Priority Sampling of Large Language Models for Compilers

Dejan Grubišić

Volker Seeker

Gabriel Synnaeve

Hugh Leather

John Mellor-Crummey

Chris Cummins

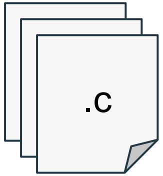
Athens, April 2024.

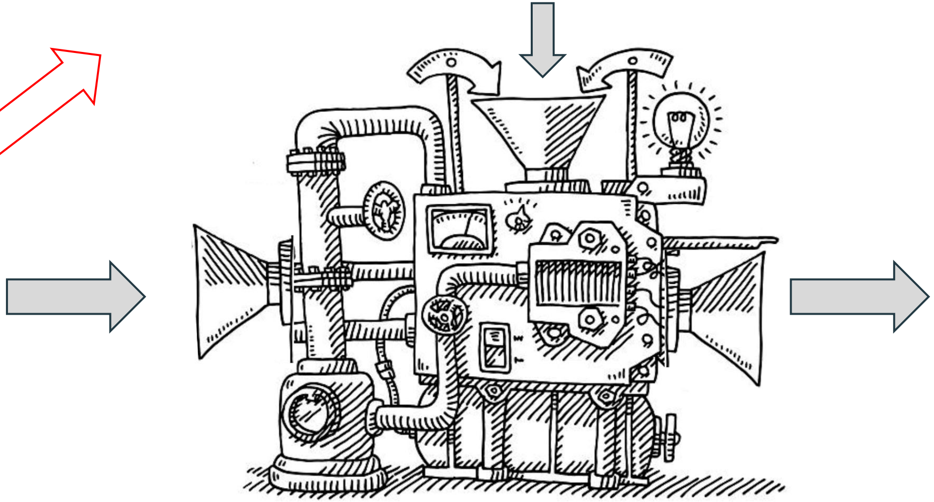
How does compiler work?

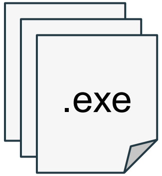
List of Transformation Passes in Oz

-ee-instrument -simplifycfg -sroa -early-cse -lower-expect -forceattrs -inferattrs -ipsccp -called-value-propagation -attributor -globalopt -mem2reg -deadargelim -instcombine -simplifycfg -prune-eh -inline -functionattrs -sroa -early-cse-memssa -speculative-execution -jump-threading -correlated-propagation -simplifycfg -instcombine -tailcallelim -simplifycfg -reassociate -loop-simplify -lcssa -loop-rotate -licm -loop-unswitch -simplifycfg -instcombine -loop-simplify -lcssa -indvars -loop-idiom -loop-deletion -loop-unroll -mldst-motion -gvn -memcpyopt -sccp -bdce -instcombine -jump-threading -correlated-propagation -dse -loop-simplify -lcssa -licm -adce -simplifycfg -instcombine -barrier -elim-avail-extern -rpo-functionattrs -globalopt -globaldce -float2int -lower-constant-intrinsics -loop-simplify -lcssa -loop-rotate -loop-distribute -loop-vectorize -loop-simplify -loop-load-elim -instcombine -simplifycfg -instcombine -loop-simplify -lcssa -loop-unroll -instcombine -loop-simplify -lcssa -licm -alignmentfromassumptions -strip-dead-prototypes -globaldce -constmerge -loop-simplify -lcssa -loop-sink -instsimplify -div-rem-pairs -simplifycfg

Can you predict optimal pass list for given input code?

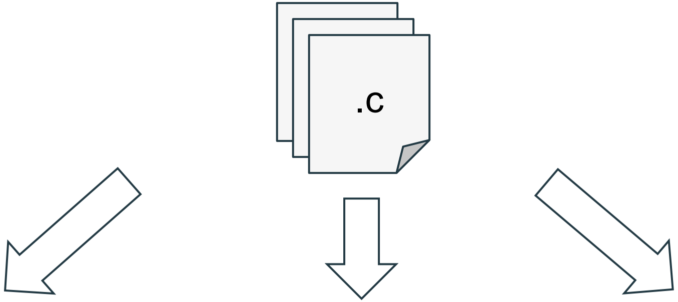






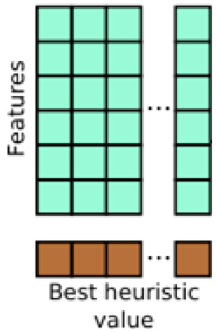
Compiler

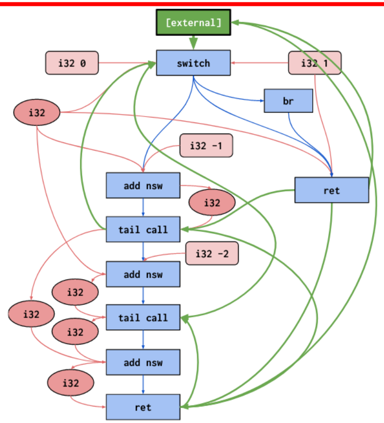
What Is A Good Representation?

