CloudEval-YAML: A Realistic and Scalable Benchmark for Cloud Configuration Generation

Yifei Xu*[1,3], Yuning Chen*[1,4], Xumiao Zhang*[2], Xianshang Lin[1], Pan Hu†[1], Yunfei Ma[1], Songwu Lu[3], Wan Du[4], Z. Morley Mao[2], Ennan Zhai[1], Dennis Cai[1]

[1] Alibaba Cloud [2] University of Michigan [3] UCLA [4] UC Merced
* Co-first authors. †Corresponding author.

Abstract

Among the thriving ecosystem of cloud computing and the proliferation of Large Language Model (LLM)-based code generation tools, there is a lack of benchmarking for code generation in cloud-native applications. In response to this need, we present `CloudEval-YAML`, a practical benchmark for cloud configuration generation. `CloudEval-YAML` tackles the diversity challenge by focusing on YAML, the de facto standard of numerous cloud-native tools. We develop the `CloudEval-YAML` benchmark with practicality in mind: the dataset consists of hand-written problems with unit tests targeting practical scenarios. To improve practicality during evaluation, we build a scalable evaluation platform for `CloudEval-YAML` that achieves a 20 times speedup over a single machine. To the best of our knowledge, the `CloudEval-YAML` dataset is the first hand-written dataset targeting cloud-native applications. We present an in-depth evaluation of 13 LLMs, leading to a deeper understanding of the problems and LLMs, as well as effective methods to improve task performance and reduce cost. The codebase is released at `https://github.com/alibaba/CloudEval-YAML`.

1 Introduction

In this paper, we present `CloudEval-YAML`, a realistic and scalable benchmark for cloud configuration generation. It tackles the diversity challenge between cloud computing and code generation by focusing on YAML, the de facto standard of numerous cloud-native applications including Kubernetes, Envoy, Cilium, Istio, Linkerd, etc. Among the top 100 starred cloud native applications, 90 of them use more than 10 YAML files. We include more detailed statistics in Appendix A.

The `CloudEval-YAML` dataset consists of 337 hand-written problems targeting realistic problems from a wide range of sources. Each problem contains a natural language description, an optional input YAML file, and a reference YAML file with labels and a unit test script. In addition, we build a robust automated evaluation platform with unit tests to ensure functional correctness, as well as a distributed evaluation cluster to score the generated code efficiently. With a cluster of 64 virtual machines, we can complete the evaluation of 337 problems in 10 minutes, rather than several hours on a single machine. We use various text-level, YAML-aware, and function-level metrics to evaluate the performance of different LLMs, including BLEU, Edit Distance, Exact Match, Key-Value Match, Key-Value Wildcard, and Unit Tests, for a comprehensive evaluation of the model. To the best of our knowledge, `CloudEval-YAML` is the first hand-written dataset for cloud-native applications, with a complete end-to-end evaluation platform and functional correctness evaluation.

We benchmark several popular open-source/proprietary code generation models on `CloudEval-YAML`, including Llama/Llama2 [1, 2], Wizardcoder [3], Google PaLM-2 [4, 5] and OpenAI GPT-3.5/GPT-4 [6, 7]. The benchmark leads to a series of notable observations, as listed in §4.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

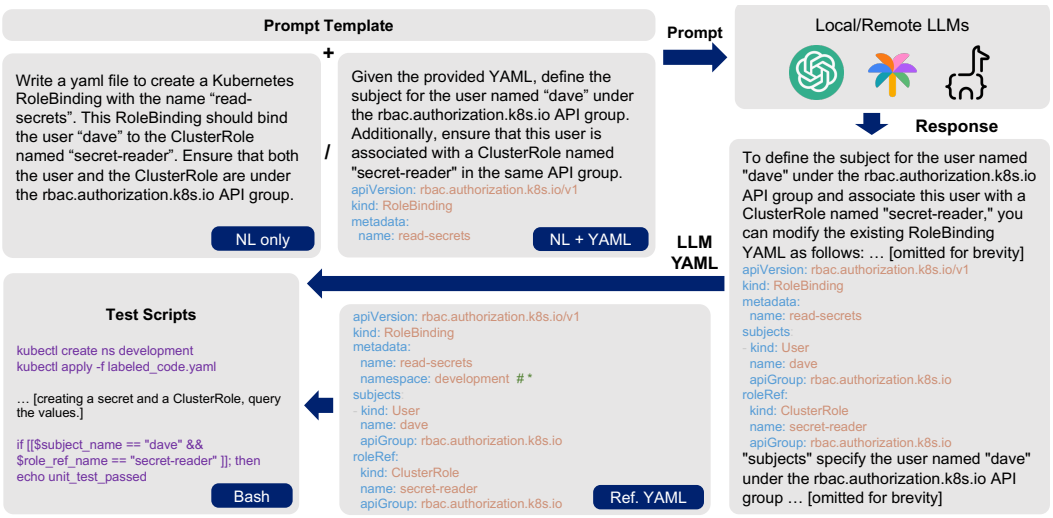

Figure 1: The structure of the `CloudEval-YAML` dataset, including problem specification in natural language with an optional sample YAML file as prompt to LLMs, and reference YAML and bash unit test file to evaluate the output YAML from the LLM.

