Founded studies:
HumanEval
Class-Eval
Code Generation on MBPP
ReCode: Robustness Evaluation
BioCoder
Automated Verilog RTL
Pseudocode-NI comparator benchmark

Comparison of CodeGen models with each other on HumanEval benchmark

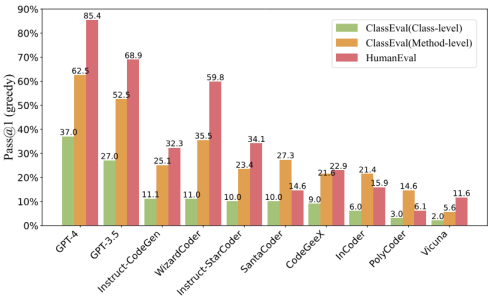
	Model	Metric Name	Metric Value	Global Rank
2	CodeGen-Mono 350M zero-shot	Pass@1	12,8	116
3	CodeGen-Mono 6.1B (zero-shot)	Pass@1	26,1	89
4	CodeGen-Mono 2.7B (zero-shot)	Pass@1	23,7	91
5	CodeGen-NL 16B (zero-shot)	Pass@1	14,2	113
6	CodeGen-Multi 16B (zero-shot)	Pass@1	18,3	101
7	CodeGen-Multi 6.1B (zero-shot)	Pass@1	18,2	103
8	CodeGen-Multi 2.7B (zero-shot)	Pass@1	14,5	112
9	CodeGen-Multi 350M (zero-shot)	Pass@1	6,7	124
10	CodeGen-NL 6.1B (zero-shot)	Pass@1	10,4	121
11	CodeGen-NL 2.7B (zero-shot)	Pass@1	6,7	124
12	CodeGen-Mono 16B (zero-shot)	Pass@1	29,03	81
13	CodeGen-Mono 16B (zero-shot)	Pass@10	49,9	1
14	CodeGen-Mono 16B (zero-shot)	Pass@100	75	1

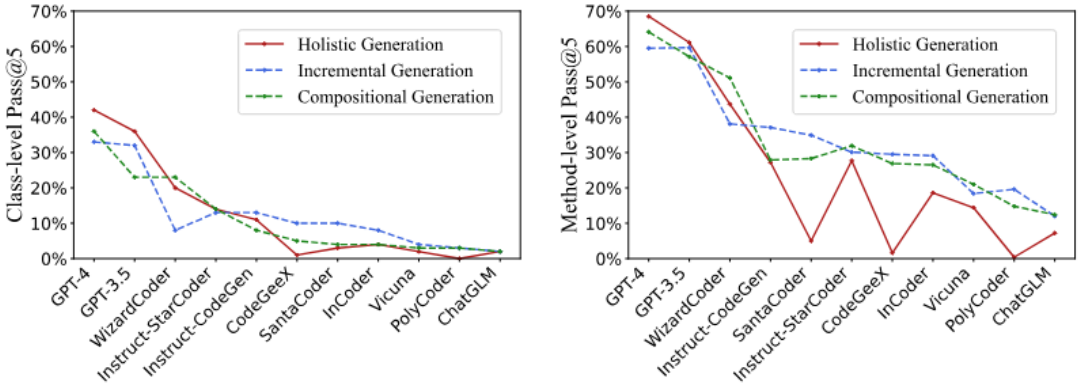
HumanEval consists of 164 Python programs with 8 tests for each. Considering CodeGen-Mono models trained only in Python and don't have noisy data, it is expected that these models will outperform other CodeGen models. For the time these testing take place CodeGen-Mono 16B models were the best performing among all LLM's on the web since they are fine tuned specifically for these tasks they were even better than the known models as GPT's. However, currently in the HumanEval benchmark; the best performing CodeGen Model is only 62th. Except the newly developed and fine tuned CodeGen-Scrum model.

