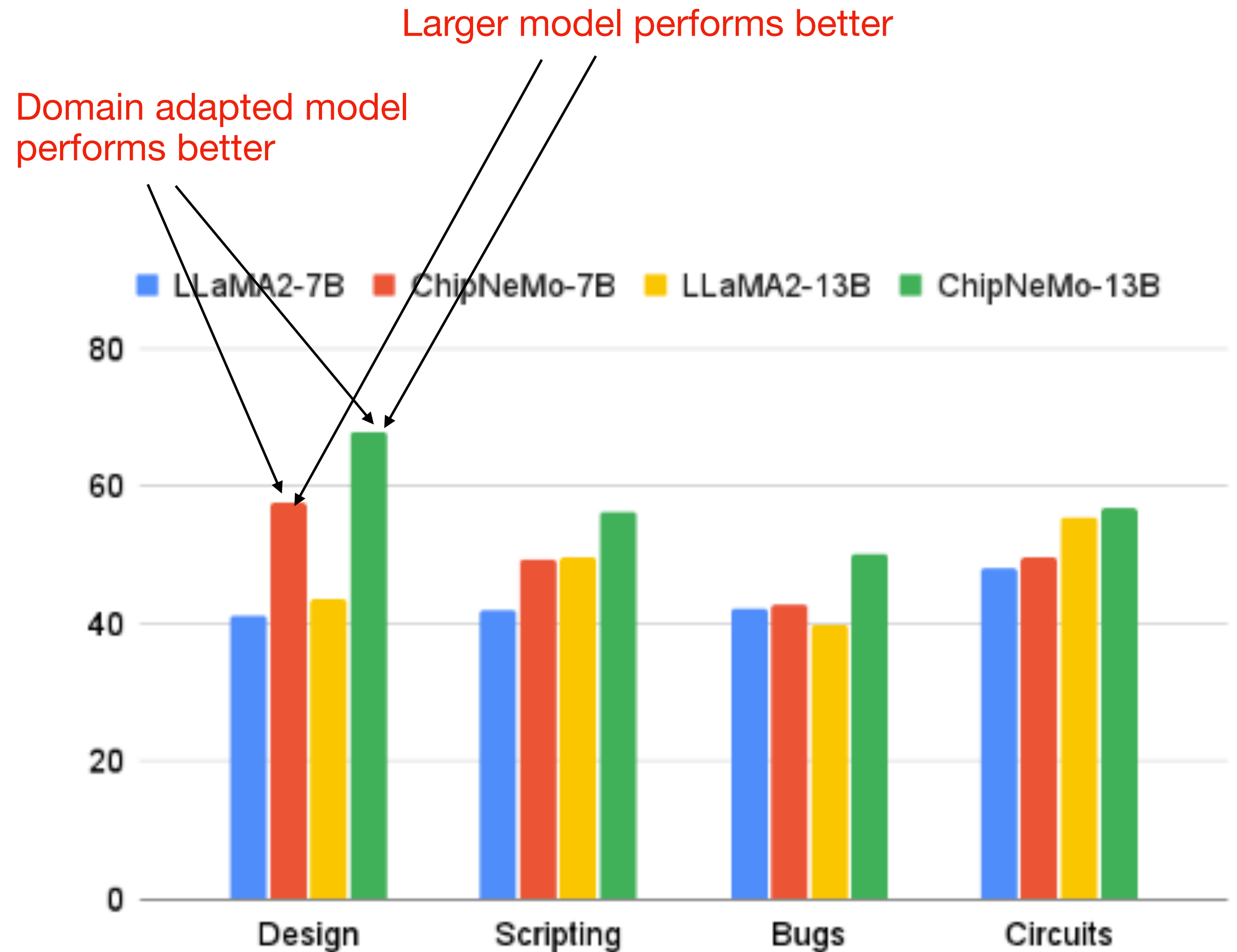
BUG summarization and analysis

- NVBugs: internal bug database
 - Bug database was part of the DAPT dataset
 - SFT dataset also includes bug example summary and task assignments
- It would be a great tool if
 - A tool that review all information and *summarize* both i) technical and ii) managerial data, and iii) *suggest* next steps
- Bug descriptions are large due to code and log dump, not fit to context window
- Solution to this:
 - Search/replace long path names with shorter aliases
 - Split the summarization task into an incremental task, so model accumulating data across chunks. i.e. hierarchical approach