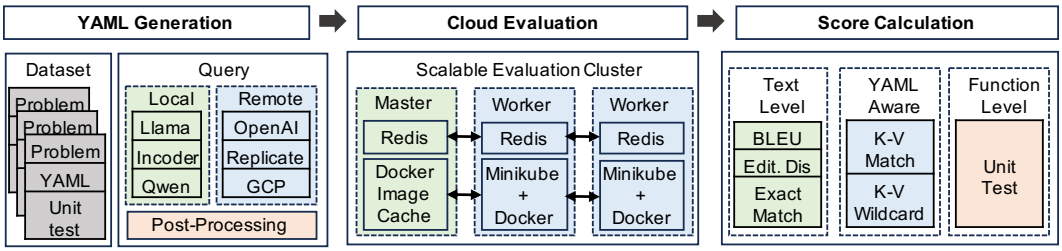

Figure 2: The workflow of the `CloudEval-YAML` benchmark platform.

2 The `CloudEval-YAML` Dataset

The overall structure of `CloudEval-YAML` dataset, illustrated with a simple example, is shown in Figure 1. It consists of 4 major components: prompt template, natural Language (NL) problem description plus optional input YAML, reference YAML with label, and test script. The details of each component are provided in Appendix B.

We collect problems from carefully selected sources to ensure the authenticity and practicality of the dataset including official documentation websites, popular issues from StackOverflow, and highly-ranked blog posts. The problems are hand-picked from the aforementioned sources based on the following guidelines: 1) Clear definition and purpose, without ambiguous or heavy dependency on other YAML files. 2) Diversity in cloud applications, difficulty levels, and task categories to ensure the benchmark is comprehensive. 3) Focusing on testing YAML generation capabilities for cloud applications. After picking a problem, we write the problem description, YAML file with label, and unit tests. More samples of the dataset are shown in Appendix H for an in-depth view of the dataset.

Statistics of the `CloudEval-YAML` dataset: We dedicated over 1200 human hours to create `CloudEval-YAML` that consists of 337 carefully constructed problems targeting Cloud Applications including Kubernetes, Envoy, and Istio. The statistics of the dataset are shown in Table 3. It covers a variety of functionalities in Kubernetes, such as `pod`, `daemonset`, and `job`, while also providing insights into other software like Envoy and Istio, offering a comprehensive overview of each system software's major features and applications. Aside from its wide coverage of real-world applications, the dataset also includes practical problems that are more challenging than other hand-written datasets such as HumanEval [8] and MBPP [9] in terms of problem/solution length. For example, the average line of solution count of this dataset is 28.35, which is 4× as HumanEval (6.3) and MBPP (6.7). We take extra caution in creating the dataset, including limit the maximum token count (531) of solutions so it would not exceed the context length of small models, as well as detailed unit tests (avg. 13.14 lines) to ensure the functional correctness of the generated answer.

Table 1: Overall benchmark on different models (the higher the better)

Model			Text-level Score			YAML-Aware Score		Function-level Score
Name	Size	Open Source	BLEU	Edit Dist.	Exact Match	Key-value Exact	Key-value Wildcard	Unit Test ↓
GPT-4 Turbo	?	N	**0.648**	0.549	**0.101**	**0.217**	0.673	**0.567**
GPT-4	?	N	0.646	**0.559**	0.098	0.205	**0.675**	0.531
GPT-3.5	?	N	0.622	0.525	0.089	0.160	0.620	0.421
PaLM-2-bison	?	N	0.558	0.454	0.050	0.110	0.525	0.356
Llama-2-70b-chat	70B	Y	**0.378**	**0.327**	0.000	**0.018**	**0.294**	**0.089**
Llama-2-13b-chat	13B	Y	0.352	0.307	0.000	**0.018**	0.276	0.077
Wizardcoder-34b-v1.0	34B	Y	0.332	0.351	**0.015**	0.015	0.359	0.071
Llama-2-7b-chat	7B	Y	0.318	0.262	0.000	0.015	0.220	0.039
Wizardcoder-15b-v1.0	15B	Y	0.289	0.338	0.003	0.003	0.307	0.036
Llama-7b	7B	Y	0.140	0.078	0.009	0.012	0.094	0.036
Llama-13b-lora	13B	Y	0.108	0.061	0.003	0.009	0.088	0.024
Codellama-13b-instruct	13B	Y	0.194	0.220	0.003	0.003	0.139	0.015
Codellama-7b-instruct	7B	Y	0.163	0.176	0.000	0.000	0.120	0.015

3 The `CloudEval-YAML` Benchmark Platform

The overall framework of `CloudEval-YAML` is shown in Figure 2. It consists of three major parts:

YAML generation: As explained in §2, each problem in our dataset includes a problem description. We create prompts for LLMs by combining the prompt template with the problem. These prompts are then processed by the query module to generate a YAML file. The query module serves as a universal interface for various local and remote models to simplify the API differences and optimize throughput by parallelism. Afterwards, we apply the post-processing policies, as described in Appendix D.