Current ranking for HumanEval benchmark:
https://paperswithcode.com/sota/code-generation-on-humaneval

Class-Eval Benchmark
It evaluates the model capability of generating a class of multiple interdependent methods for the given natural language description. Instruction-CodeGen 16B multi tested on this benchmark along with many.

Model	Class-level			Method-level		
	Pass@1	Pass@3	Pass@5	Pass@1	Pass@3	Pass@5
GPT-4	**37.6%**	**41.3%**	**42.0%**	**62.8%**	**67.4%**	**68.5%**
GPT-3.5	29.6%	34.9%	36.0%	50.4%	59.0%	61.1%
WizardCoder	12.2%	20.0%	23.0%	35.2%	47.1%	51.1%
Instruct-StarCoder	10.2%	12.7%	14.0%	23.1%	26.5%	27.7%
SantaCoder	8.6%	9.9%	10.0%	27.7%	33.0%	34.9%
Instruct-CodeGen	8.2%	12.3%	13.0%	24.9%	34.3%	37.1%
CodeGeeX	7.2%	9.4%	10.0%	21.2%	27.1%	29.5%
InCoder	6.2%	7.6%	8.0%	21.1%	26.5%	29.1%
Vicuna	3.0%	3.6%	4.0%	11.0%	15.8%	18.4%
ChatGLM	1.4%	2.6%	3.0%	8.2%	11.2%	12.4%
PolyCoder	1.4%	2.2%	3.0%	13.2%	17.5%	19.6%





(a) Class-level *Pass@5* (b) Method-level *Pass@5*

On 3 pass@ levels and 2 different method rankings, CodeGen model shows average performance. Generally closer with other models except GPT and WizardCoder models.

Code Generation on MBPP

Below results are current ones for all the models tested on this benchmark.

GPT-3.5 Turbo + LLMCodeGen Scrum : 06th with 82.5 Accuracy
CodeGen-Mono 16B + CodeT : 47th with 49.5 Accuracy
CodeGen 16B + MBR-Exec : 56th with 47.3 Accuracy
CodeGen 16B + Coder-Reviewer : 60th with 46.2 Accuracy
CodeGen 16B + Reviewer : 65th with 44.1 Accuracy

Value	Metric	Model name	Dataset
55.4	Accuracy	code-cushman-001 12	MBPP
61.9	Accuracy	code-davinci-001 175B	MBPP
67.7	Accuracy	code-davinci-002 175B	MBPP
34.4	Accuracy	InCoder 6.7B + CodeT	MBPP
49.5	Accuracy	CodeGen-Mono 16B +	MBPP

With specific problems on the MBPP benchmark without Fine Tuning, CodeGen-Mono 16B model with the help of CodeT model resulted with average values compared to above models.
Comparison between them for “Mostly Basic Python Problems MBPP”

Model	pass@1	pass@10	pass@100
CODEGEN-NL 350M	0.96	6.37	19.91
CODEGEN-NL 2.7B	5.34	24.63	48.95
CODEGEN-NL 6.1B	8.15	31.21	55.27
CODEGEN-NL 16.1B	10.92	38.43	62.76
CODEGEN-MULTI 350M	7.46	24.18	46.37
CODEGEN-MULTI 2.7B	18.06	45.80	65.34
CODEGEN-MULTI 6.1B	18.35	47.27	67.92
CODEGEN-MULTI 16.1B	20.94	51.61	70.02
CODEGEN-MONO 350M	14.59	41.49	63.00
CODEGEN-MONO 2.7B	27.31	59.19	74.24
CODEGEN-MONO 6.1B	32.48	64.20	76.81
CODEGEN-MONO 16.1B	35.28	67.32	80.09
INCODER 6B	21.30	46.50	66.20
code-cushman-001	45.90	66.90	79.90
code-davinci-001	51.80	72.80	84.10
code-davinci-002	58.10	76.70	84.50

ReCode: Robustness Evaluation of Code Generation Models

This benchmark is a comprehensive robustness evaluation benchmark for code generation models.