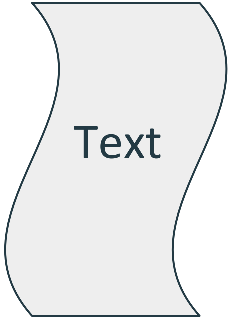


Traditional representations lose information

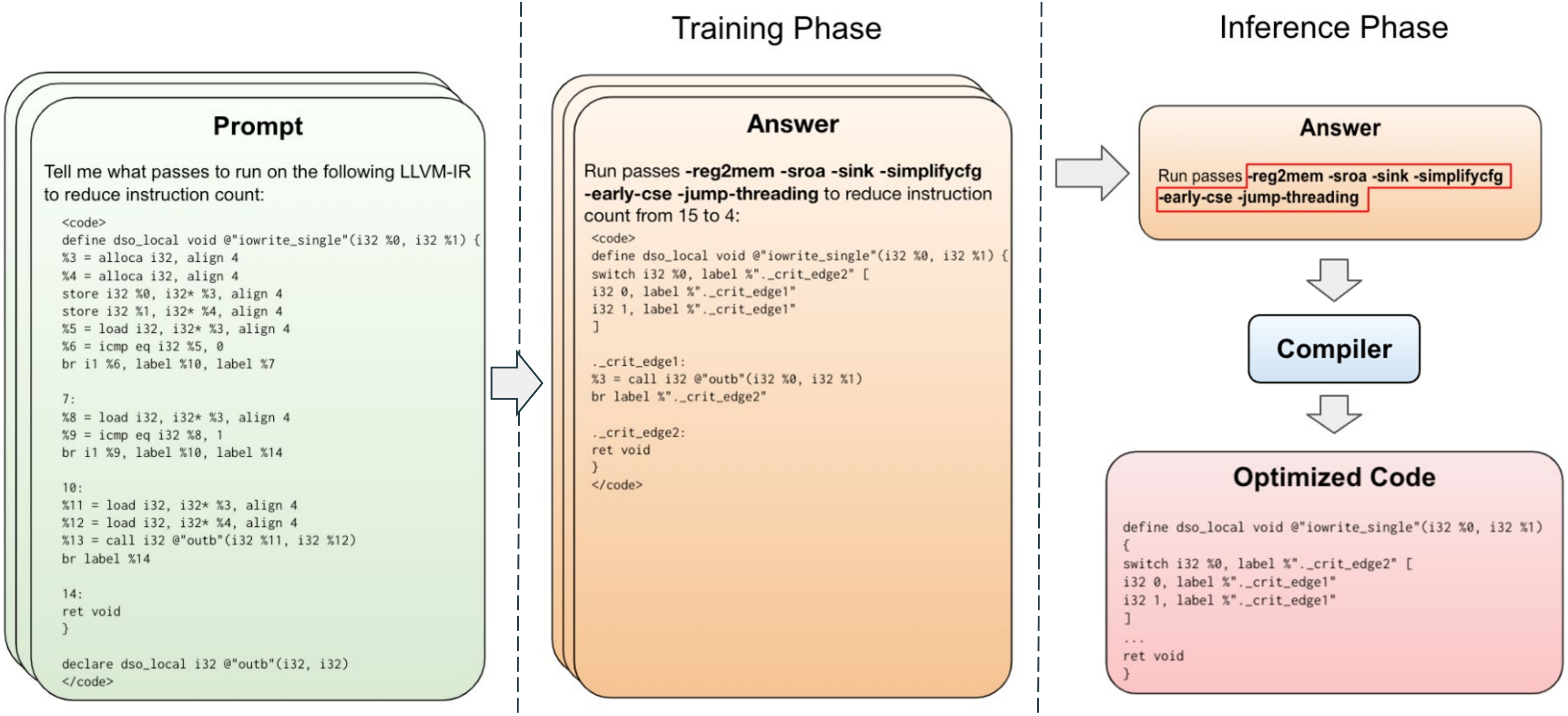
LLMs can do it!







Large Language Model Task

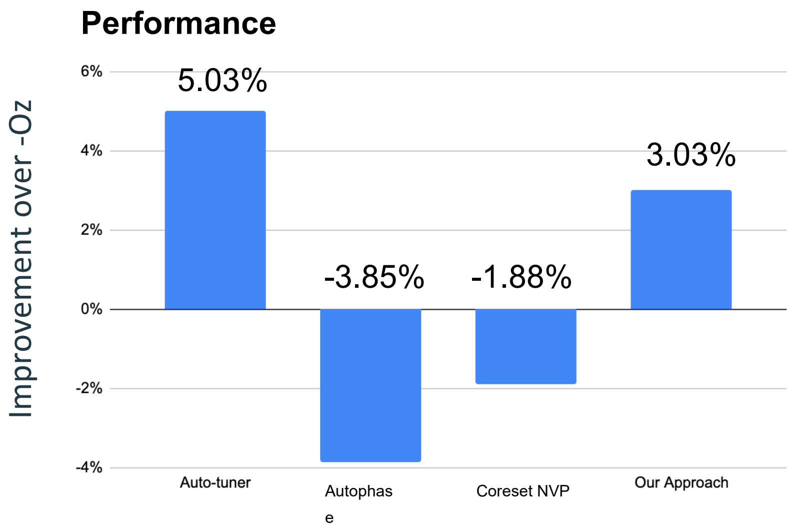


[Large Language Models for Compiler Optimization, Cummins et al.]