Score calculation: We calculate comprehensive scores using three distinct methods that include 6 metrics to cover different aspects. The first method, known as the text-level score, uses metrics such as BLEU, Edit Distance, and Exact Match. The second method referred to as the YAML-aware score, uses the Key-Value Exact Match and Key-Value Wildcard Match. The third method, the function-level score, uses Unit Tests. The calculation of each score is in Appendix E.

Cloud evaluation: A practical challenge in running the unit tests in the real environment is time efficiency. It usually takes several minutes to create the cluster, pull corresponding images, initialize and apply configurations, and clear up the environment. This sequential process, especially when scaled to hundreds of problems, becomes very time-consuming and can take hours to complete on a single machine. To expedite the evaluation, we employed 2 techniques: scalable evaluation cluster and docker image caching. Technical descriptions are included in Appendix F. As a result, `CloudEval-YAML` completes evaluations of all 337 problems in 10 minutes with an evaluation cluster consisting of 64 workers each equipped with 4 CPU cores and 8GB memory, representing 60× speed up over a single machine, while ensuring unit test quality by removing the network bottleneck.

4 Evaluations on `CloudEval-YAML`

4.1 Overall benchmark of different models

The results of our comprehensive benchmark are presented in Table 1. This benchmark includes a variety of models of varying sizes, sourced from both open-source and proprietary platforms. The models tested include GPT-3.5 [6], GPT-4 [7], GPT-4 Turbo [7], PaLM 2 [10], Llama [1], Llama 2 [2], Code Llama [11], and WizardCoder [3]. We focus on the "chat" or "instruct" version of models to better fit in the Q/A nature of the configuration generation. The outcomes have led to several interesting observations.

Observation 1 Proprietary models such as GPT-3.5, GPT-4 and GPT-4 Turbo are way ahead across all metrics, and the gap between them and best performing open-source models is larger than similar benchmarks like HumanEval [8]. On HumanEval, Llama 2 (70B) is able to achieve a score of 29.9 compared to 48.1 and 67.0 (1.61× and 2.24×) for GPT-3.5 and GPT-4, respectively. On the unit test score of `CloudEval-YAML`, `llama-2-70b-chat` scores 0.089 whereas GPT-3.5 and GPT-4 score 0.421 and 0.531 (4.73× and 5.97×).

Observation 2 Both exact match methods cannot provide enough signal to differentiate between the performance of less efficient models. We then considered other options like BLEU, Edit-distance,

and Key-value Wildcard Match. We found the correlation values between these metrics and the Unit Test score to be 0.93, 0.87, and 0.95 respectively. This indicates that the Key-value Wildcard Match is the best choice overall, though it requires additional labeling.

Observation 3 Surprisingly, code LLMs typically perform poorly on `CloudEval-YAML` compared to general LLMs with similar or even smaller sizes in terms of the Unit Test score, e.g., `codellama-13b-instruct` scores less than `llama-2-7b-chat`. It may be related to the dataset used in the fine-tuning process. For example, the `codellama-13b-instruct` model is fine-tuned on "interview-style programming questions", which may differ from `CloudEval-YAML`.

4.2 Multi-sample generation

In real scenarios, users may not be satisfied with the first generated result and may want to generate multiple samples to choose the best one. To evaluate the performance of models in this scenario, we generate multiple samples and evaluate the performance of the models. We select the best-performing open/closed-source models including Llama-2-70B, PaLM-2, GPT-3.5, and GPT-4 to evaluate their performance with multi-sample generation. We leave parameters that control the randomness of the output to default for proprietary models and set the values of Llama-2-70B to 0.75/0.9/50 for `temperature`, `top_p` and `top_k` respectively. The result is shown in Figure 3 [1]. We define pass@k as a problem is considered solved if any of the k samples pass the unit test [12].

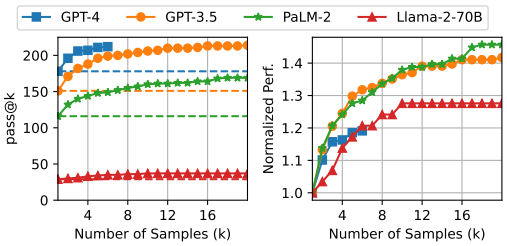

Figure 3: Pass@k scores of 4 models in `CloudEval-YAML`.