MBPP	Metric	CodeGen 2B mono	CodeGen 2B multi	CodeGen 6B mono	CodeGen 6B multi	CodeGen 16B mono	CodeGen 16B multi	InCoder 1B	InCoder 6B	GPT-J 6B
Docstring	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	RP_5@1↑	0.137	0.050	0.147	0.042	**0.163**	0.045	0.011	0.031	0.013
	RD_5@1(%)↓	**56.96**	73.66	59.38	80.93	59.85	81.28	91.20	84.54	90.00
	RR_5@1(%)↓	36.86	34.39	41.89	36.76	46.72	44.66	**25.57**	35.32	30.08
Function	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	RP_5@1↑	0.221	0.101	0.252	0.110	**0.279**	0.139	0.047	0.087	0.043
	RD_5@1(%)↓	30.42	47.31	**30.40**	50.23	31.31	42.55	63.20	56.19	67.69
	RR_5@1(%)↓	19.51	20.43	24.13	22.79	24.95	23.51	**16.22**	20.02	17.56
Syntax	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	RP_5@1↑	0.027	0.008	0.027	0.008	**0.038**	0.017	0.008	0.006	0.004
	RD_5@1(%)↓	**94.06**	97.12	95.01	97.52	93.34	95.39	96.24	97.89	97.66
	RR_5@1(%)↓	59.03	45.07	64.17	47.74	67.04	54.21	35.42	45.79	**30.60**
Format	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	RP_5@1↑	0.333	0.146	0.289	0.166	**0.403**	0.214	0.091	0.130	0.080
	RD_5@1(%)↓	**26.03**	48.92	46.07	49.69	29.32	43.63	58.22	55.28	54.39
	RR_5@1(%)↓	19.82	25.15	31.11	27.00	25.26	26.59	19.61	28.54	**18.28**

Table 4: ReCode benchmark robustness evaluation on popular code generation models for MBPP.

It showed that larger models help improve worst-case robustness. But may risk overfitting as the relative performance drops under perturbations are significant.
Complete study also showed that wrong syntaxes (inputs to the model with followings: extra spaces, indentation errors, capital letter errors etc.)are the main reason for LLM models to fail, especially for CodeGen models.

BioCoder Benchmark

This benchmark is designed for challenging, practical bioinformatics scenarios. Both CodeGen1.0 and CodeGen2.0 models were one of the worst ones. It might be because CodeGen models are trained not to generate full-function but to do one-line completions.