Comparing LLMs with Alternatives

Test Set of 100,000 functions

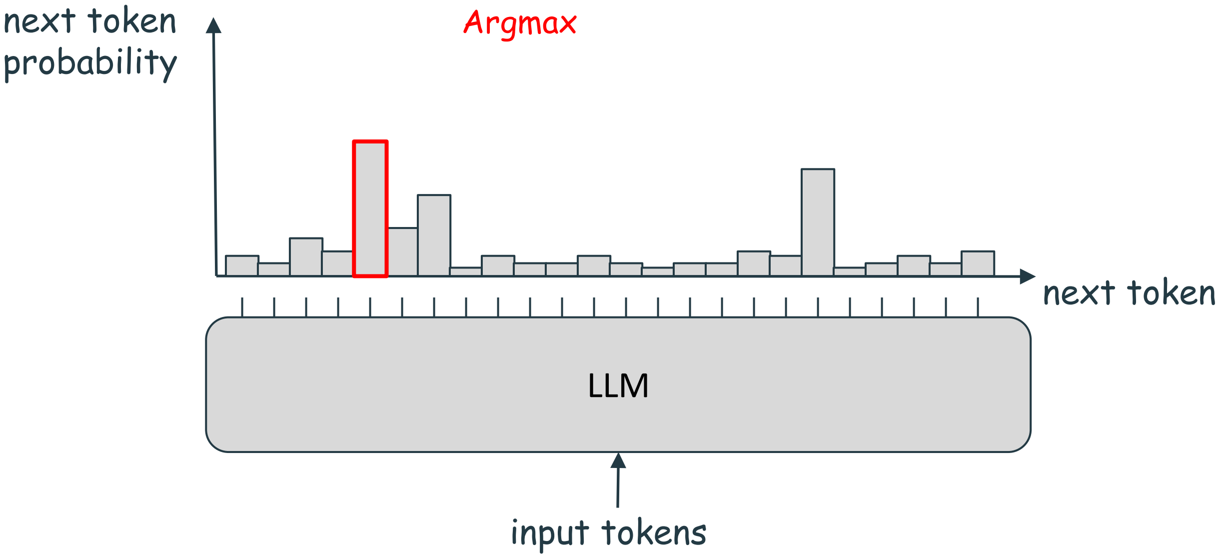
Compilations	
Auto-tuner	2,522,253,069
Autophase (MLSys '20)	4,500,000
Coreset NVP (ICML '23)	442,747
Our Approach	**0**



Large Language Models for Compiler Optimization, Cummins et al.

Can we do better?