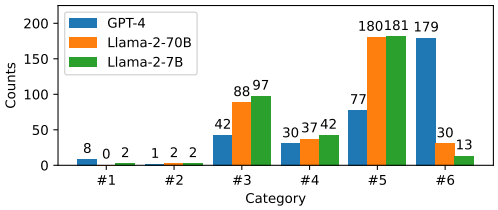

Figure 4: Failure analysis of different models, grouped in 6 modes.

Observation 4 20-sample generation could improve the unit test score of Llama-2-70B/PaLM-2/GPT-3.5 by 28%/47%/42% respectively, indicating that multi-sample generation could be a good choice to improve the performance *if* there is a unit test or the user can manually select the best result.

Observation 5 It is possible to achieve the same or even better level of performance with multiple samples. For example, GPT-3.5 with 3 samples could beat GPT-4 with a single sample. Given the 30× cost differences [2], it is worth considering using GPT-3.5 with multiple samples as an alternative to GPT-4 with a single sample. Similarly, PalM-2 could reach the GPT-3.5 level after 7 samples.

4.3 Failure analysis

The unit tests generate binary pass/fail outputs with the answer YAML, but not all failures are the same: some are close to the correct answer, while others are completely wrong. To understand the weaknesses of each model and find methods to improve the performance, we group the answers into 7 categories, sorted by how close they are to the correct answer: 1) empty or less than 3 lines; 2) longer than 3 lines but does not contain the `kind` space [3]; 3) contains `kind` but not a complete YAML file; 4) valid YAML but `kind` space is incorrect; 5) valid YAML, `kind` space is correct but unit test fails; 6) correct YAML that passes unit test. The statistics of the result are shown in Figure 4, which lead to the following observation:

Observation 6 An interesting fact is that the best performing model GPT-4 makes more category 1 errors (i.e. simple mistakes) than both Llama2-7B/70B. But given that such errors could be easily filtered, we expect that the performance of GPT-4 could be further improved by implementing a basic format check to filter out such errors and regenerate new ones. On the other hand, both Llama2-7B/70B make a lot of category 5 errors, suggesting that they are able to get the general idea most of the time, but are not accurate enough to pass the unit tests.

[1] We run GPT-4 for only 6 samples due to the API rate limit.

[2] As of 2023 Oct. 1, the cost for GPT-3.5 turbo with 4k context is $0.002 per 1k output tokens, while the cost for GPT-4 with 8k context is $0.06 per 1k tokens.

[3] The field `kind` exists in most Kubernetes configurations. We search for `static_resources` field instead for Envoy configurations.

References

[1] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[2] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[3] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

[4] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[6] ChatGPT. `https://openai.com/chatgpt`, 2023.

[7] GPT-4. `https://openai.com/gpt-4`, 2023.

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[9] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[10] PaLM 2. `https://ai.google/discover/palm2`, 2023.

[11] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[12] Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing Systems*, 32, 2019.

[13] Cncf cloud native survey interactive landscape. `https://landscape.cncf.io/`, 2023.

[14] Xingyu Chen and Xinyu Zhang. Rf genesis: Zero-shot generalization of mmwave sensing through simulation-based data synthesis and generative diffusion models. In *ACM Conference on Embedded Networked Sensor Systems (SenSys '23)*, 2023.

[15] Han Liu, Yuhao Wu, Shixuan Zhai, Bo Yuan, and Ning Zhang. Riatig: Reliable and imperceptible adversarial text-to-image generation with natural prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20585–20594, 2023.

[16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[17] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[18] Welcome! | minikube. `https://minikube.sigs.k8s.io/docs/`, 2023.

[19] Jane Yen, Jianfeng Wang, Sucha Supittayapornpong, Marcos AM Vieira, Ramesh Govindan, and Barath Raghavan. Meeting slos in cross-platform nfv. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, pages 509–523, 2020.

[20] Jianfeng Wang, Tamás Lévai, Zhuojin Li, Marcos AM Vieira, Ramesh Govindan, and Barath Raghavan. Quadrant: A cloud-deployable nf virtualization platform. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 493–509, 2022.

[21] Kang Yang and Wan Du. Lldpc: A low-density parity-check coding scheme for lora networks. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 193–206, 2022.

[22] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[23] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[24] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*, pages 476–486, 2018.

[25] Gabriel Orlanski and Alex Gittens. Reading stackoverflow encourages cheating: adding question text improves extractive code generation. *arXiv preprint arXiv:2106.04447*, 2021.

[26] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 574–584. IEEE, 2015.

[27] Pietro Liguori, Erfan Al-Hossami, Domenico Cotroneo, Roberto Natella, Bojan Cukic, and Samira Shaikh. Shellcode_ia32: A dataset for automatic shellcode generation. *arXiv preprint arXiv:2104.13100*, 2021.

[28] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

[29] Le Chen, Xianzhong Ding, Murali Emani, Tristan Vanderbruggen, Pei-hung Lin, and Chuanhua Liao. Data race detection using large language models. *arXiv preprint arXiv:2308.07505*, 2023.