Model	Prompt	Java				Python			
		Pass@1	Pass@5	Pass@10	Pass@20	Pass@1	Pass@5	Pass@10	Pass@20
InCoder-6B	*Summary at Top*	0	0	0	0	0.828	2.016	3.006	4.459
	Uncommented	0	0	0	0	0.032	0.159	0.318	0.637
	Summary Only	0	0	0	0	1.688	5.320	8.332	12.006
	Necessary Only	0	0	0	0	0.032	0.159	0.318	0.637
SantaCoder-1.1B	*Summary at Top*	0	0	0	0	0.637	1.338	1.844	2.548
	Uncommented	0	0	0	0	0.287	0.764	0.955	1.274
	Summary Only	0	0	0	0	2.965	9.848	14.227	18.181
	Necessary Only	0	0	0	0	0.032	0.159	0.318	0.637
StarCoder-15.5B	*Summary at Top*	0	0	0	0	3.694	13.197	19.359	24.554
	Uncommented	0	0	0	0	0.318	1.062	1.591	2.548
	Summary Only	0	0	0	0	4.682	15.225	21.200	27.166
	Necessary Only	0	0	0	0	0.127	0.603	1.123	1.911
StarCoder-15.5B (finetuned)	*Summary at top*	0	0	0	0	5.818	16.562	21.091	27.048
	Uncommented	0	0	0	0	3.312	9.073	12.574	17.536
	Summary Only	0.200	1.000	2.000	4.000	7.295	20.838	26.143	39.570
	Necessary Only	3.300	12.097	19.545	30.000	0.597	1.173	1.813	2.611
StarCoder+	*Summary at Top*	0	0	0	0	2.675	9.133	14.019	19.650
	Uncommented	0	0	0	0	0.510	0.955	1.274	1.911
	Summary Only	1.300	5.031	8.042	12.000	2.548	8.279	12.864	18.057
	Necessary Only	0	0	0	0	0.127	0.457	0.609	0.637
InstructCodeT5+	*All prompt types*	0	0	0	0	0	0	0	0
CodeGen-6B-mono	*Summary at Top*	0	0	0	0	0.637	0.637	0.637	0.637
	Uncommented	0	0	0	0	0	0	0	0
	Summary Only	0	0	0	0	0.637	0.637	0.637	0.637
	Necessary Only	0	0	0	0	0	0	0	0
CodeGen-16B-mono	*Summary at Top*	0	0	0	0	0.637	0.637	0.637	0.637
	Uncommented	0	0	0	0	0	0	0	0
	Summary Only	0	0	0	0	0.637	0.637	0.637	0.637
	Necessary Only	0	0	0	0	0	0	0	0
CodeGen2-7B	*Summary at Top*	0	0	0	0	0.637	0.637	0.637	0.637
	Uncommented	0	0	0	0	0.510	0.637	0.637	0.637
	Summary Only	0	0	0	0	0.860	2.494	3.962	6.242
	Necessary Only	0	0	0	0	0	0	0	0
GPT-3.5-Turbo	*Summary at Top*	4.100	7.235	8.989	11.600	22.771	33.461	36.551	39.490
	Uncommented	6.300	11.563	14.436	18.000	11.019	19.075	21.680	24.204
	Summary Only	17.400	33.199	37.878	42.000	24.682	33.997	37.132	40.127
	Necessary Only	43.500	52.582	53.995	55.400	28.758	39.529	44.029	47.771
GPT-4	*Summary at top*	1.100	5.500	11.000	22.000	10.701	25.500	32.910	39.490
	Uncommented	6.367	11.234	15.897	18.562	12.654	20.129	24.387	27.932
	Summary Only	19.483	24.721	29.634	2.543	13.172	24.578	28.394	31.938
	Necessary Only	**45.011**	**55.350**	**57.616**	**60.000**	**38.439**	**48.491**	**50.619**	**52.229**

Benchmarking LLMs for Automated Verilog RTL Code Generation

The results show that fine-tuning LLMs in programming languages other than Python and Java and in task specific tasks such as verilog code increases the success rate greatly and enables it to surpass even models such as GPT3.5 -4 in commercial use.

PASS@(SCENARIO*n) AT n =10 FOR TEST BENCH PASSING COMPLETIONS (PASS=PASSED FUNCTIONAL TESTS), PT = PRE-TRAINED, FT = FINE-TUNED. BOLDED VALUE IN EACH TEST COLUMN REFLECTS THE (BEST) HIGHEST PERFORMANCE FOR THAT PROBLEM SET AND DIFFICULTY.

Model	Model Type	Inference Time (s)	Basic			Intermediate			Advanced		
			L	M	H	L	M	H	L	M	H
MegatronLM-355M	PT	3.628	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	FT	0.175	0.170	0.591	0.245	0.043	0.018	0.025	0.000	0.000	0.000
CodeGen-2B	PT	1.478	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.016	0.020
	FT	0.665	0.835	0.350	0.630	0.130	0.092	0.163	0.132	0.048	0.068
CodeGen-6B	PT	2.332	0.000	0.000	0.000	0.000	0.000	0.013	0.000	0.000	0.000
	FT	0.710	**1.000**	0.500	0.760	0.135	0.150	0.168	**0.284**	0.164	0.164
J1-Large-7B	PT	7.146	0.044	0.058	0.067	0.000	0.000	0.021	0.000	0.000	0.000
	FT	2.029	0.388	0.283	0.342	0.125	0.075	0.200	0.000	0.000	0.000
CodeGen-16B	PT	2.835	0.000	0.085	0.055	0.035	0.003	0.045	0.012	0.000	0.016
	FT	1.994	0.745	**0.720**	0.745	**0.213**	**0.270**	**0.255**	0.246	**0.290**	0.294
code-davinci-002	PT	3.885	0.520	0.685	**0.775**	0.175	0.200	0.150	0.156	0.184	**0.344**