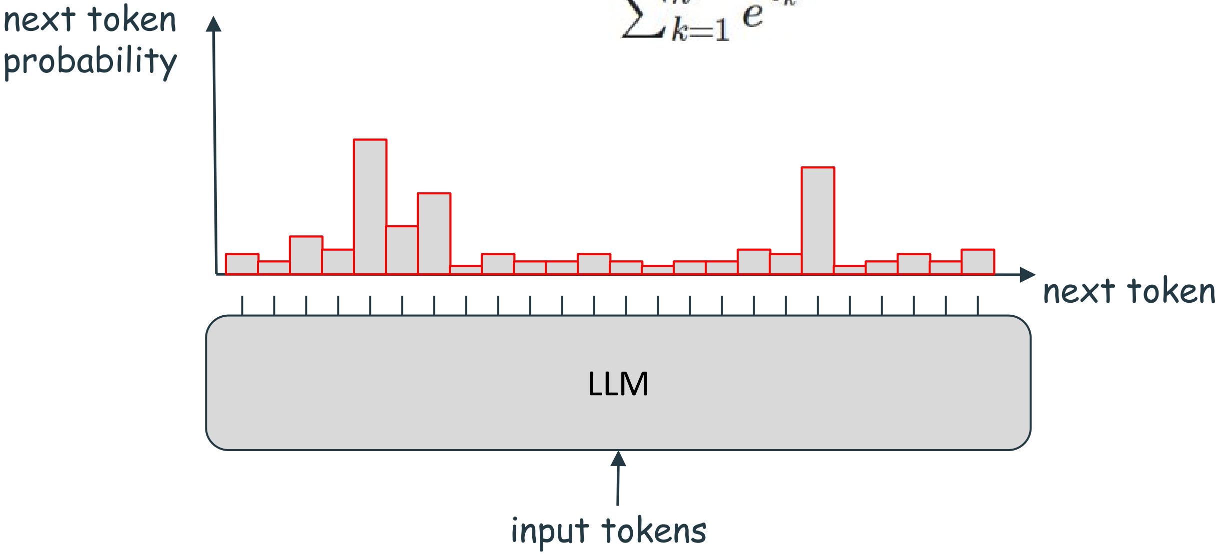
Next token generation



Temperature Sampling

Softmax function

$$P_i = \frac{e^{y_i}}{\sum_{k=1}^{n} e^{y_k}}$$



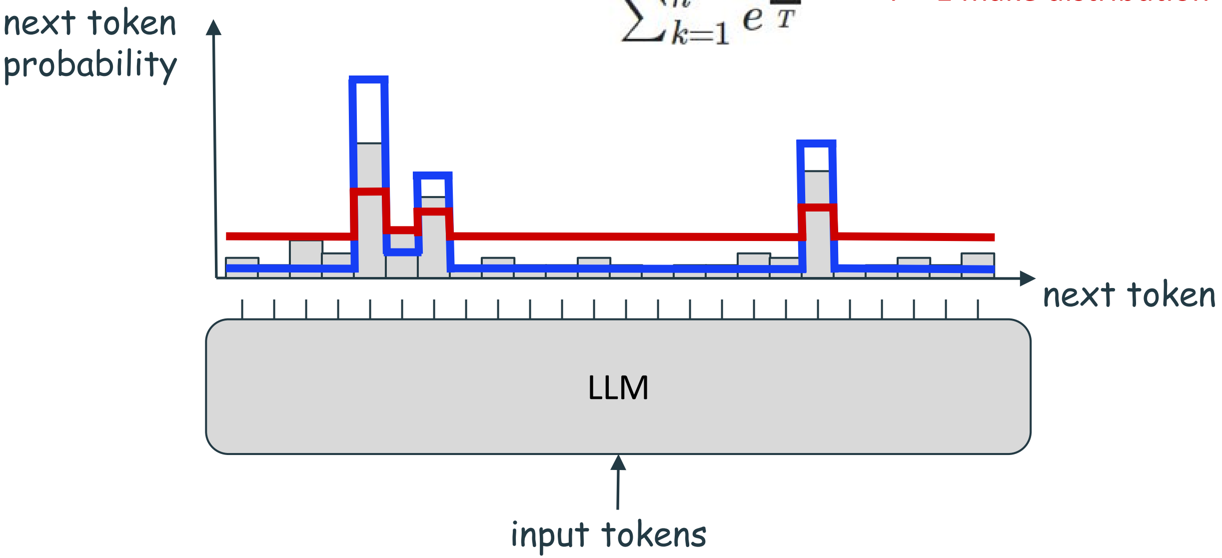
Temperature Sampling

Softmax function with temperature

$$P_i = \frac{e^{\frac{y_i}{T}}}{\sum_{k=1}^{n} e^{\frac{y_k}{T}}}$$

T < 1 make distribution sharper

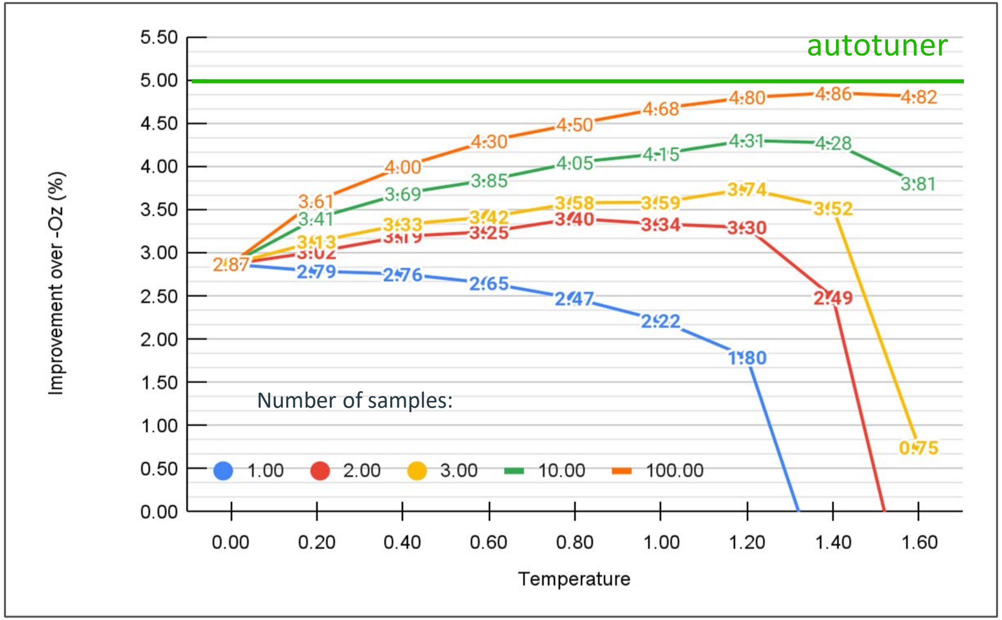
T > 1 make distribution flatter



Results

Temperature Sampling of Original Model

Original model on 50k test examples



Problems with Temperature Sampling

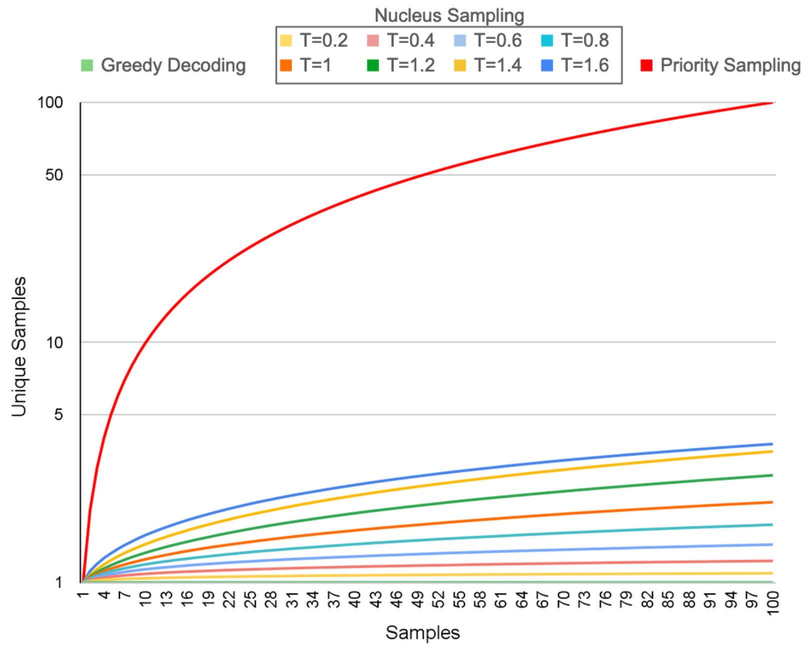
63 legal sequences, 37 illegal sequences