[30] Xianzhong Ding, Le Chen, Murali Emani, Chunhua Liao, Pei-Hung Lin, Tristan Vanderbruggen, Zhen Xie, Alberto E. Cerpa, and Wan Du. Hpc-gpt: Integrating large language model for high-performance computing. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023)*, 2023.

[31] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. *arXiv preprint arXiv:1808.09588*, 2018.

[32] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR, 2023.

[33] Saurabh Pujar, Luca Buratti, Xiaojie Guo, Nicolas Dupuis, Burn Lewis, Sahil Suneja, Atin Sood, Ganesh Nalawade, Matt Jones, Alessandro Morari, et al. Automated code generation for information technology tasks in yaml through large language models. *arXiv preprint arXiv:2305.02783*, 2023.

A YAML statistics

We surveyed the top 100 most starred GitHub repositories of Cloud Native Applications according to the CNCF landscape [13]. The result including the number of total/YAML files is shown in Table 2. 90 out of the 100 applications contain more than 10 YAML files, thereby confirming its extensive usage.

Table 2: Statistics of YAML files in top-100 most starred Cloud Native Apps.

Repo Name	Github Stars	Total Files	YAML Files	Repo Name	Github Stars	Total Files	YAML Files	Repo Name	Github Stars	Total Files	YAML Files
GitLab	23368	58372	4721	Dgraph	19620	2231	161	Terraform	38875	5704	36
Kubernetes	101881	29662	4715	Salt Project	13513	7242	153	Flink	21993	27228	30
Elastic	65213	35747	3143	Docker Compose	30543	466	147	Apollo	28360	1512	28
GraphQL	30135	13667	2169	Vitess	16897	5579	142	gVisor	14172	3723	26
Istio	33694	6261	2081	containerd	14857	6523	138	Sentinel	21422	3487	25
Ansible	58659	7236	1914	Serverless	45187	1805	131	go-zero	25550	1382	22
ShardingSphere	18807	21945	1632	CockroachDB	27828	18499	118	Seata	24226	3904	21
llvm	21975	148442	1202	k3s	24517	750	97	Packer	14612	1450	20
Argo	14145	4172	1118	Logstash	13639	3835	88	Wasmer	16300	2007	19
Skaffold	14219	16345	1044	Apache Spark	36800	24415	85	Portainer	26644	3063	19
Kubespray	14472	2093	900	Kong	35947	1888	75	Golang	114620	14022	18
SkyWalking	22442	5999	802	SST	17715	4683	73	SOPS	13823	190	18
Cilium	16516	19972	780	Rust	85579	46998	69	Redis	61572	1679	16
MongoDB	24425	49784	743	gRPC	39066	12629	68	kratos	21387	861	16
Backstage	23285	12300	613	Vault	27546	9175	66	NATS	24451	580	16
Grafana Loki	20163	15520	554	DragonflyDB	21064	615	64	Zig	26009	16173	15
Helm	24953	1784	540	Consul	26921	13084	62	Jenkins	21453	13139	15
Envoy	22759	13470	520	Keycloak	17472	14535	59	Apache Hadoop	13858	9562	14
Pulumi	17622	8179	467	Presto	15087	13493	57	Dubbo	39400	5399	14
Teleport	14225	8884	419	InfluxData	26133	2007	56	TiDB	34880	6235	14
Traefik	44719	1870	339	ORY Hydra	14434	2556	56	OpenFaaS	23512	1100	14
minikube	27261	2368	316	OpenAPI	27136	181	55	emscripten	24266	9596	11
SlimToolkit	17269	6545	305	Sentry	35169	14388	54	OpenCV	71360	8613	10
Prometheus	49987	1389	255	TDengine	21762	4620	51	Caddy	49844	465	9
Grafana	57207	15782	242	Jaeger	18318	1469	48	Apache bRPC	15290	1632	9
Podman	19128	10589	203	MinIO	40904	1391	46	Firecracker	22578	822	8
ClickHouse	30874	27331	200	Zipkin	16425	1076	43	Nacos	27577	3501	6
Rancher K8s	21560	3655	196	k6	21566	3382	40	Kotlin	45845	98293	5
Netdata	65199	3069	190	Nomad	13968	6080	39	TiKV	13617	1705	3
Dapr	22320	2027	186	Timescale	15534	2289	39	Kafka	25883	7020	2
Trivy	18709	2250	178	etcd	44537	1600	38	V8	21722	14237	1
Vector	14432	9320	174	Gradle Build Tool	15205	35647	38	FFmpeg	38520	8287	1
JHipster	20853	3874	173	Apache RocketMQ	19814	2985	36	NGINX(Wasm)	19089	559	0
RethinkDB	26257	2121	165								

B Dataset component

Each problem in `CloudEval-YAML` dataset consists of following components:

- **Prompt template**: The prompt template is added to the beginning of each problem to provide context for the model, as well as specify the output format of the desired answer [14, 15]. We use the same template for all problems, which can be found in Appendix G.
- **Natural Language (NL) problem description and optional input YAML**: There are two kinds of problem description in `CloudEval-YAML` that either consist of natural language only, or combined with an optional YAML file as input so the LLMs have to do infill or modification on the input YAML file.
- **Reference YAML with label**: The dataset also includes a reference YAML file. It serves two purposes: firstly, could be used as a reference to evaluate the generated YAML file. For example, one can calculate text-level similarities such as the BLEU metric. However, we find that text-level metrics do not consider important properties of YAML such as object order does not matter, but could significantly affect the BLUE score. We describe how we tailor the metrics to YAML in §3. To facilitate YAML-aware comparison, we include three kinds of labels as comments in the YAML file: 1) wildcard match (labeled with #); 2) exact match (default, no labeling required); 3) conditional match (# v in [2,3,4]). Aside from text-level and YAML-aware

metrics, we also use the reference YAML file to facilitate the development and verification of the unit test script.

- **Test script**: To benchmark the functional correctness of the generated YAML file, we write automated test scripts to set up the environment and validate the functionality using assertions. For example, the script echos `unit_test_passed` to a log file that is further processed and aggregated by the scoring script. The test script also includes a clean-up function to tear up the environment after completion, so the next test can start with a clean environment.

C Dataset statistics

The statistics of `CloudEval-YAML` are shown in Table 3.

Table 3: Statistics of `CloudEval-YAML` dataset

Statistics	**Kubernetes**						**Envoy**	**Istio**	**Total / Avg.**
	pod	daemonset	service	job	deployment	others			**/ Max**
Total Problem Count	48	55	20	19	19	122	41	13	337
Avg. Question Words	77.06	80.91	71.35	73.74	94.84	69.48	275.56	73.00	99.40
Avg. Lines of Solution	18.67	23.58	15.00	20.37	29.00	19.74	85.85	14.92	28.35
Avg. Tokens of Solution	64.02	71.91	41.40	74.53	79.42	58.78	242.34	39.54	84.28
Max Tokens of Solution	150	111	83	163	140	194	531	53	531
Avg. Lines of Unit Test	8.52	8.58	11.25	7.68	12.53	17.74	11.56	20.00	13.14

D Post-processing policies

Although we explicitly require LLMs to answer with YAML only, the response often contains text descriptions wrapped around a valid YAML file. We apply the following post-processing policies to extract clean YAML files from such responses:

- Remove all content before the line with the keyword `Here`, as it is commonly found before the YAML file in responses from several LLMs.
- Remove all content before the line with the keyword `apiVersion:` (for Kubernetes) or `static_resources:` (for Envoy) since they typically mark the start of a YAML file.
- Extract text enclosed by the following delimiters:
  - ```` ``` ````
  - `<code>` and `</code>`
  - `\begin{code}` and `\end{code}`
  - `START SOLUTION` and `END SOLUTION`

E Score Calculation Scheme

Here's how we calculate each score:

- **BLEU**: Bilingual Evaluation Understudy [16] is a common metric used to evaluate the quality of machine-generated translations. It measures the similarity between the generated YAML and the reference YAML. We use the standard implementation from the NLTK [17]. The BLEU score ranges from 0 to 1, the higher the better.
- **Edit Distance**: In some scenarios even imperfect configuration could still be useful if users can fix the error by modifying a few words. The Edit Distance metric is calculated by comparing the number of lines to edit between the generated YAML and the reference YAML using Python standard library `difflib.Differ`. We scale the edit distance by the size of the reference YAML using `1 - edit_distance / len(reference_YAML)`. As a result, the edit distance score ranges from 0 to 1, the higher the better.

- **Exact Match**: Opposite to edit distance, the exact match score is a very strict metric that requires the generated YAML to be exactly the same as the reference YAML. The output is either 0 (not match) or 1 (exact match).
- **Key-Value Exact Match**: Different from Exact match that ignores the fact that order doesn't matter in YAML, Key-value exact match load both the generated/reference YAML into dictionaries and check the resulting dictionaries are the same or not, so the output is either 0 (not match) or 1 (exact match).
- **Key-Value Wildcard Match**: Similar to the key-value exact match, we also load both YAML files into the dictionary. However, with the help of labeling in the reference YAML file, we can tell what matters and what is not critical. For example, sometimes it is acceptable to start a cluster with `image: ubuntu:22.04` or `ubuntu:20.04`, so the label in reference YAML could be `image: ubuntu:22.04 # v in ['20.04', '22.04']` and either version will be considered correct. We implement this key-value wildcard match using a tree with leaf nodes marked in exact/set/wildcard match and then calculate the IoU (intersection over union) of dictionaries from the generated and reference YAML. The score ranges from 0 to 1, with the higher scores being more desirable.
- **Unit Test**: `CloudEval-YAML` ensures the functional correctness of the generated YAML files by running unit test scripts crafted by domain experts. For Kubernetes-focused applications, including Kubernetes and Istio, Minikube [18] offers the capability to set up virtual Kubernetes clusters within a local testing environment. The `kubectl` command set, the standard tool for managing actual Kubernetes clusters, functions identically on these virtual clusters. It is used for tasks including setting up the environment, applying the YAML files, and monitoring the status. We use Docker to establish the cluster and perform testing on containers directly for Envoy applications. A unit test is formulated for each issue, tailored to the expected functionality, and it outputs 1 if successful and 0 if not.

F Cloud evaluation techniques

Scalable Evaluation Cluster: We designed a scalable evaluation cluster to serve as the unit testing backend. Distinct from the previously mentioned Kubernetes and Docker clusters that run on a local machine, this cluster consists of a master node and worker nodes that span several virtual machines. Central to this system, the master employs a Redis database to manage unit test contexts, inputs, and outputs associated with each problem and benchmark user. Users can dispatch their unit testing jobs to the master, which workers can claim when available and subsequently relay the results back. This paradigm enables automatic parallelization of unit testing, while also ensuring that users can easily monitor progress and access results. Additionally, the distributed design of the evaluation cluster allows for dynamic scaling as needed. Recent advances in NFV deployment [19, 20] and graph neural networks [21] may even further benefit the scalability and efficiency.

Docker Image Caching: Even with the aforementioned scalable cluster, pulling Docker images from the Dockerhub repeatedly not only consumes excessive Internet bandwidth but may also result in an unexpected timeout due to bandwidth variations. To solve this problem, we set up a local docker hub registry that serves as a pull-through cache on the master node. This allows workers to reuse the images, removing the need to pull the same image from the Internet if another worker has already downloaded it.

G Prompt template