ISS: Similar idea (chunk and hierarchical summarization) is used in DevOP (or AIOps) where it needs to analyze large amount of text

Model evaluation

AutoEval multi-choice

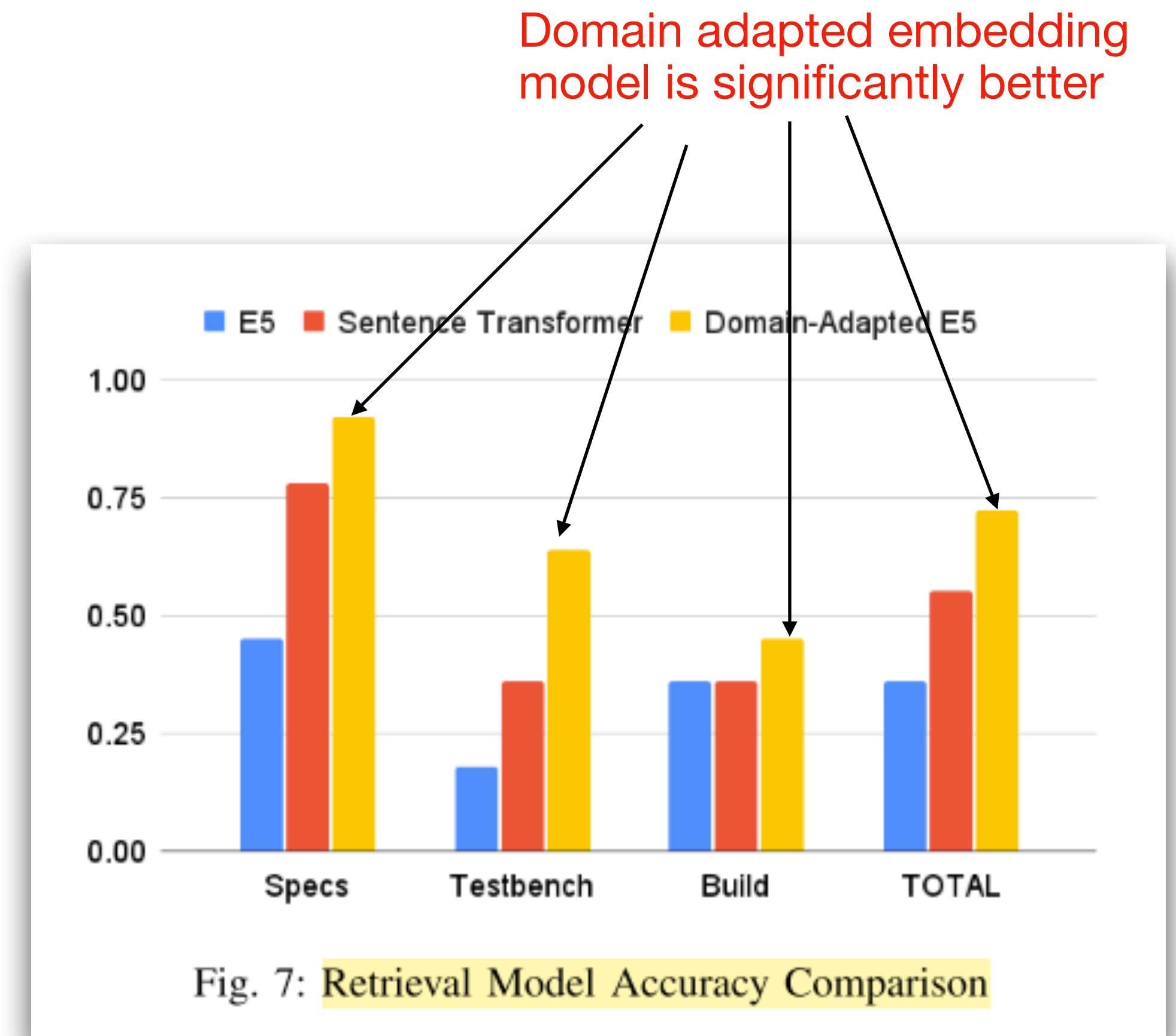


(a) Chip Design Domain Benchmarks.

