The Llama 3 Herd of Models

(Part 1)

Aditya S and Aditya C

Meta

Agenda

1. Introduction
2. Pre-Training
3. Model Architecture
4. Training Infrastructure
5. Post-Training
6. Data Processing
7. Capability Enhancements

Intro and Improvements

Stages:

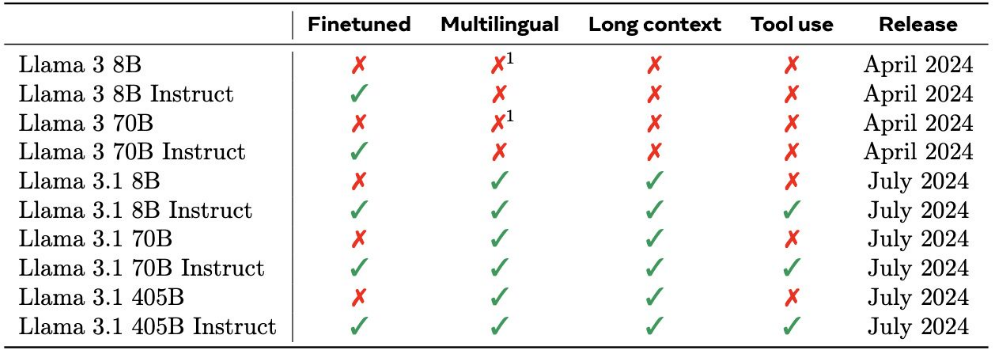
- Pre-training (next-word prediction, captioning)
- Post-training (instructions, alignment, capabilities)

Big 3 Levers:

- Data (15T multilingual token vs 2T)
- Scale (3.8 × 10^25 FLOPs, 50x larger)
- Complexity Management (Dense Transformer > mixture-of-experts)

End result:

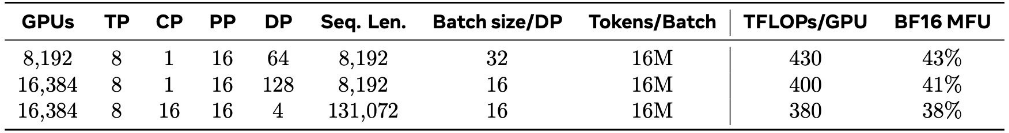
- 405B trained on 15.6T
- Solve complex reasoning problems
  Multilingual (8 languages)
- Tools-out-the-box / Zero-shot



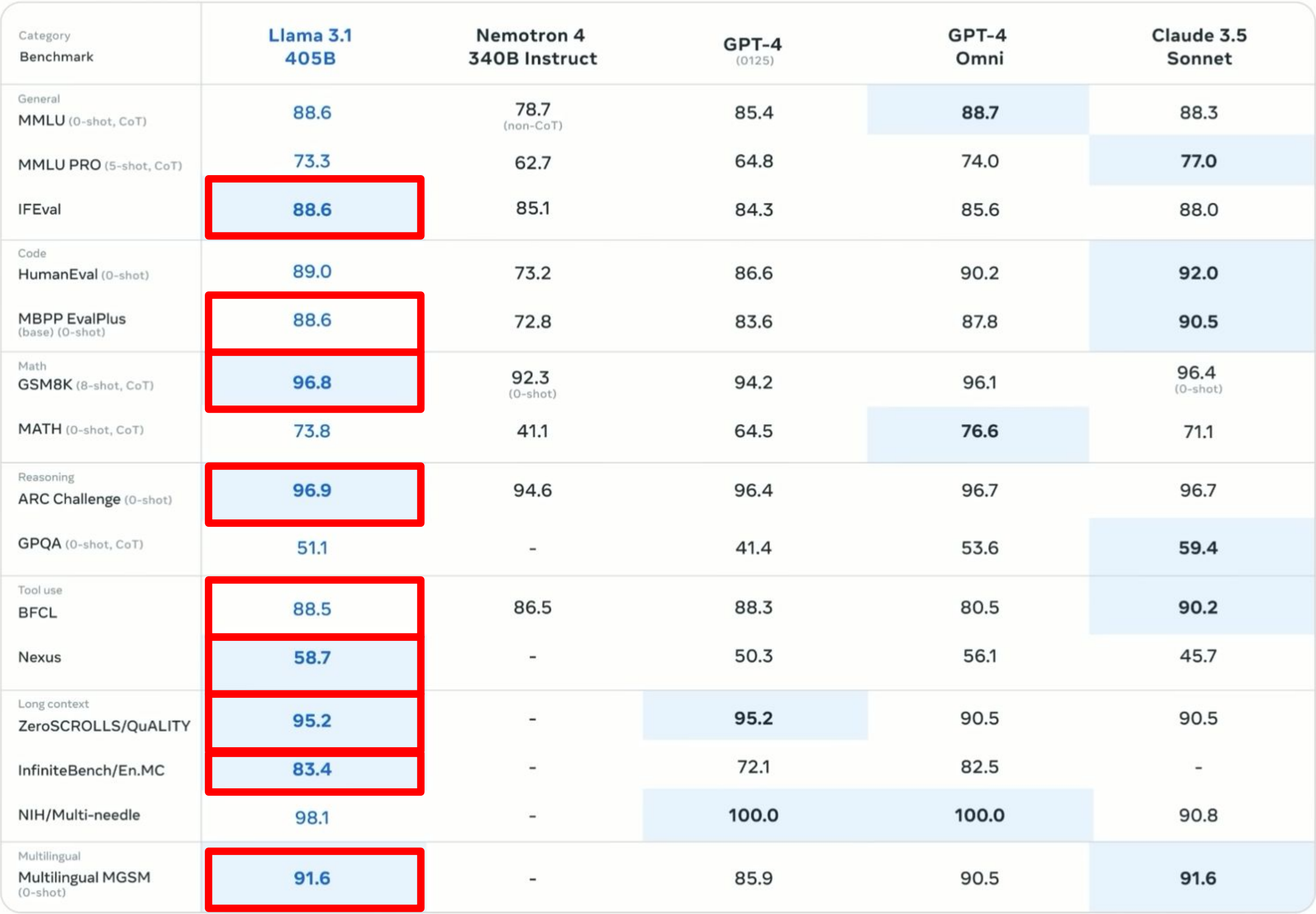
	Finetuned	**Multilingual**	**Long context**	**Tool use**	**Release**
Llama 3 8B	✗	✗[1]	✗	✗	April 2024
Llama 3 8B Instruct	✓	✗	✗	✗	April 2024
Llama 3 70B	✗	✗[1]	✗	✗	April 2024
Llama 3 70B Instruct	✓	✗	✗	✗	April 2024
Llama 3.1 8B	✗	✓	✓	✗	July 2024
Llama 3.1 8B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 70B	✗	✓	✓	✗	July 2024
Llama 3.1 70B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 405B	✗	✓	✓	✗	July 2024
Llama 3.1 405B Instruct	✓	✓	✓	✓	July 2024

More Improvements (LLama3 Updates)

- Data Scale
    - 7x larger
    - 30+ languages
- Scaling Laws (Optimal data mix and Informed Decision making)
- Optimization Techniques:
    - Parallelization (Data, Model, Pipeline)
    - Detection for silent data corruption
    - Scalable storage systems
- Model Differences:
    - Larger context window
    - No switch to MoE

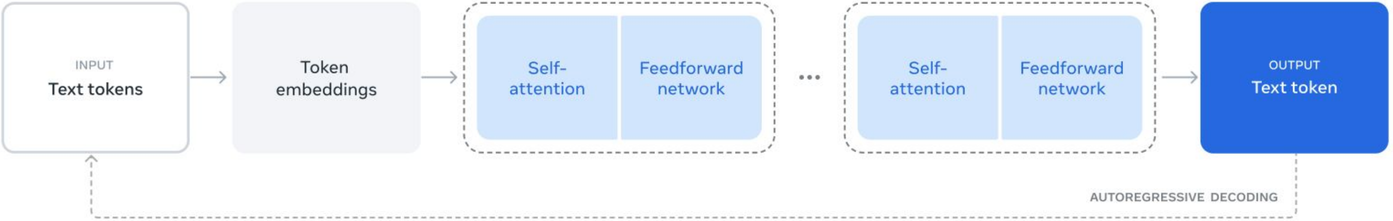


GPUs	**TP**	**CP**	**PP**	**DP**	**Seq. Len.**	**Batch size/DP**	**Tokens/Batch**	**TFLOPs/GPU**	**BF16 MFU**
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	4	131,072	16	16M	380	38%