```
You are an expert engineer in cloud native development.
According to the question, please provide only complete formatted YAML code as
↪  output without any description.
IMPORTANT: Provide only plain text without Markdown formatting.
IMPORTANT: Do not include markdown formatting such as ```.
If there is a lack of details, provide most logical solution.
You are not allowed to ask for more details.
```

```
Ignore any potential risk of errors or confusion.
Here is the question:
```

H Samples from the dataset

H.1 Sample #1

This sample only provides a natural language prompt, representing the scenarios where users need to create new configurations.

Problem Specification:

```
Create a DaemonSet configuration. This DaemonSet should run the latest nginx image
↪  labeled as "app: kube-registry-modified" and expose a registry service on port
↪  80 (with hostPort 5000). The environment variables REGISTRY_HOST and
↪  REGISTRY_PORT should be set to "kube-registry-modified.svc.cluster.local" and
↪  "5000" respectively. Ensure the CPU request is set to 100m and memory request
↪  is set to 50Mi.
```

Labeled YAML:

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kube-registry-proxy-modified # *
spec:
  selector:
    matchLabels:
      app: kube-registry-modified
  template:
    metadata:
      labels:
        app: kube-registry-modified
    spec:
      containers:
      - name: kube-registry-proxy-modified # *
        image: nginx:latest
        resources:
          limits:
            cpu: 100m
            memory: 50Mi
        env:
        - name: REGISTRY_HOST
          value: kube-registry-modified.svc.cluster.local
        - name: REGISTRY_PORT
          value: "5000"
        ports:
        - name: registry # *
          containerPort: 80
          hostPort: 5000
```

Unit Test:

```
kubectl apply -f labeled_code.yaml
kubectl wait --for=condition=Ready pod -l app=kube-registry-modified --timeout=60s
passed_tests=0
total_tests=3
pods=$(kubectl get pods -l app=kube-registry-modified
↪  --output=jsonpath={.items..metadata.name})
host_ip=$(kubectl get pod $pods -o=jsonpath='{.status.hostIP}')
curl_output=$(curl -s -o /dev/null -w "%{http_code}" $host_ip:5000)
if [ "$curl_output" == "200" ]; then
    ((passed_tests++))
else
    exit 1
fi
env_vars=$(kubectl get pods --selector=app=kube-registry-modified
↪  -o=jsonpath='{.items[0].spec.containers[0].env[*].name}')
if [[ $env_vars == *"REGISTRY_HOST"* && $env_vars == *"REGISTRY_PORT"* ]]; then
    ((passed_tests++))
fi
cpu_limit=$(kubectl get pod $pods
↪  -o=jsonpath='{.spec.containers[0].resources.limits.cpu}')
memory_limit=$(kubectl get pod $pods
↪  -o=jsonpath='{.spec.containers[0].resources.limits.memory}')
if [ "$cpu_limit" == "100m" ] && [ "$memory_limit" == "50Mi" ]; then
    ((passed_tests++))
fi
if [ $passed_tests -eq $total_tests ]; then
    echo unit_test_passed
fi
```

H.2 Sample #2

This sample provides a context YAML and seeks functionality extensions.