duplicates

```
23 -Oz
2 -simplifycfg -gvn -instcombine -sroa
2 -Os -newgvn
2 -early-cse-memssa -reg2mem -memcpyopt -gvn -instcombine -simplifycfg
1 -mem2reg -simplifycfg -instcombine -newgvn
1 -simplifycfg -gvn-hoist -sroa
1 -gvn -instcombine -simplifycfg
1 -sroa -simplifycfg -gvn -instcombine
1 -loop-rotate -newgvn -reg2mem -ipsccp -memcpyopt -gvn -O3
1 -mem2reg -simplifycfg -gvn-hoist -instcombine
1 -simplifycfg -instcombine -sroa -newgvn
1 -reg2mem -jump-threading -licm -newgvn -mem2reg -jump-threading -loop-deletion
1 -simplifycfg -gvn-hoist -O2
1 -simplifycfg -gvn-hoist -instcombine -sroa
1 -instcombine -newgvn -simplifycfg -sroa -gvn
1 -loop-rotate -Os
1 -instcombine -mem2reg -newgvn -simplifycfg
1 -instcombine -O2
1 -instsimplify -simplifycfg -sroa -instcombine
1 -simplifycfg -newgvn -gvn-hoist -instcombine
1 -simplifycfg -O1
1 -instcombine -newgvn -simplifycfg -sroa -early-cse
1 -gvn -instcombine -simplifycfg -newgvn
1 -speculative-execution -O1 -Os
...
```

Frequency of 100 samples generated with the original model on T=1.4

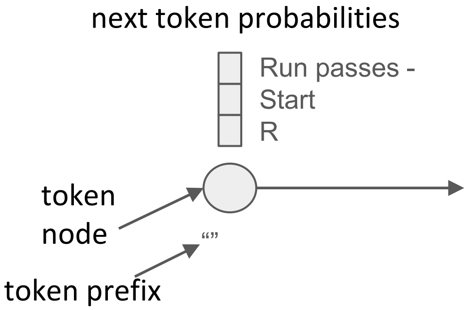
Problems with Temperature Sampling



Number of unique samples for Greedy Decoding, Nucleus Sampling and Priority Sampling

Priority Sampling Algorithm

Priority Sampling Algorithm

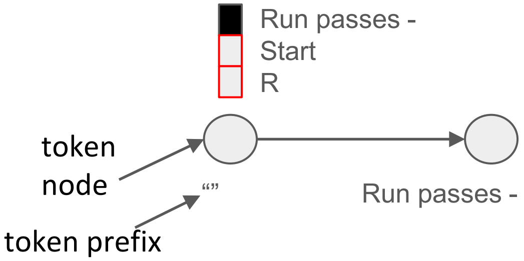


Priority queue for next branch

P(x)	Next sample starts with text

Priority Sampling Algorithm

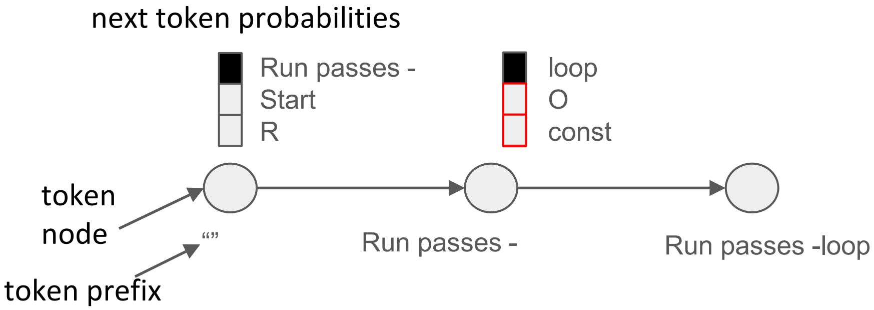
next token probabilities



Priority queue for next branch

P(x)	Next sample starts with text
0.02	Start
0.01	R

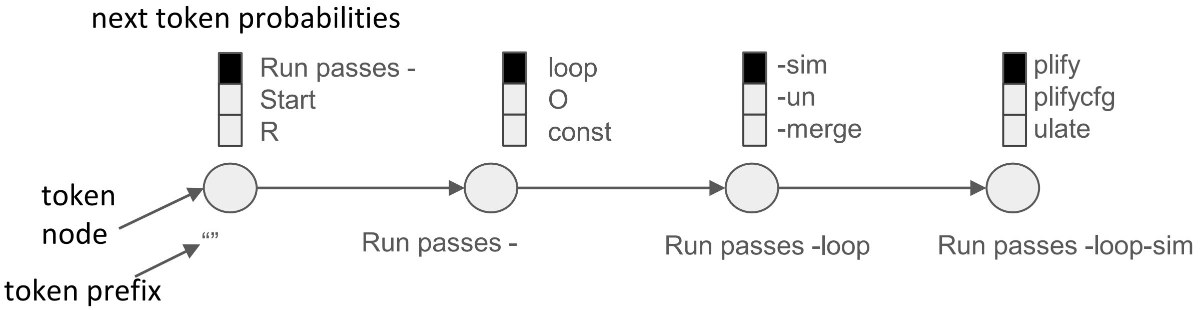
Priority Sampling Algorithm



Priority queue for next branch

P(x)	Next sample starts with text
0.61	Run passes -O
0.08	Run passes -const
0.02	Start
0.01	R