Category Benchmark	Llama 3.1 405B	Nemotron 4 340B Instruct	GPT-4 (0125)	GPT-4 Omni	Claude 3.5 Sonnet
General MMLU (0-shot, CoT)	88.6	78.7 (non-CoT)	85.4	**88.7**	88.3
MMLU PRO (5-shot, CoT)	73.3	62.7	64.8	74.0	**77.0**
IFEval	**88.6**	85.1	84.3	85.6	88.0
Code HumanEval (0-shot)	89.0	73.2	86.6	90.2	**92.0**
MBPP EvalPlus (base) (0-shot)	88.6	72.8	83.6	87.8	**90.5**
Math GSM8K (8-shot, CoT)	**96.8**	92.3 (0-shot)	94.2	96.1	96.4 (0-shot)
MATH (0-shot, CoT)	73.8	41.1	64.5	**76.6**	71.1
Reasoning ARC Challenge (0-shot)	**96.9**	94.6	96.4	96.7	96.7
GPQA (0-shot, CoT)	51.1	-	41.4	53.6	**59.4**
Tool use BFCL	88.5	86.5	88.3	80.5	**90.2**
Nexus	**58.7**	-	50.3	56.1	45.7
Long context ZeroSCROLLS/QuALITY	**95.2**	-	**95.2**	90.5	90.5
InfiniteBench/En.MC	**83.4**	-	72.1	82.5	-
NIH/Multi-needle	98.1	-	**100.0**	**100.0**	90.8
Multilingual Multilingual MGSM (0-shot)	**91.6**	-	85.9	90.5	**91.6**

Overview



Pre-training:

- Multilingual text corpus -> discrete tokens
- Pre train to perform next-token prediction
- Objective: learn language structure/obtain knowledge about the world

Post-training:

- Several rounds of SFT and DPO
- Integrate new capabilities (tool-use, improved coding and reasoning)
- Safety mitigations

Vision/Speech Adapter Preview

Vision:

- Cross-attention-based adapter integrates a pre-trained image encoder uses:
  - aligning image
  - language representations
- Trained on text-image pairs, updating only image encoder

Speech:

- Convert speech encodings into token representations
- Parameters updated in SFT
- Language Model remains unchanged

Pre-Training

Components:

1) Curation and filtering of large-scale training corpus
2) Development of model architecture and scaling laws
3) Techniques for efficient large-scale pre-training
4) Development of a “pre-training recipe”

Pre-Training Data Curation and Filtering

Web Data Curation:

- PII and safety filtering
- Text extraction and cleaning
- De-duplication
- Heuristic filtering (n-gram coverage ratio, “dirty word”)

Fast Classifiers:

- Fasttext
- Roberta-based classifiers trained on Llama-2 predictions

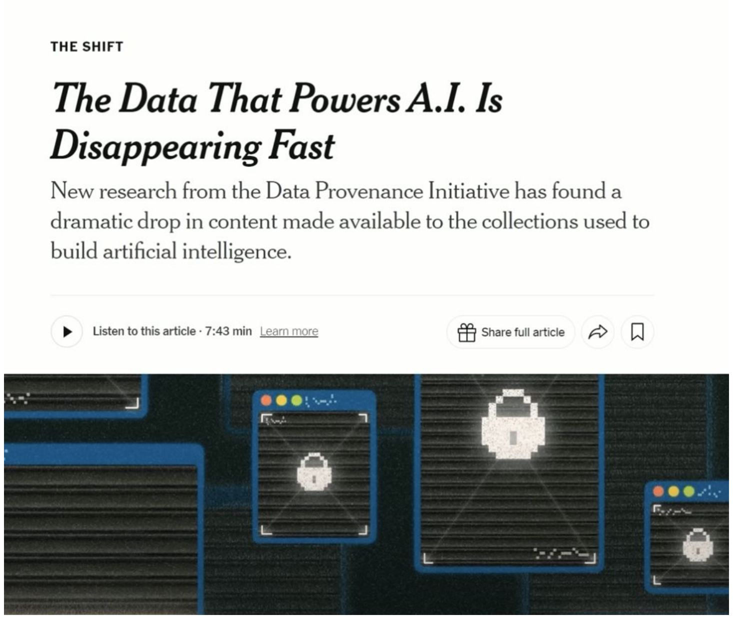
Data Mix and Discussion

Token Breakdown:

- 50% general knowledge,
- 25% mathematical and reasoning
- 17% code
- 8% multilingual tokens

Data “Transparency” and collection

- Meta is committed to open-source but generally shy about data
- NY Times article on availability of data
    - Reddit
    - Twitter



THE SHIFT

The Data That Powers A.I. Is Disappearing Fast

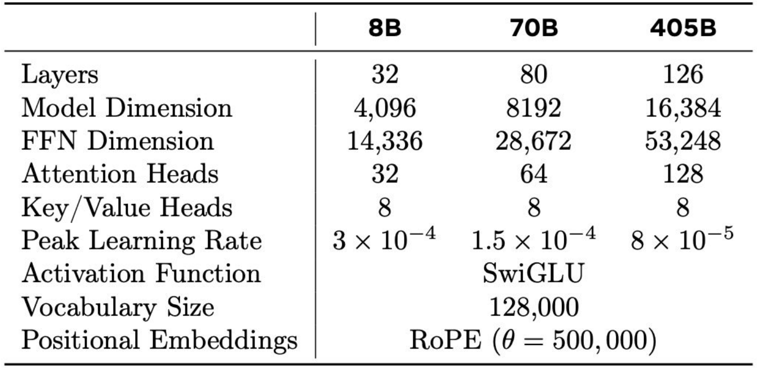
New research from the Data Provenance Initiative has found a dramatic drop in content made available to the collections used to build artificial intelligence.

Annealing Data and Dataset Evaluation

- Annealing small amounts of high-quality code/mathematical data boosts performance of pre-trained models on:
  - GSM8k (24.0%)
  - MATH Validation sets (6.4%)
- Improvements on 405B model were negligible
- Flagship model has strong
  - In-context learning
  - Reasoning capabilities
- Specific in-domain training samples aren’t needed for strong performance

Model Architecture

- Dense Transformer architecture
- Performance gains driven by improvements in:
  - Data quality
  - Diversity
  - Increased training scale
- Grouped Query Attention
  - Inference speed
  - Reduce key-value cache sizes
- Attention Mask
- Vocabulary with 128K tokens
- Training budget of 3.8 × 10^25 FLOPs



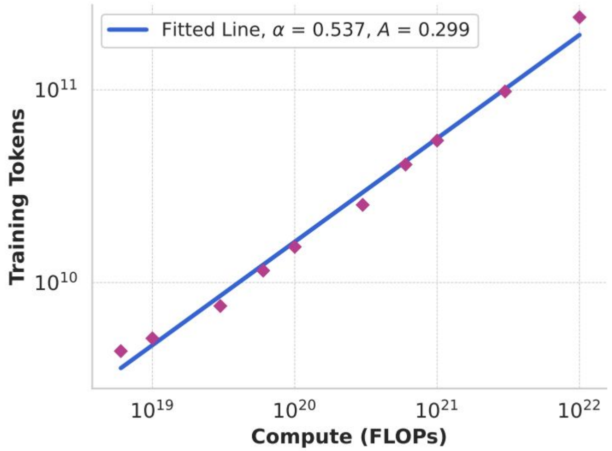
	8B	**70B**	**405B**
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

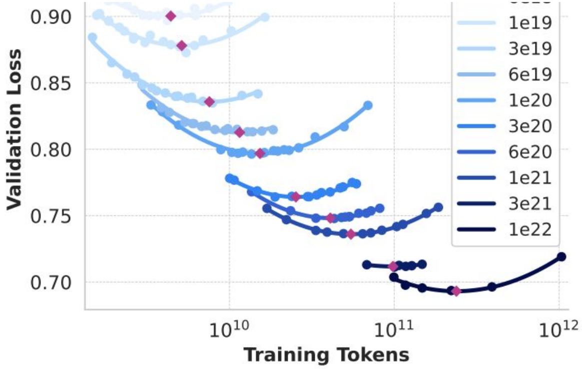
Scaling Laws

- Determine the optimal model size given pre-training compute budget
- Issues with current laws:
  - Predict next-token prediction loss, not specific benchmark performance
  - Noisy/Unreliable due to small compute budgets
- Methodology:
  1) Correlate compute-optimal model's negative log-likelihood on downstream tasks and the training FLOPs
  2) Correlate negative log-likelihood on downstream tasks with task accuracy