RAG

Retriever

- RAG setup
 - 1.8K documents
 - 67K chunks, each with 512 characters
- Evaluation is based on test questions
- Created 88 questions from the following areas
 - architecture, design, verification specs
 - Test bench regression document
 - Build infrastructure document
- Designers created questions with golden answers and contexts



RAG based system

Human evaluation

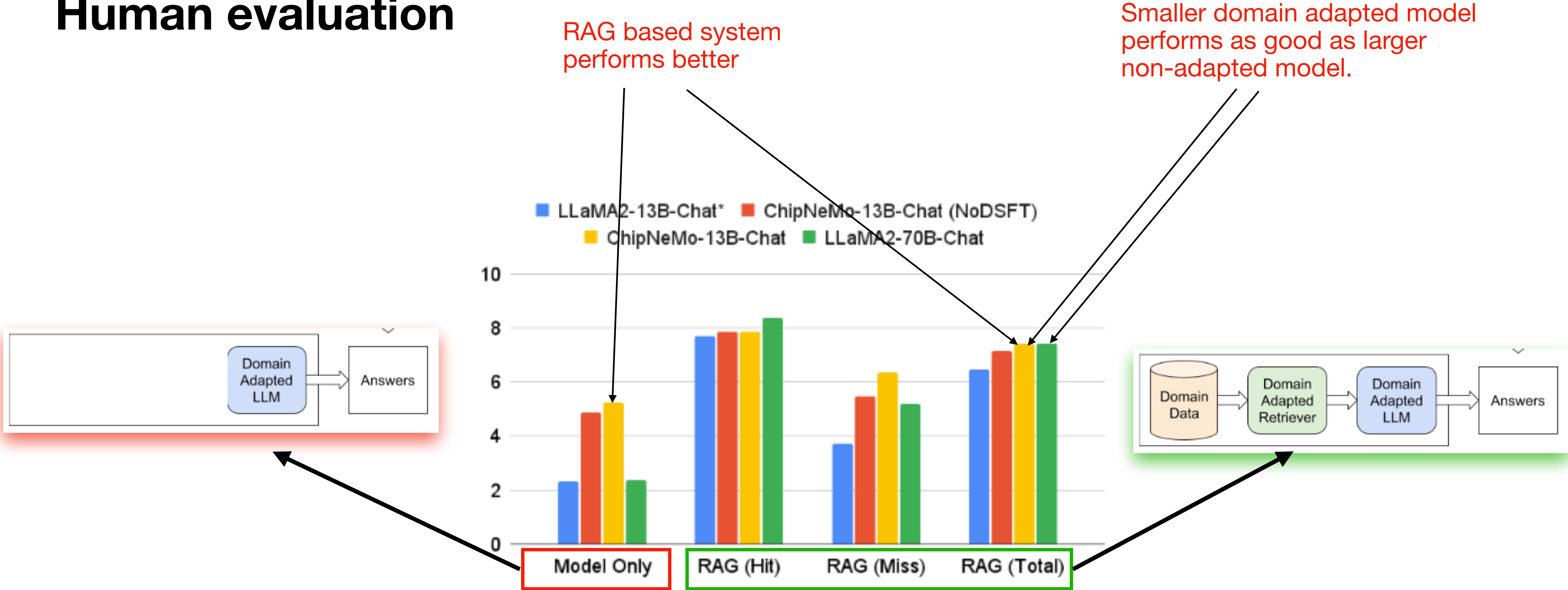


Fig. 8: Human Evaluation of Different Models. Model Only represents results without RAG. RAG (Hit)/(Miss) only include questions whose retrieved passages hit/miss their ideal context, RAG (Total) includes all questions.