Priority Sampling Algorithm



Priority queue for next branch

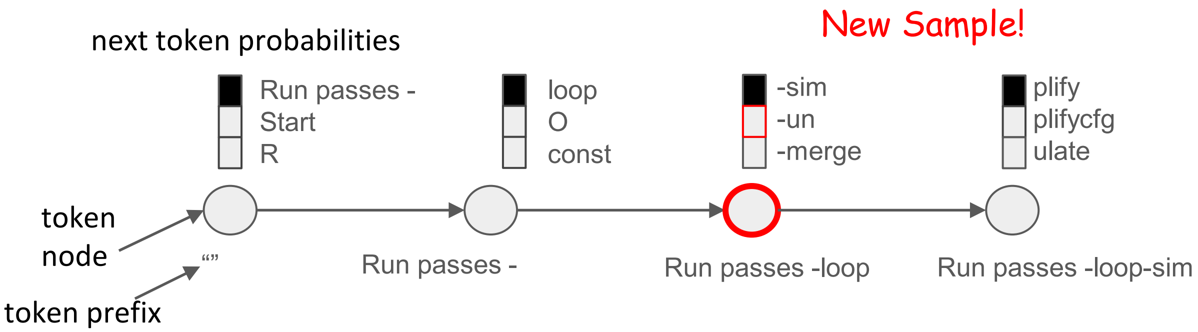
P(x)	Next sample starts with text
0.70	Run passes -loop-un
0.61	Run passes -O
0.4	Run passes -loop-simplifycfg
0.12	Run passes -loop-merge
0.08	Run passes -const
0.02	Start
0.01	Run passes -loop-simulate

STOP!

new sample

Run passes -loop-simplify

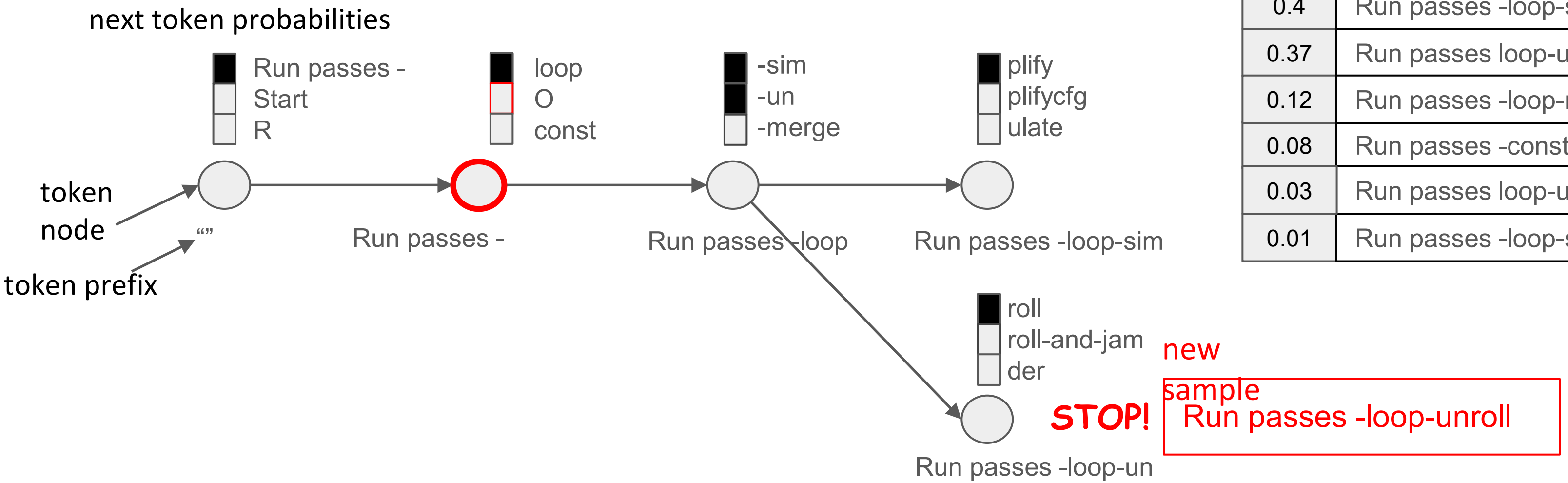
Priority Sampling Algorithm



Priority queue for next branch

P(x)	Next sample starts with text
0.70	Run passes -loop-un
0.61	Run passes -O
0.4	Run passes -loop-simplifycfg
0.12	Run passes -loop-merge
0.08	Run passes -const
0.02	Start
0.01	Run passes -loop-simulate