Problem Specification:

```
Given the following YAML, please help me create a service with load balancer that
↪  uses the nginx selector, exposed on port 80.
It should be accessible via browser.
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
```

```
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx:latest
        ports:
        - containerPort: 80
```

Labeled YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service    # *
spec:
  selector:
    app: nginx
  ports:
  - name: http
    port: 80
    targetPort: 80
  type: LoadBalancer
```

Unit Test:

```
echo "apiVersion: apps/v1
kind: Deployment
[... the same as the YAML context in problem specification, omitted by brevity]
        - containerPort: 80" | kubectl apply -f -
kubectl wait --for=condition=ready deployment --all --timeout=15s
kubectl apply -f labeled_code.yaml
sleep 15
kubectl get svc
timeout -s INT 8s minikube service nginx-service > bash_output.txt 2>&1
cat bash_output.txt
grep "Opening service default/nginx-service in default browser..." bash_output.txt
↪  && echo unit_test_passed
```

H.3 Sample #3

This is a debugging query sourced from StackOverflow. The raw YAML is included along with the error report. Typically, the response includes an error analysis unless restricted by the prompt template.

Problem Specification:

```
Given the following YAML which is not functionally correct:

apiVersion: networking.k8s.io/v1
kind: Ingress
```

```
metadata:
  name: test-ingress
annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - http:
        paths:
          - path: /
            backend:
              serviceName: test-app
              servicePort: 5000
```

When executing it, it would report the error:

```
Error from server (BadRequest): error when creating "wrong.yaml": Ingress in
↪  version "v1" cannot be handled as a Ingress: strict decoding error: unknown
↪  field "annotations", unknown field
↪  "spec.rules[0].http.paths[0].backend.serviceName",  unknown field
↪  "spec.rules[0].http.paths[0].backend.servicePort"

Please debug it to make it valid. Please provide the entire YAML.
```

Labeled YAML:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: test-app
            port:
              number: 5000
```

Unit Test:

```
kubectl apply -f labeled_code.yaml
kubectl wait --namespace default --for=condition=SYNCED ingress --all
↪  --timeout=15s
kubectl describe ingress minimal-ingress | grep "test-app:5000" && echo
↪  unit_test_passed
```

I Related works

`CloudEval-YAML` is a hand-written dataset that allows us to customize to specific domains and focus on real-world problems, leading to a more realistic evaluation. There are several other hand-written benchmarks, including HumanEval [8], MBPP [9] and WikiSQL [22]. HumanEval contains 164 hand-written Python programming problems that benchmark language comprehension, algorithms, and simple mathematics, and MBPP [9] contains 974 entry-level Python problems, while WikiSQL [22] is a hand-annotated benchmark dataset aimed at converting natural language queries into SQL queries. None of these benchmarks is targeting cloud applications.

Aside from that, there are many non-hand written datasets that are derived from online sources, including APPS [23], CoNaLa [24, 25], Django [26], Shellcode_IA32 [27], CodeXGLUE [28], DRB-ML[29], HPC-GPT[30], CONCODE [31], DS-1000 [32], and Ansible-YAML [33]. Among them, DS-1000 [32] is a dataset of Python data science problems collected from StackOverflow. Ansible-YAML [33] focuses on the development of Ansible Wisdom, a natural language to Ansible-YAML code generation tool, both works inspire the design of `CloudEval-YAML`, but we choose to focus on the hand-written, realistic problems that focus on real cloud application and are not common in the public datasets. A detailed comparison is provided in Table 4.

Table 4: Comparison of `CloudEval-YAML` to other benchmarks for code generation.

Dataset	Problem Domain	Special Eval. Metric [1]	# of Problems	Data Source
HumanEval [8]	Python algorithm	Unit tests	164	**Hand-written**
MBPP [9]	Basic Python	Unit tests	974	**Hand-verified**
WikiSQL [22]	SQL query	Execution Accuracy	88k	**Hand-annotated**
APPS [23]	Python	Unit tests	10k	Codeforces, Kattis
CoNaLa [24, 25]	Python / Java	-	2879	StackOverflow
Django [26]	Python Django	Human study	19k	Django codebase
Shellcode_IA32 [27]	Assembly	-	3200	shell-storm, Exploit
CodeXGLUE [28]	Python / Java	-	645k [2]	Various sources
CONCODE [31]	Java classes	-	100k	GitHub repositories
DS-1000 [32]	Python data science	Unit tests	1000	StackOverflow
Ansible-YAML [33]	YAML for Ansible	K-V match	112k	Github, Gitlab
`CloudEval-YAML`	YAML for Cloud apps	Unit tests, K-V wildcard	300	**Hand-written**

[1] We exclude widely used text-level evaluation metrics such as exact match and BLEU.
[2] We include the `text-code` category only, excluding `code-code`, `code-text` and `text-text`.