Script generation

- Easy, medium: 1-2 line code
 - Large Original model performs poorly since it does not know how to use custom API compared to DAPT model
- Hard: real case
 - Questions are too hard, so they provided human curated context
 - Large original model performs better for python, since the large model is good at python and 'context' is provided so it will use it to answer
- Non-adapted large model with context for python code is better than smaller adapter model
 - Not for tcl: this is likely because tcl dataset for the original model is smaller than python code dataset

Domain adapted model performs significantly better

Domain SFT data helps

Providing context, non-adapted large model is still better than adapted small model in python

Providing context, small adapted model performs better than large model in Tcl

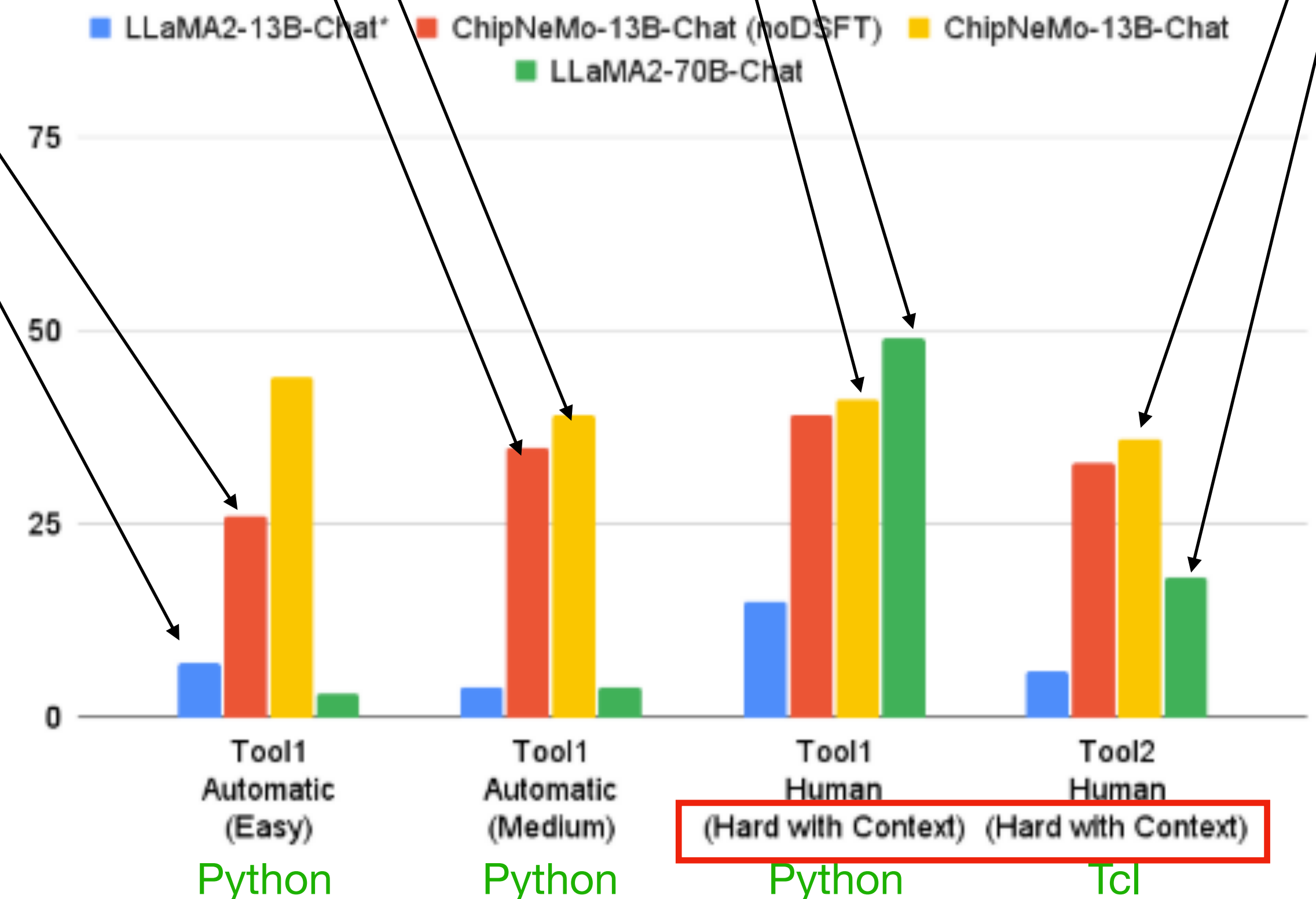


Fig. 9: EDA Script Generation Evaluation Results

Takeaway message

- Domain specific OSS LLM is performant enough to use in selected engineering tasks
- With RAG, a domain specific OSS LLM performs better than without RAG in both text and code
 - Using different document embedding store (vectordb) could further improve the performance
- OSS LLM is the only way to keep IP inside the organization
 - It natural to see why Nvidia relies on OSS LLM instead of OpenAI GPT
 - Both embedding and generation models are custom OSS LLMs