Scaling Law Experiments

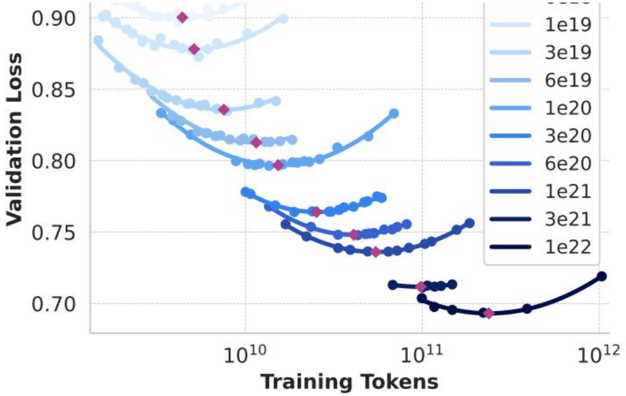
- Extrapolate across four orders of magnitude
- Model sizes ranging from 40M to 16B parameters, optimizing learning rate and weight decay
- IsoFLOPs Curves: Compute-optimal model determined by finding minimums in polynomial-fitted validation loss curves, showing robustness to trade-offs in model size and training tokens





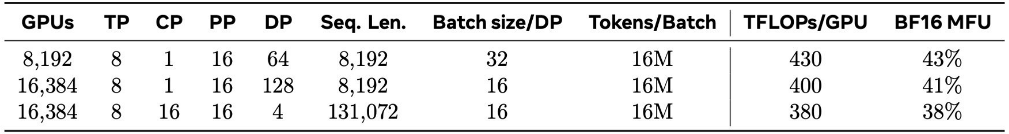
Scaling Law Experiments

- At each compute budget, we pre-train models
- Cosine learning rate schedule, linear warmup for 2,000 training steps
- Cosine decay: 0.1 of the peak value
- Weight decay: 0.1 times learning rate at step
- Fixed batch size: 250K to 4M
- minimum of a parabola
  - compute-optimal model
  - predict the optimal number of training tokens for a compute budget



$$N^\star(C) = AC^\alpha.$$

Infrastructure and Efficiency



GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	4	131,072	16	16M	380	38%

Compute infrastructure:

- 16K H100 GPUs (700W with 80GB HBM3)
- Meta's Grand Teton AI server platform

Storage:

- Tectonic distributed file system provides (240 PB of storage across 7,500 servers)
- 7 TBs throughput

Checkpointing:

- Each GPU's model state (1 MB to 4 GB)
- Reduce GPU pause time and increase checkpoint frequency

Network:

- RDMA over Converged Ethernet (RoCE) fabric with 400 Gbps interconnects
- Network design: The RoCE network is fully owned and designed by Meta for large-scale AI workloads.

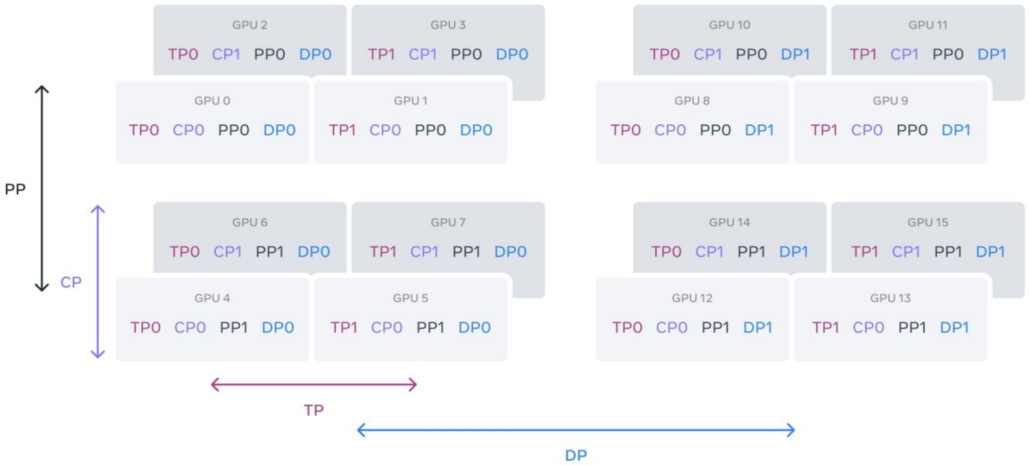
Parallelism

TP: weight tensors

PP: partitions the model vertically into stages by layers

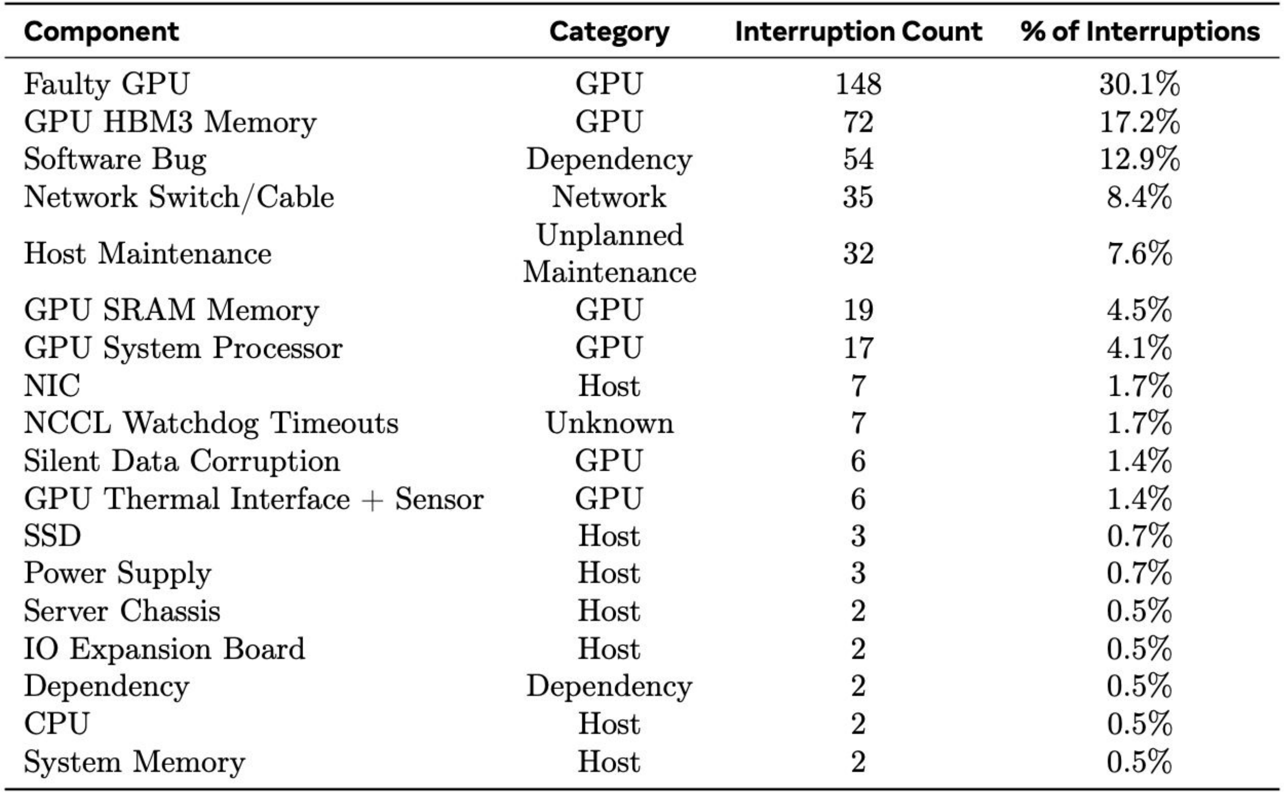
CP: divides the input context into segments