PASS@(SCENARIO*n) AT n=10 FOR COMPILED COMPLETIONS (PASS=COMPILING), PT = PRE-TRAINED, FT = FINE-TUNED. BOLD REFLECTS THE (BEST) HIGHEST PERFORMANCE FOR THAT DIFFICULTY.

Model	Model Type	Basic	Intermediate	Advanced
MegatronLM-345M	PT	0.000	0.000	0.000
	FT	0.730	0.391	0.165
CodeGen-2B	PT	0.080	0.065	0.176
	FT	0.902	0.612	0.592
CodeGen-6B	PT	0.052	0.152	0.187
	FT	**0.987**	0.689	**0.599**
J1-Large-7B	PT	0.182	0.176	0.108
	FT	0.882	0.635	0.588
CodeGen-16B	PT	0.132	0.203	0.240
	FT	0.942	**0.728**	0.596
code-davinci-002	PT	0.847	0.452	0.569

Fine-tuned CodeGen-16B LLM outperforms all LLMs. All fine-tuned LLMs outperform their pre-trained counterparts. Further, when analyzing functional correctness, a fine-tuned open-source CodeGen LLM can outperform the state-of-the-art commercial Codex LLM (6.5% overall)

Prompting with Pseudo-Code Instructions

This study explores if prompting via pseudo-code instructions helps improve the performance of pre-trained language models. It uses a dataset of pseudo-code prompts for 132 different tasks spanning classification, QA, and generative language tasks.

Model	Instruction Format	Classification Tasks			QA Tasks	Generation tasks	All Tasks		
		Macro F1	Micro F1	Weighted F1	ROUGE-L	ROUGE-L	ROUGE-L	ANLS	EM
Majority Class		**0.296**	**0.509**	**0.362**	-	-	-	-	-
CodeGen 2B	Code Instructions	**0.272**	**0.417**	**0.354**	**0.175**	**0.317**	**0.330**	**0.261**	**0.202**
	NL Instructions	0.068	0.306	0.239	0.154	0.254	0.265	0.195	0.147
CodeGen 6B	Code Instructions	**0.311**	**0.443**	**0.375**	**0.201**	**0.327**	**0.354**	**0.283**	**0.218**
	NL Instructions	0.052	0.278	0.215	0.132	0.271	0.257	0.187	0.134
BLOOM 3B	Code Instructions	**0.116**	**0.351**	**0.288**	0.147	**0.271**	**0.279**	**0.215**	**0.165**
	NL Instructions	0.082	0.275	0.214	**0.159**	0.234	0.250	0.180	0.132
BLOOM 7B	Code Instructions	**0.174**	**0.369**	**0.285**	0.150	**0.298**	**0.297**	**0.232**	**0.176**
	NL Instructions	0.046	0.247	0.203	**0.156**	0.276	0.247	0.172	0.122

It showed that there is a significant accuracy difference for instructions with natural language to Pseudocode for model's performance. Hence, prompting away to NL and as close as possible to Pl is the wiser choice for the model to perform better.

For below benchmarks none of the CodeGen models has not been tested yet.

Glue
SuperGlue
MMLU
Alpaca-Eval
Helm
Arc
Drop
TruthfulQA
Math
Gsm8k
BigBench
HellaSwag