- A smaller models (7/13B) with domain adaptation pre-training performs as good or better than non-customized larger model (70B). This helps to save cost and reduce latency.
- Task specific instruction dataset for SFT helps to improve the results.
- LoRA significantly underperform in complex tasks (such as chip design) compared to full parameter fine-tuning

Will this paper win the test of time?

what's in it for us?

- Yes
- Ideas presented are not specific to specific company and can be used in other organization and in the future with some modifications

Question

- Why the model only used for script generation? And not used for generating RTL and sw code.
 - It maybe they are testing their model to the relatively easier and lower risk tasks first???
 - Then later, it is expected that their model can be extended to RTL and SW code generation???
 - Maybe, they are already doing it, in addition to what's described in the paper???

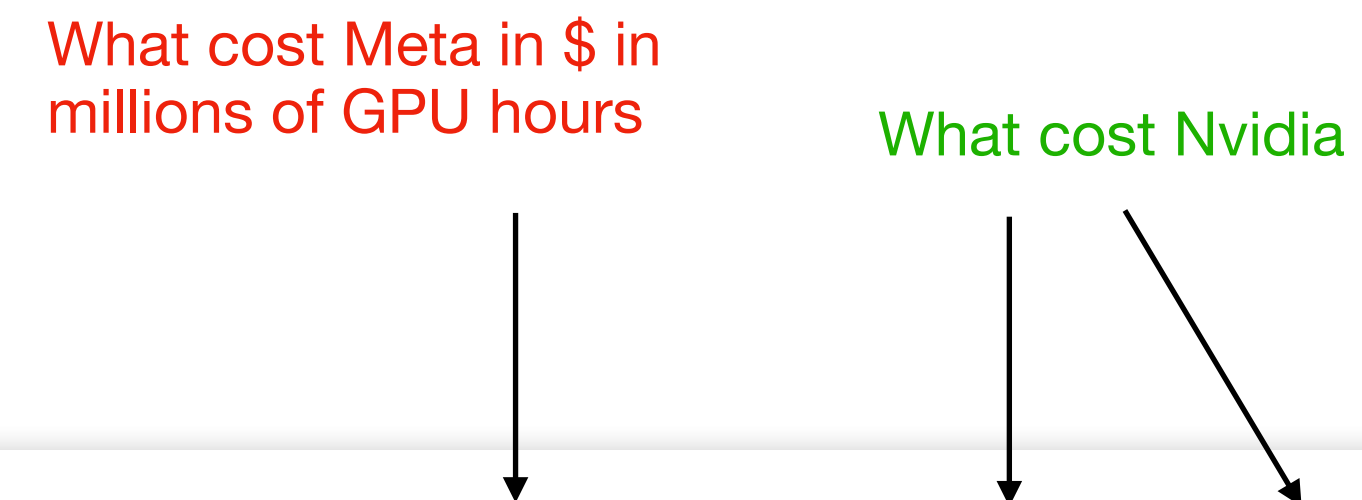
Backup

Training cost

- 128 A100 GPUs
- Few thousand GPU hours

What cost Meta in \$ in
millions of GPU hours

What cost Nvidia



Model Size	Pretraining	DAPT	SFT
7B	184,320	2,620	90
13B	368,640	4,940	160
70B	1,720,320	-	-

TABLE IV: Training cost of LLaMA2 models in GPU hours. Pretraining cost from [5].

Embedding

- Fine tune embedding model, e5_small_unsupervised
- 300 domain specific autogenerated samples