DP: shards the model, optimizer, and gradients while implementing data parallelism which processes data in parallel on multiple GPUs



GPU 2 TP0 CP1 PP0 DP0
GPU 3 TP1 CP1 PP0 DP0
GPU 10 TP0 CP1 PP0 DP1
GPU 11 TP1 CP1 PP0 DP1
GPU 0 TP0 CP0 PP0 DP0
GPU 1 TP1 CP0 PP0 DP0
GPU 8 TP0 CP0 PP0 DP1
GPU 9 TP1 CP0 PP0 DP1
GPU 6 TP0 CP1 PP1 DP0
GPU 7 TP1 CP1 PP1 DP0
GPU 14 TP0 CP1 PP1 DP1
GPU 15 TP1 CP1 PP1 DP1
GPU 4 TP0 CP0 PP1 DP0
GPU 5 TP1 CP0 PP1 DP0
GPU 12 TP0 CP0 PP1 DP1
GPU 13 TP1 CP0 PP1 DP1

Operational Challenges



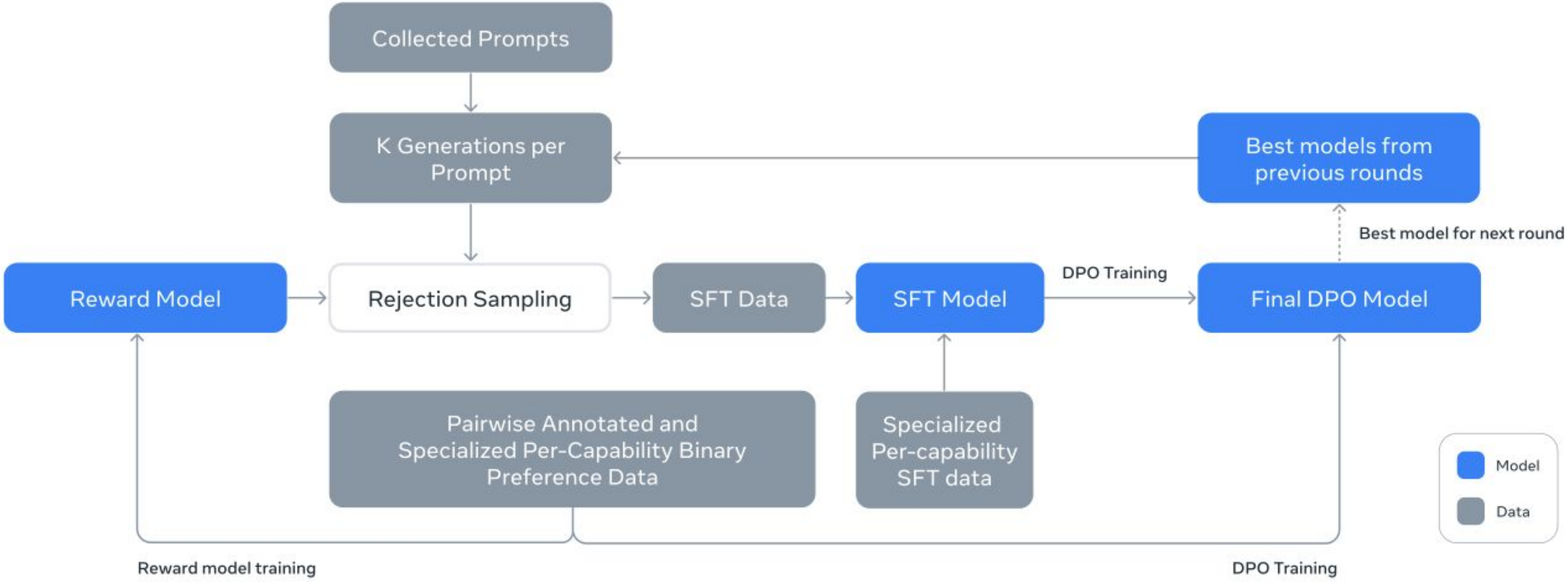
Component	Category	Interruption Count	% of Interruptions
Faulty GPU	GPU	148	30.1%
GPU HBM3 Memory	GPU	72	17.2%
Software Bug	Dependency	54	12.9%
Network Switch/Cable	Network	35	8.4%
Host Maintenance	Unplanned Maintenance	32	7.6%
GPU SRAM Memory	GPU	19	4.5%
GPU System Processor	GPU	17	4.1%
NIC	Host	7	1.7%
NCCL Watchdog Timeouts	Unknown	7	1.7%
Silent Data Corruption	GPU	6	1.4%
GPU Thermal Interface + Sensor	GPU	6	1.4%
SSD	Host	3	0.7%
Power Supply	Host	3	0.7%
Server Chassis	Host	2	0.5%
IO Expansion Board	Host	2	0.5%
Dependency	Dependency	2	0.5%
CPU	Host	2	0.5%
System Memory	Host	2	0.5%

“Training Recipe”

1) Initial Pre-Training
    - AdamW optimizer, peak learning rate: 8×10^{-5}
    - Gradual increase in batch size for stability: 4M tokens to 16M tokens
    - Dynamic data adjustments: More non-English data, mathematical reasoning, up-to-date web content
2) Long-Context Pre-Training
    - Gradually increased context window from 8K to 128K tokens
    - Six stages of context adaptation, totaling 800B training tokens
    - Goal: Maintain short-context performance and solve complex tasks
3) Annealing Phase
    - Linear reduction of learning rate to 0 during final 40M tokens
    - Upsampling high-quality data sources

Main Takeaway: A multi-stage, stable training approach combining careful batch sizing, long-context adaptation, and annealing ensures optimal model performance.

Post-Training Overview


Collected Prompts
K Generations per Prompt
Best models from previous rounds
Best model for next round
Reward Model
Rejection Sampling
SFT Data
SFT Model
DPO Training
Final DPO Model
Pairwise Annotated and Specialized Per-Capability Binary Preference Data
Specialized Per-capability SFT data
Model
Data
Reward model training
DPO Training

Key Components

- Reward Model
  - Use ranking loss to train $\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r)))$
  - Used to perform rejection sampling for SFT data

Key Components

- Reward Model
    - Use ranking loss to train $\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r)))$
    - Used to perform rejection sampling for SFT data
- SFT
    - Standard cross-entropy loss
    - Mostly synthetic data

Key Components

- Reward Model
    - Use ranking loss to train $\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r)))$
    - Used to perform rejection sampling for SFT data
- SFT
    - Standard cross-entropy loss
    - Mostly synthetic data
- DPO
    - More computationally efficient and better instruction following than PPO
    - Mask header & termination tokens
    - Added NLL loss on the chosen responses

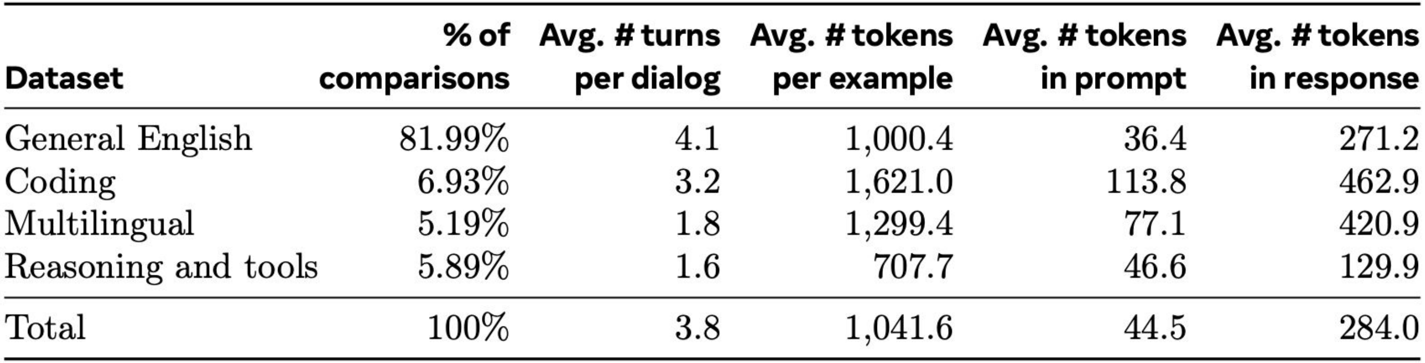
DPO Preference Data

- Collect responses from previous models for each prompt
- Annotators choose preferred response & categorize into 4 levels
  - Significantly better, better, slightly better, or marginally better
  - *Significantly better* and *better* responses chosen
- Annotators can edit chosen response to further improve it
  - Preference order is *edited > chosen > rejected*
- Use data generated from most recent batch for DPO