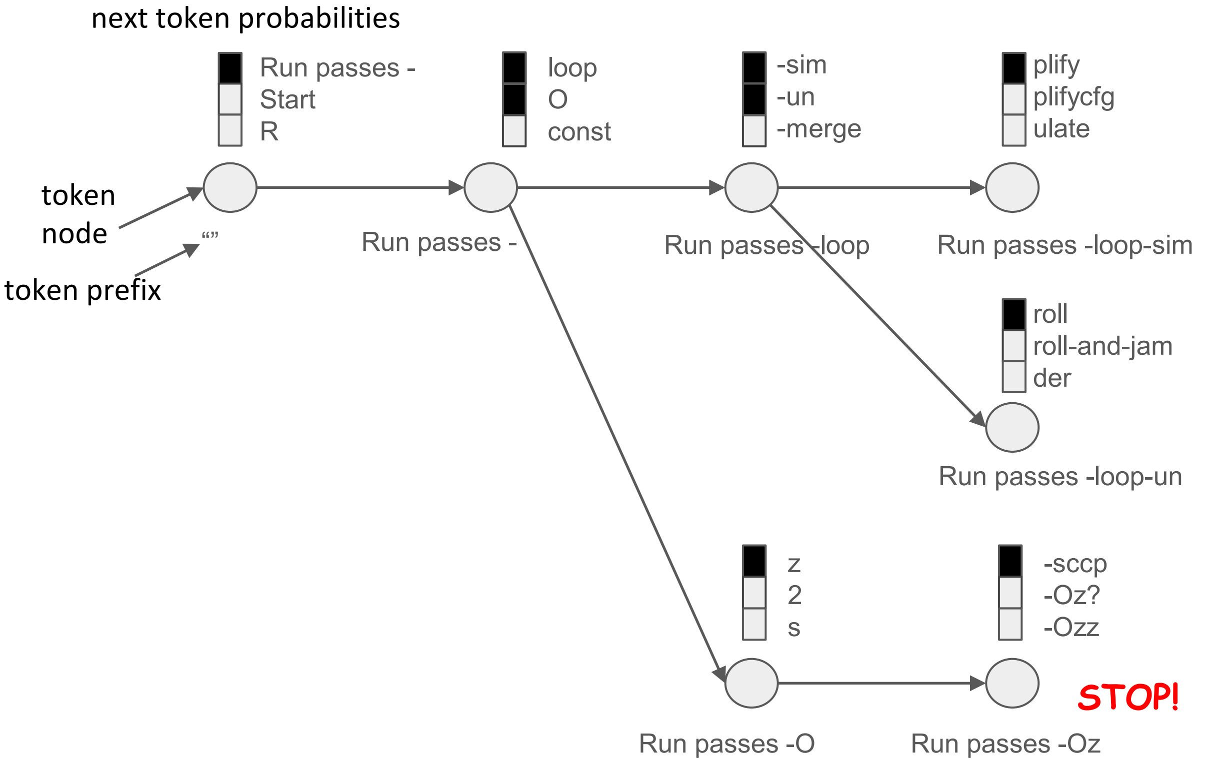
Priority Sampling Algorithm



Priority queue for next branch

P(x)	Next sample starts with text
0.61	Run passes -O
0.4	Run passes -loop-simplifycfg
0.37	Run passes loop-unroll-and-jam
0.12	Run passes -loop-merge
0.08	Run passes -const
0.03	Run passes loop-under
0.01	Run passes -loop-simulate

Priority Sampling Algorithm



new sample

Run passes -Oz -sccp

Priority queue for next branch

P(x)	Next sample starts with text
0.86	Run passes -O2
0.4	Run passes -loop-simplifycfg
0.37	Run passes loop-unroll-and-jam
0.12	Run passes -loop-merge
0.08	Run passes -const
0.03	Run passes loop-under
0.01	Run passes -loop-simulate