DPO Preference Data

- Collect responses from previous models for each prompt
- Annotators choose preferred response & categorize into 4 levels
  - Significantly better, better, slightly better, or marginally better
  - *Significantly better* and *better* responses chosen
- Annotators can edit chosen response to further improve it
  - Preference order is *edited > chosen > rejected*
- Use data generated from most recent batch for DPO
  - **Discussion: What are some advantages of adding this constraint? Disadvantages?**

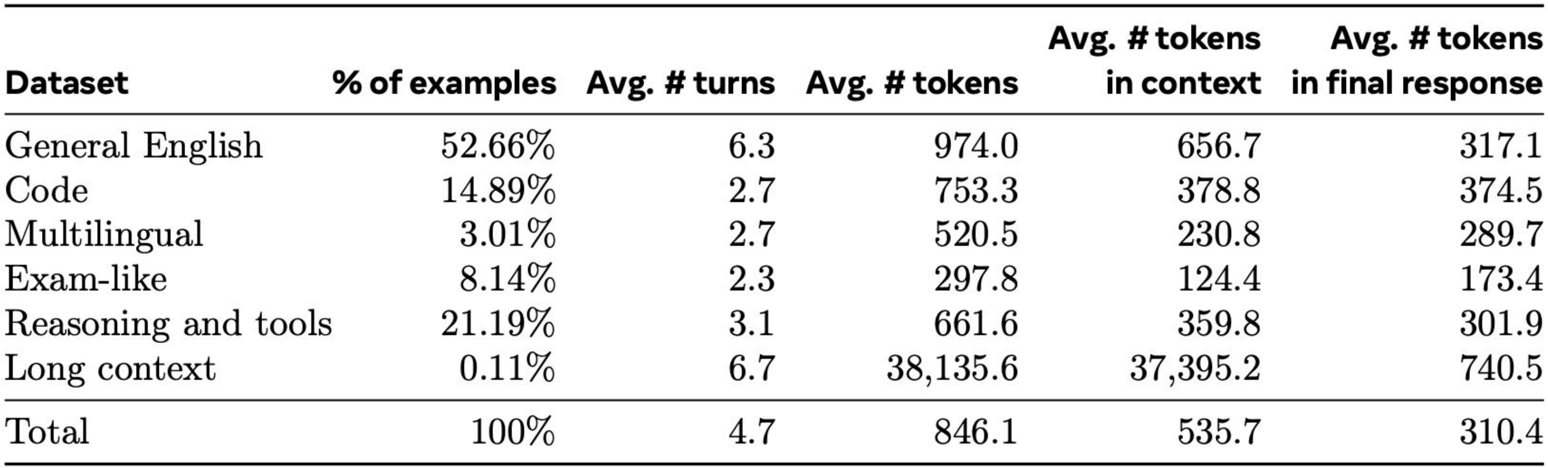
DPO Preference Data



Dataset	% of comparisons	Avg. # turns per dialog	Avg. # tokens per example	Avg. # tokens in prompt	Avg. # tokens in response
General English	81.99%	4.1	1,000.4	36.4	271.2
Coding	6.93%	3.2	1,621.0	113.8	462.9
Multilingual	5.19%	1.8	1,299.4	77.1	420.9
Reasoning and tools	5.89%	1.6	707.7	46.6	129.9
Total	100%	3.8	1,041.6	44.5	284.0

- Length of prompt & response increased → more complex tasks
- Perform rigorous quality control & evaluation for in-training modifications
  - Ex. Increase prompt complexity in low-performing areas

SFT Data



Dataset	% of examples	Avg. # turns	Avg. # tokens	Avg. # tokens in context	Avg. # tokens in final response
General English	52.66%	6.3	974.0	656.7	317.1
Code	14.89%	2.7	753.3	378.8	374.5
Multilingual	3.01%	2.7	520.5	230.8	289.7
Exam-like	8.14%	2.3	297.8	124.4	173.4
Reasoning and tools	21.19%	3.1	661.6	359.8	301.9
Long context	0.11%	6.7	38,135.6	37,395.2	740.5
Total	100%	4.7	846.1	535.7	310.4

Comprised of:

- Human-written prompts & model responses with rejection-sampling
- Synthetic & human-annotated data targeting capabilities

Data Processing

- **Rule-based removal and modification** (emojis, overused phrases)

Data Processing

- **Rule-based removal and modification** (emojis, overused phrases)
- **Topic classification**: fine-tuned Llama3-8B

Data Processing

- **Rule-based removal and modification** (emojis, overused phrases)
- **Topic classification**: fine-tuned Llama3-8B
- **Quality scoring**: RM, Llama3-405B checkpoint
  - Top quartile of RM scores
  - Llama3 ratings - 3 point scale for accuracy, instruction following, and tone/presentation; 2 point scale for bug identification and user intention

Data Processing

- **Rule-based removal and modification** (emojis, overused phrases)
- **Topic classification**: fine-tuned Llama3-8B
- **Quality scoring**: RM, Llama3-405B checkpoint
- **Difficulty scoring:** Instag using Llama3-70B, Llama3-405B ratings
  - Instag: measures intentions, more intentions → higher complexity
  - 3 point scale for Llama3 difficulty ratings

Data Processing

- **Rule-based removal and modification** (emojis, overused phrases)
- **Topic classification**: fine-tuned Llama3-8B
- **Quality scoring**: RM, Llama3-405B checkpoint
- **Difficulty scoring:** Instag using Llama3-70B, Llama3-405B ratings
- **Semantic deduplication**: Sort dialogues by *Quality* x *Difficulty*, keep dialogues with low cosine similarity

Capability Specific Data

- Extraction of desired capabilities is stronger with data targeting those tasks

Capability Specific Data

- Extraction of desired capabilities is stronger with data targeting those tasks
- For Llama3 these are:
    - Coding
    - Multilinguality
    - Math and Reasoning
    - Long Context Adaptability
    - Tool Use
    - Factuality
    - Steerability

Coding

- Train *coding expert* model by continuing pre-training of checkpoint
  - 1T tokens, approx 85% code data
  - Apply SFT and DPO to expert with code data

Coding

- Train *coding expert* model by continuing pre-training of checkpoint
  - 1T tokens, approx 85% code data
  - Apply SFT and DPO to expert with code data
- 3 approaches for synthetic data generation:
  - Execution Feedback
  - Programming Language Translation
  - Backtranslation

Execution Feedback

1. Generate programming problems by prompting the model with example code snippets

Execution Feedback

1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments

Execution Feedback

1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments
3. Verify correctness of solution via parser + linter and model-generated unit tests

Execution Feedback

1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments
3. Verify correctness of solution via parser + linter and model-generated unit tests
4. On syntax or operational error, prompt model to fix code or unit tests
    a. ~20% of solutions corrected

Execution Feedback

1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments
3. Verify correctness of solution via parser + linter and model-generated unit tests
4. On syntax or operational error, prompt model to fix code or unit tests
   1. ~20% of solutions corrected
5. Accept response when it passes all checks

Execution Feedback

1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments
3. Verify correctness of solution via parser + linter and model-generated unit tests
4. On syntax or operational error, prompt model to fix code or unit tests
    a. ~20% of solutions corrected
5. Accept response when it passes all checks
6. Repeat process with model generating new data each round

Execution Feedback

Any issues?

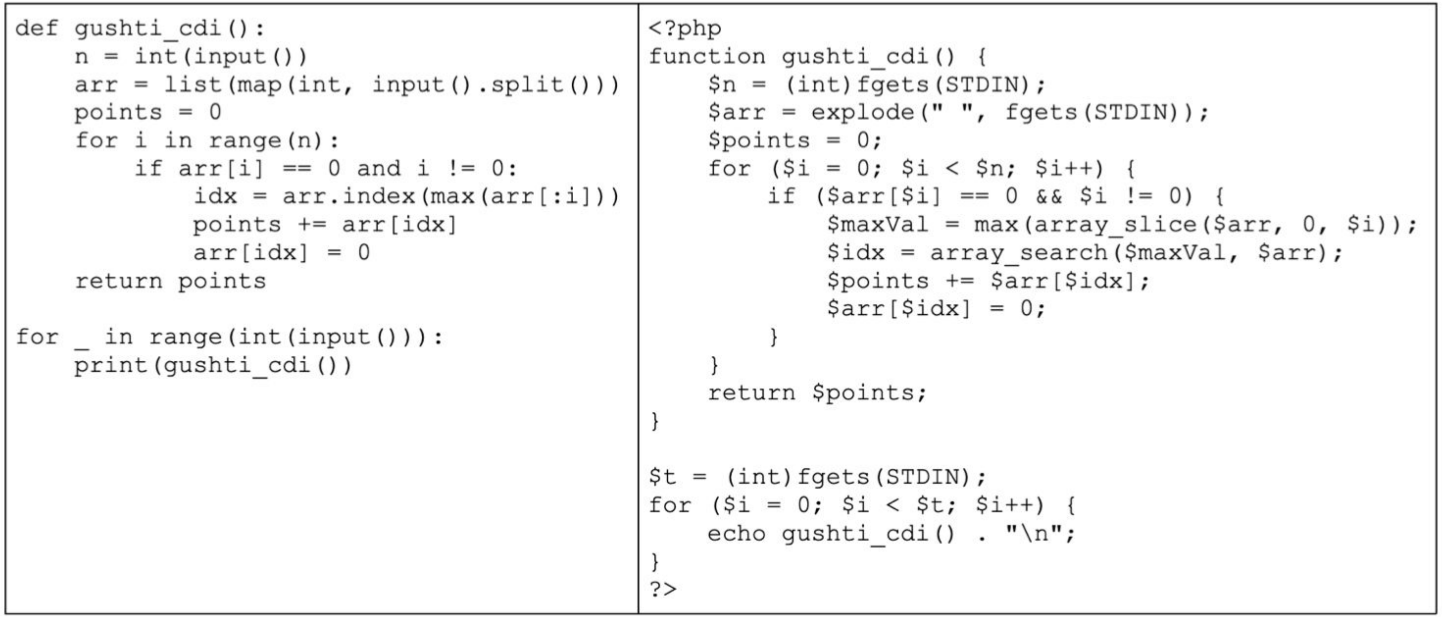
1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments
3. Verify correctness of solution via parser + linter and model-generated unit tests
4. On syntax or operational error, prompt model to fix code or unit tests
    a. ~20% of solutions corrected
5. Accept response when it passes all checks
6. Repeat process with model generating new data each round

Execution Feedback

Any issues?

1. Generate programming problems by prompting the model with example code snippets
2. Prompt model to solve coding problem and include thought process in comments
3. Verify correctness of solution via parser + linter and model-generated unit tests
4. On syntax or operational error, prompt model to fix code or unit tests
   a. ~20% of solutions corrected
5. Accept response when it passes all checks
6. Repeat process with model generating new data each round

Programming Language Translation



```python
def gushti_cdi():
    n = int(input())
    arr = list(map(int, input().split()))
    points = 0
    for i in range(n):
        if arr[i] == 0 and i != 0:
            idx = arr.index(max(arr[:i]))
            points += arr[idx]
            arr[idx] = 0
    return points

for _ in range(int(input())):
    print(gushti_cdi())
```

```php
<?php
function gushti_cdi() {
    $n = (int)fgets(STDIN);
    $arr = explode(" ", fgets(STDIN));
    $points = 0;
    for ($i = 0; $i < $n; $i++) {
        if ($arr[$i] == 0 && $i != 0) {
            $maxVal = max(array_slice($arr, 0, $i));
            $idx = array_search($maxVal, $arr);
            $points += $arr[$idx];
            $arr[$idx] = 0;
        }
    }
    return $points;
}

$t = (int)fgets(STDIN);
for ($i = 0; $i < $t; $i++) {
    echo gushti_cdi() . "\n";
}
?>
```

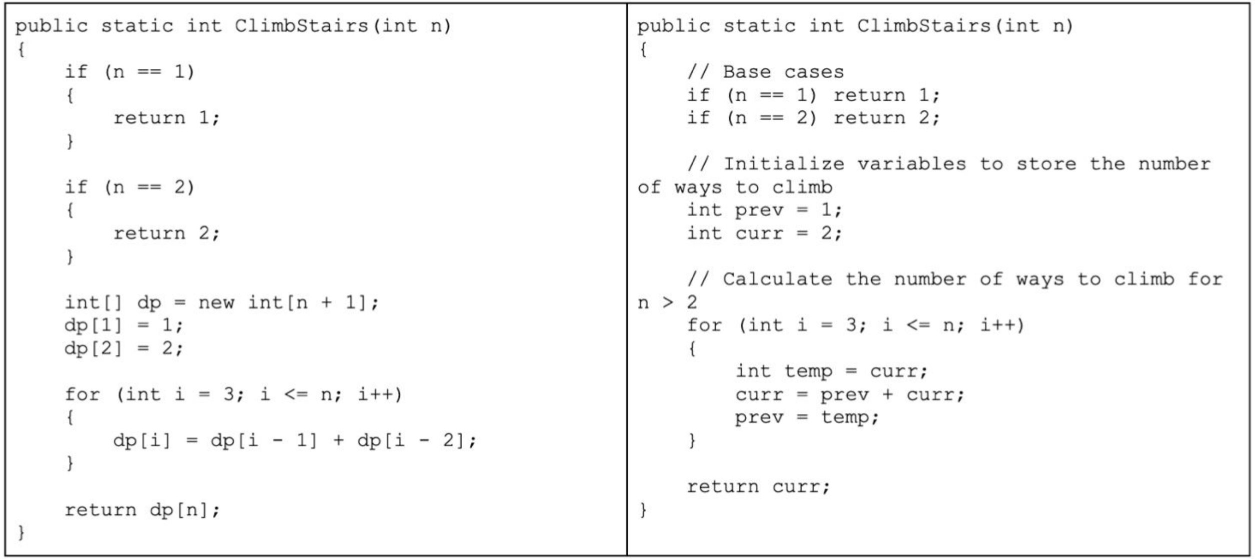
- Translation from Python to PHP
- Quality verified by syntax parsing, compiling code, and executing on tests

Backtranslation

- Primarily for documentation, explanation, debugging
- Process:
    1. Generate *target* data from code snippet (comments on code, explanation for code)
    2. Prompt model to backtranslate data to the original code
    3. Verify quality of backtranslation using original code
        a. High-scoring samples (backtranslation, generated data) are kept

Data Quality

- Adding steering system prompt during rejection sampling boosts generation quality
- Use model-as-a-judge to assess synthetic data
    - Some challenging prompts fail → manually generate samples



```
public static int ClimbStairs(int n)
{
    if (n == 1)
    {
        return 1;
    }

    if (n == 2)
    {
        return 2;
    }

    int[] dp = new int[n + 1];
    dp[1] = 1;
    dp[2] = 2;

    for (int i = 3; i <= n; i++)
    {
        dp[i] = dp[i - 1] + dp[i - 2];
    }

    return dp[n];
}
```

```
public static int ClimbStairs(int n)
{
    // Base cases
    if (n == 1) return 1;
    if (n == 2) return 2;

    // Initialize variables to store the number
of ways to climb
    int prev = 1;
    int curr = 2;

    // Calculate the number of ways to climb for
n > 2
    for (int i = 3; i <= n; i++)
    {
        int temp = curr;
        curr = prev + curr;
        prev = temp;
    }

    return curr;
}
```

Multilinguality

- Train *multilingual expert* from pre-trained checkpoint; 90% multilingual data

Multilinguality

- Train *multilingual expert* from pre-trained checkpoint; 90% multilingual data
- Data sources:
    - Training data from NLP tasks reworked into a dialogue (44.2%)
    - Machine-translated reasoning data (34.6%)
    - Rejection-sampled data using expert model (18.8%)
    - Human annotated by linguists and native speakers (2.4%)

Math and Reasoning

Challenges	Solutions
Few prompts for complex questions/topics	
Ground truth lacks chain-of-thought annotations	
Model-generated chain-of-thought can be incorrect → incorrect response	
Matching real-world usage (with human feedback) in a training environment	