Results

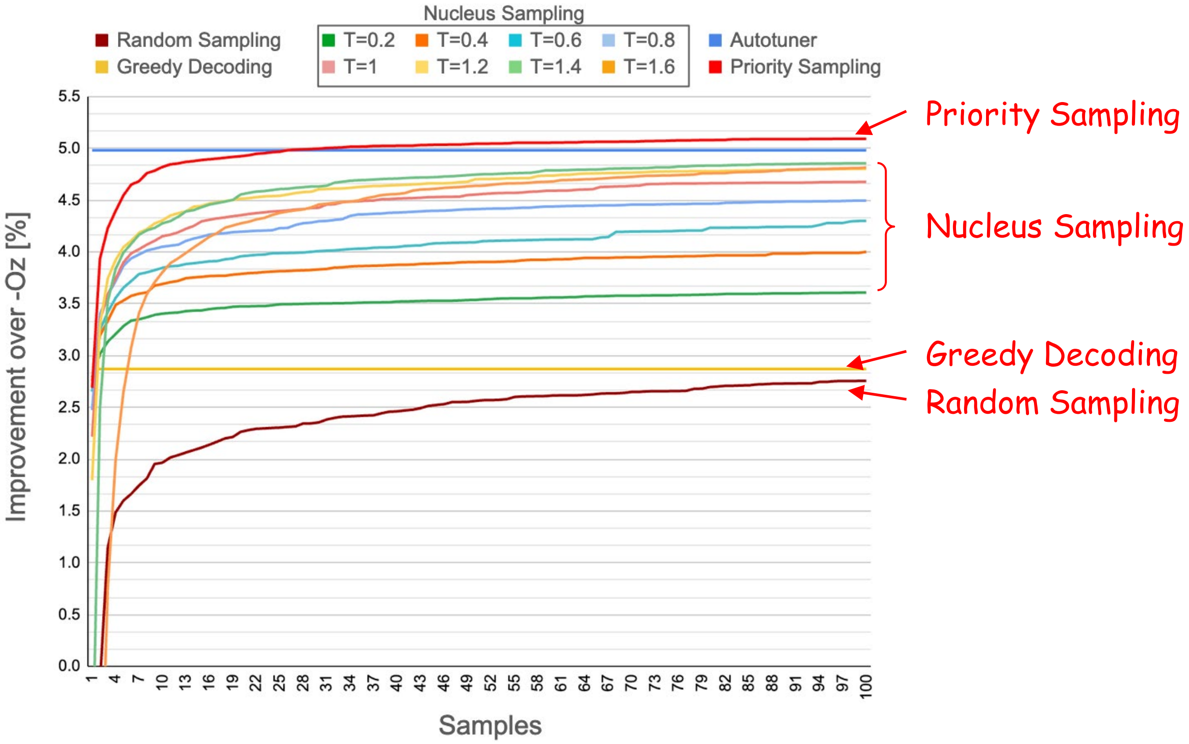
Priority Sampling Dominates

Priority Sampling:

5 samples -> 91%

30 samples -> 100%

100 samples -> 102%



Abalations

	Improvement over -Oz [%]					
Method	Sample 1	Sample 3	Sample 5	Sample 10	Sample 30	Sample 100
Autotuner	4.98%					
Priority Sampling (PS)	2.69%	**4.23%**	4.55%	4.82%	**5.00%**	5.09%
PS (no regex)	**3.17%**	4.18%	4.41%	4.64%	4.93%	**5.12%**
PS (max_branch 3)	2.62%	4.22%	4.56%	4.83%	4.99%	5.09%
PS (max_branch 5)	2.62%	4.22%	**4.61%**	**4.85%**	4.99%	5.09%
PS geometric (PSG)	2.68%	4.17%	4.45%	4.75%	4.96%	5.07%
PSG (max_branch 3)	2.62%	4.17%	4.52%	4.77%	4.98%	5.11%
PSG (max_branch 5)	2.62%	4.17%	4.56%	4.80%	4.98%	**5.12%**

Ablation on regular expression, branching factor, and geometric mean metrics

Takeaways

Takeaways

- LLMs could predict LLVM flags
  - 3% improvement over -Oz with 0 compilations
- Temperature Sampling
  - 98% of autotuner performance given 100 samples
  - duplicates and illegal sequences
- Priority Sampling
  - no duplicates or illegal sequences
  - 5 samples -> 91% of autotuner performance
  - >30 samples -> outperforms autotuner

Backup slides

Model

- 7 Billion Parameter LLaMa2 Model
- Training set: 1M examples
  - IR text -> passes + instruction counts + optimized IR
- 30k training steps from scratch (15.7B tokens)
- 620 GPU days training time


Credit: craiyon.com

Large Language Models for Compiler Optimization, Cummins et al.

LLMs for Compilers Data

	n functions	unoptimized instruction count	size on disk	n tokens
Handwritten	610,610	8,417,799	653.5 MB	214,746,711
Synthetic	389,390	13,775,149	352.3 MB	158,435,151
Total	1,000,000	16,411,249	1.0 GB	373,181,862

a) Training data

	n functions	unoptimized instruction count	-Oz instruction count
AI-SOCO [31]	8,929	97,800	47,578
ExeBench [32]	26,806	386,878	181,277
POJ-104 [33]	310	8,912	4,492
Transcoder [12]	17,392	289,689	129,611
CSmith [34]	33,794	647,815	138,276
YARPGen [35]	12,769	285,360	144,539
Total	100,000	1,716,354	645,773

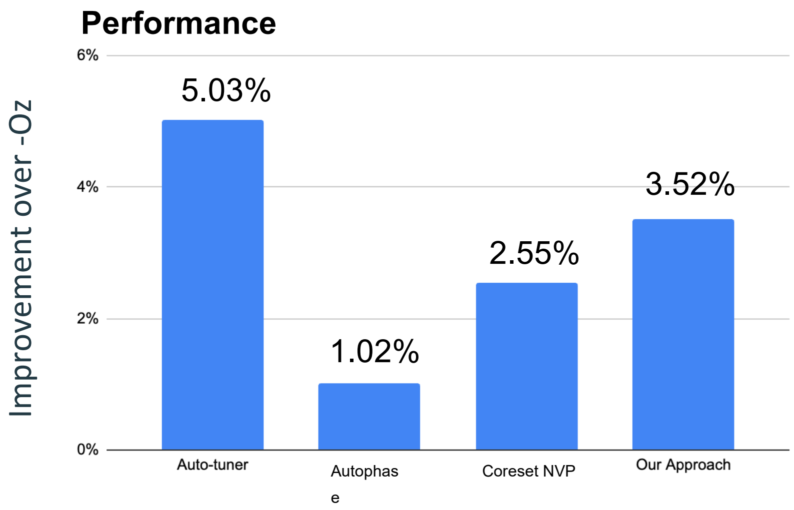
b) Test data

Data for training and validation of Large Language Models

Comparing LLMs with Alternatives

Test Set of 100,000 functions

Compilations (-Oz backup)	
Auto-tuner	2,522,253,069
Autophase (MLSys '20)	4,600,000
Coreset NVP (ICML '23)	542,747
Our Approach	**5721**



Large Language Models for Compiler Optimization, Cummins et al.