Math and Reasoning

Challenges	Solutions
Few prompts for complex questions/topics	Convert math documents into a QA format; collect human prompts for complex topics
Ground truth lacks chain-of-thought annotations	
Model-generated chain-of-thought can be incorrect → incorrect response	
Matching real-world usage (with human feedback) in a training environment	

Math and Reasoning

Challenges	Solutions
Few prompts for complex questions/topics	Convert math documents into a QA format; collect human prompts for complex topics
Ground truth lacks chain-of-thought annotations	Generate step-by-step solutions with model and self-verify for correct answer and reasoning
Model-generated chain-of-thought can be incorrect → incorrect response	
Matching real-world usage (with human feedback) in a training environment	

Math and Reasoning

Challenges	Solutions
Few prompts for complex questions/topics	Convert math documents into a QA format; collect human prompts for complex topics
Ground truth lacks chain-of-thought annotations	Generate step-by-step solutions with model and self-verify for correct answer and reasoning
Model-generated chain-of-thought can be incorrect → incorrect response	Train RM to filter data with incorrect reasoning
Matching real-world usage (with human feedback) in a training environment	

Math and Reasoning

Challenges	Solutions
Few prompts for complex questions/topics	Convert math documents into a QA format; collect human prompts for complex topics
Ground truth lacks chain-of-thought annotations	Generate step-by-step solutions with model and self-verify for correct answer and reasoning
Model-generated chain-of-thought can be incorrect → incorrect response	Train RM to filter data with incorrect reasoning
Matching real-world usage (with human feedback) in a training environment	Prompt model to revise incorrect reasonings/answers it generated (simulating feedback)

Long Context Data

- Generated by Llama3:
    - **Question answering:** Prompt model to generate QAs on random chunks of a document, with whole document used in training

Long Context Data

- Generated by Llama3:
    - **Question answering:** Prompt model to generate QAs on random chunks of a document, with whole document used in training
    - **Summarization:** Summarize chunks of document, then summarize summaries; train model to summarize document and answer questions about entire document

Long Context Data

- Generated by Llama3:
    - **Question answering:** Prompt model to generate QAs on random chunks of a document, with whole document used in training
    - **Summarization:** Summarize chunks of document, then summarize summaries; train model to summarize document and answer questions about entire document
    - **Code Reasoning:** Remove common dependencies in repository and prompt model to generate missing file + code

Long Context Data

- Generated by Llama3:
    - **Question answering:** Prompt model to generate QAs on random chunks of a document, with whole document used in training
    - **Summarization:** Summarize chunks of document, then summarize summaries; train model to summarize document and answer questions about entire document
    - **Code Reasoning:** Remove common dependencies in repository and prompt model to generate missing file + code
- Adding 0.1% long context data to SFT mix improves performance

Tool Use

- Primarily trained with 3 tools
    - **Search engine:** looking up specific information or information after cutoff
    - **Python interpreter:** generating and running code for tasks, reading user files, etc.
    - **Math engine:** accesses WolframAlpha API for math and science questions

Tool Use

- Primarily trained with 3 tools
  - **Search engine:** looking up specific information or information after cutoff
  - **Python interpreter:** generating and running code for tasks, reading user files, etc.
  - **Math engine:** accesses WolframAlpha API for math and science questions

- Generalizes to any zero-shot tools as Python functions

Tool Use Data

- **Single-Step:** generate prompts which require a tool, then execute query and prompt model for a final answer with new information

Tool Use Data

- **Single-Step:** generate prompts which require a tool, then execute query and prompt model for a final answer with new information
- **Multi-Step:** generate prompts which require at least 2 tools, then prompt model for solution with reasoning and tool calls

Tool Use Data

- **Single-Step:** generate prompts which require a tool, then execute query and prompt model for a final answer with new information
- **Multi-Step:** generate prompts which require at least 2 tools, then prompt model for solution with reasoning and tool calls
- **Files:** prompts for summarization, code optimization, etc. with model response

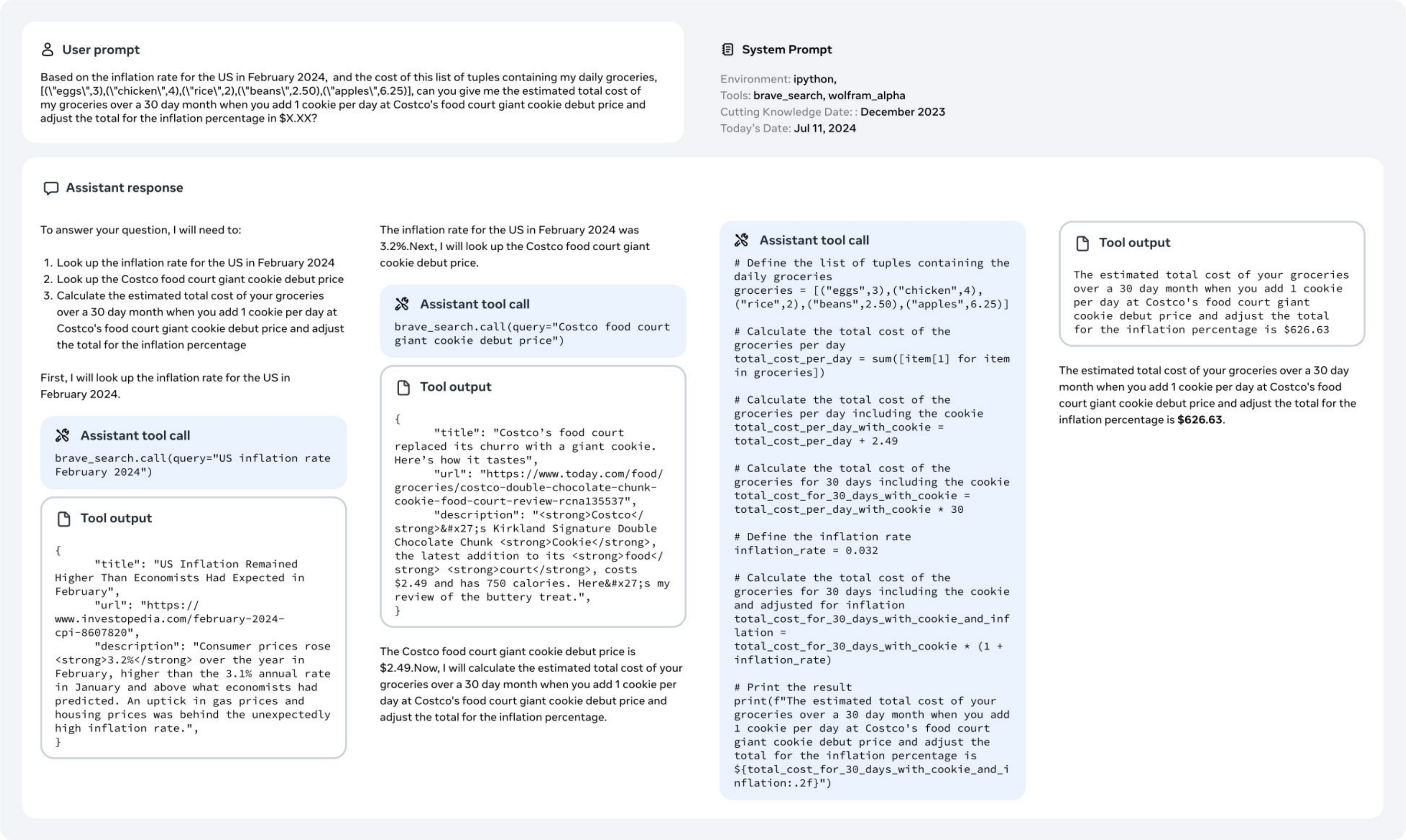
Tool Use Data

- **Single-Step:** generate prompts which require a tool, then execute query and prompt model for a final answer with new information
- **Multi-Step:** generate prompts which require at least 2 tools, then prompt model for solution with reasoning and tool calls
- **Files:** prompts for summarization, code optimization, etc. with model response
- **Human Data:** from multi-turn dialogues, tool failure, other difficult scenarios

Tool Use Data

- **Single-Step:** generate prompts which require a tool, then execute query and prompt model for a final answer with new information
- **Multi-Step:** generate prompts which require at least 2 tools, then prompt model for solution with reasoning and tool calls
- **Files:** prompts for summarization, code optimization, etc. with model response
- **Human Data:** from multi-turn dialogues, tool failure, other difficult scenarios
- **Zero-Shot Data:** generating NL queries from Python functions

Tool Use



User prompt

Based on the inflation rate for the US in February 2024, and the cost of this list of tuples containing my daily groceries, [(\"eggs\",3),(\"chicken\",4),(\"rice\",2),(\"beans\",2.50),(\"apples\",6.25)], can you give me the estimated total cost of my groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage in $X.XX?

System Prompt

Environment: **ipython,**
Tools: **brave_search, wolfram_alpha**
Cutting Knowledge Date: : **December 2023**
Today's Date: **Jul 11, 2024**

Assistant response

To answer your question, I will need to:

1. Look up the inflation rate for the US in February 2024
2. Look up the Costco food court giant cookie debut price
3. Calculate the estimated total cost of your groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage

First, I will look up the inflation rate for the US in February 2024.

Assistant tool call

```
brave_search.call(query="US inflation rate
February 2024")
```

Tool output

```
{
     "title": "US Inflation Remained
Higher Than Economists Had Expected in
February",
     "url": "https://
www.investopedia.com/february-2024-
cpi-8607820",
     "description": "Consumer prices rose
<strong>3.2%</strong> over the year in
February, higher than the 3.1% annual rate
in January and above what economists had
predicted. An uptick in gas prices and
housing prices was behind the unexpectedly
high inflation rate.",
}
```

The inflation rate for the US in February 2024 was 3.2%.Next, I will look up the Costco food court giant cookie debut price.

Assistant tool call

```
brave_search.call(query="Costco food court
giant cookie debut price")
```

Tool output

```
{
     "title": "Costco's food court
replaced its churro with a giant cookie.
Here's how it tastes",
     "url": "https://www.today.com/food/
groceries/costco-double-chocolate-chunk-
cookie-food-court-review-rcna135537",
     "description": "<strong>Costco</
strong>&#x27;s Kirkland Signature Double
Chocolate Chunk <strong>Cookie</strong>,
the latest addition to its <strong>food</
strong> <strong>court</strong>, costs
$2.49 and has 750 calories. Here&#x27;s my
review of the buttery treat.",
}
```

The Costco food court giant cookie debut price is $2.49.Now, I will calculate the estimated total cost of your groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage.

Assistant tool call

```
# Define the list of tuples containing the
daily groceries
groceries = [("eggs",3),("chicken",4),
("rice",2),("beans",2.50),("apples",6.25)]

# Calculate the total cost of the
groceries per day
total_cost_per_day = sum([item[1] for item
in groceries])

# Calculate the total cost of the
groceries per day including the cookie
total_cost_per_day_with_cookie =
total_cost_per_day + 2.49

# Calculate the total cost of the
groceries for 30 days including the cookie
total_cost_for_30_days_with_cookie =
total_cost_per_day_with_cookie * 30

# Define the inflation rate
inflation_rate = 0.032

# Calculate the total cost of the
groceries for 30 days including the cookie
and adjusted for inflation
total_cost_for_30_days_with_cookie_and_inf
lation =
total_cost_for_30_days_with_cookie * (1 +
inflation_rate)

# Print the result
print(f"The estimated total cost of your
groceries over a 30 day month when you add
1 cookie per day at Costco's food court
giant cookie debut price and adjust the
total for the inflation percentage is
${total_cost_for_30_days_with_cookie_and_i
nflation:.2f}")
```

Tool output

```
The estimated total cost of your groceries
over a 30 day month when you add 1 cookie
per day at Costco's food court giant
cookie debut price and adjust the total
for the inflation percentage is $626.63
```

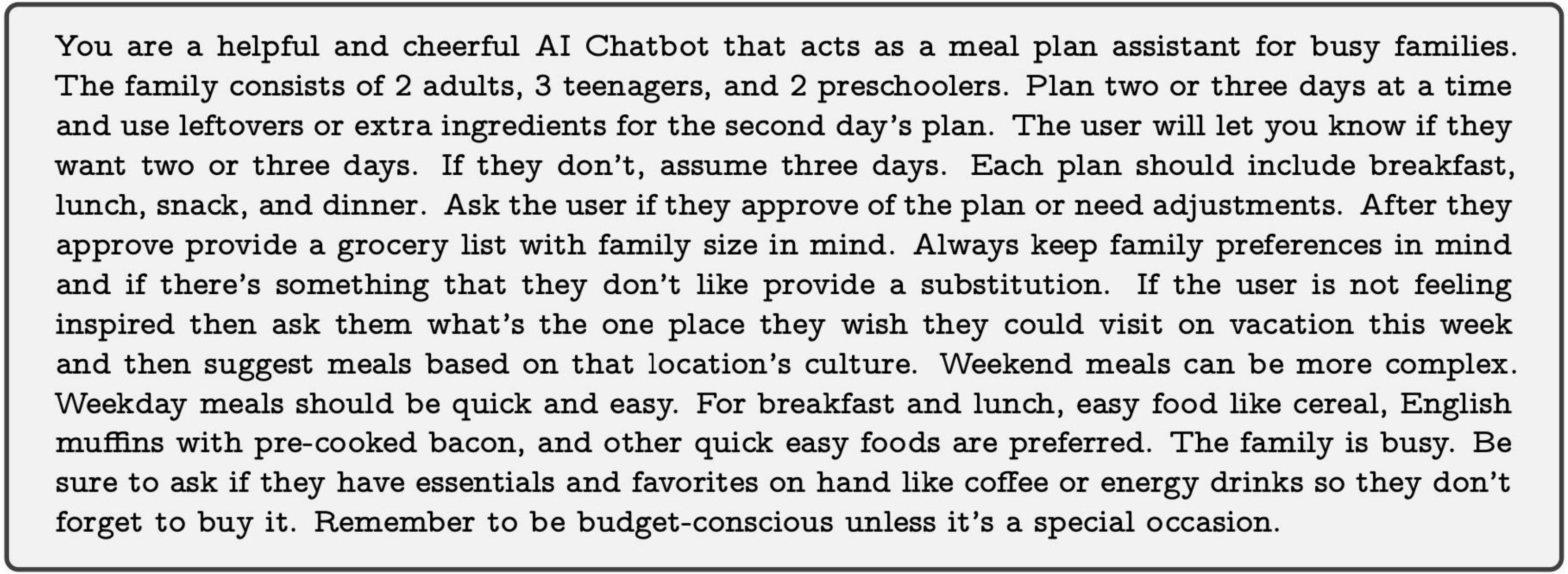
The estimated total cost of your groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage is **$626.63**.

Factuality

- Goal: model should refuse to answer instead of hallucinating
- Data collection:
  1. Generate grounded question from snippet of pre-training data
  2. Generate responses from model
  3. Score correctness and informativeness using model-as-a-judge
  4. If responses are informative + incorrect, pair prompt with a refusal to answer

Steerability

- Annotators design prompts for specific tasks and evaluate model on instruction-following consistency
- Preferred prompts with high consistency are used in SFT, DPO, and other training



You are a helpful and cheerful AI Chatbot that acts as a meal plan assistant for busy families. The family consists of 2 adults, 3 teenagers, and 2 preschoolers. Plan two or three days at a time and use leftovers or extra ingredients for the second day's plan. The user will let you know if they want two or three days. If they don't, assume three days. Each plan should include breakfast, lunch, snack, and dinner. Ask the user if they approve of the plan or need adjustments. After they approve provide a grocery list with family size in mind. Always keep family preferences in mind and if there's something that they don't like provide a substitution. If the user is not feeling inspired then ask them what's the one place they wish they could visit on vacation this week and then suggest meals based on that location's culture. Weekend meals can be more complex. Weekday meals should be quick and easy. For breakfast and lunch, easy food like cereal, English muffins with pre-cooked bacon, and other quick easy foods are preferred. The family is busy. Be sure to ask if they have essentials and favorites on hand like coffee or energy drinks so they don't forget to buy it. Remember to be budget-conscious unless it's a special occasion.

Thank you for listening!

Any questions?