[Document(page\_content='YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object\ndetectors\nChien-Yao Wang1, Alexey Bochkovskiy, and Hong-Yuan Mark Liao1\n1Institute of Information Science, Academia Sinica, Taiwan\[nkinyiu@iis.sinica.edu.tw](mailto:nkinyiu@iis.sinica.edu.tw), [alexeyab84@gmail.com](mailto:alexeyab84@gmail.com), and [liao@iis.sinica.edu.tw](mailto:liao@iis.sinica.edu.tw)\nAbstract\nYOLOv7 surpasses all known object detectors in both\nspeed and accuracy in the range from 5 FPS to 160 FPS\nand has the highest accuracy 56.8% AP among all known\nreal-time object detectors with 30 FPS or higher on GPU\nV100. YOLOv7-E6 object detector (56 FPS V100, 55.9%\nAP) outperforms both transformer-based detector SWIN-\nL Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by\n509% in speed and 2% in accuracy, and convolutional-\nbased detector ConvNeXt-XL Cascade-Mask R-CNN (8.6\nFPS A100, 55.2% AP) by 551% in speed and 0.7% AP\nin accuracy, as well as YOLOv7 outperforms: YOLOR,\nYOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable\nDETR, DINO-5scale-R50, ViT-Adapter-B and many other\nobject detectors in speed and accuracy. Moreover, we train\nYOLOv7 only on MS COCO dataset from scratch without\nusing any other datasets or pre-trained weights. Source\ncode is released in [https://github.com/WongKinYiu/yolov7.\n1](https://github.com/WongKinYiu/yolov7./n1). Introduction\nReal-time object detection is a very important topic in\ncomputer vision, as it is often a necessary component in\ncomputer vision systems. For example, multi-object track-\ning [94, 93], autonomous driving [40, 18], robotics [35, 58],\nmedical image analysis [34, 46], etc. The computing de-\nvices that execute real-time object detection is usually some\nmobile CPU or GPU, as well as various neural processing\nunits (NPU) developed by major manufacturers. For exam-\nple, the Apple neural engine (Apple), the neural compute\nstick (Intel), Jetson AI edge devices (Nvidia), the edge TPU\n(Google), the neural processing engine (Qualcomm), the AI\nprocessing unit (MediaTek), and the AI SoCs (Kneron), are\nall NPUs. Some of the above mentioned edge devices focus\non speeding up different operations such as vanilla convolu-\ntion, depth-wise convolution, or MLP operations. In this pa-\nper, the real-time object detector we proposed mainly hopes\nthat it can support both mobile GPU and GPU devices from\nthe edge to the cloud.\nIn recent years, the real-time object detector is still de-\nveloped for different edge device. For example, the devel-\nFigure 1: Comparison with other real-time object detectors, our\nproposed methods achieve state-of-the-arts performance.\nopment of MCUNet [49, 48] and NanoDet [54] focused on\nproducing low-power single-chip and improving the infer-\nence speed on edge CPU. As for methods such as YOLOX\n[21] and YOLOR [81], they focus on improving the infer-\nence speed of various GPUs. More recently, the develop-\nment of real-time object detector has focused on the de-\nsign of efﬁcient architecture. As for real-time object de-\ntectors that can be used on CPU [54, 88, 84, 83], their de-\nsign is mostly based on MobileNet [28, 66, 27], ShufﬂeNet\n[92, 55], or GhostNet [25]. Another mainstream real-time\nobject detectors are developed for GPU [81, 21, 97], they\nmostly use ResNet [26], DarkNet [63], or DLA [87], and\nthen use the CSPNet [80] strategy to optimize the architec-\nture. The development direction of the proposed methods in\nthis paper are different from that of the current mainstream\nreal-time object detectors. In addition to architecture op-\ntimization, our proposed methods will focus on the opti-\nmization of the training process. Our focus will be on some\noptimized modules and optimization methods which may\nstrengthen the training cost for improving the accuracy of\nobject detection, but without increasing the inference cost.\nWe call the proposed modules and optimization methods\ntrainable bag-of-freebies.\n1arXiv:2207.02696v1 [cs.CV] 6 Jul 2022', metadata={'source': 'papers/yolov7paper.pdf', 'page': 0}),

Document(page\_content='Recently, model re-parameterization [13, 12, 29] and dy-\nnamic label assignment [20, 17, 42] have become important\ntopics in network training and object detection. Mainly af-\nter the above new concepts are proposed, the training of\nobject detector evolves many new issues. In this paper, we\nwill present some of the new issues we have discovered and\ndevise effective methods to address them. For model re-\nparameterization, we analyze the model re-parameterization\nstrategies applicable to layers in different networks with the\nconcept of gradient propagation path, and propose planned\nre-parameterized model. In addition, when we discover that\nwith dynamic label assignment technology, the training of\nmodel with multiple output layers will generate new issues.\nThat is: “How to assign dynamic targets for the outputs of\ndifferent branches?” For this problem, we propose a new\nlabel assignment method called coarse-to-ﬁne lead guided\nlabel assignment.\nThe contributions of this paper are summarized as fol-\nlows: (1) we design several trainable bag-of-freebies meth-\nods, so that real-time object detection can greatly improve\nthe detection accuracy without increasing the inference\ncost; (2) for the evolution of object detection methods, we\nfound two new issues, namely how re-parameterized mod-\nule replaces original module, and how dynamic label as-\nsignment strategy deals with assignment to different output\nlayers. In addition, we also propose methods to address the\ndifﬁculties arising from these issues; (3) we propose “ex-\ntend” and “compound scaling” methods for the real-time\nobject detector that can effectively utilize parameters and\ncomputation; and (4) the method we proposed can effec-\ntively reduce about 40% parameters and 50% computation\nof state-of-the-art real-time object detector, and has faster\ninference speed and higher detection accuracy.\n2. Related work\n2.1. Real-time object detectors\nCurrently state-of-the-art real-time object detectors are\nmainly based on YOLO [61, 62, 63] and FCOS [76, 77],\nwhich are [3, 79, 81, 21, 54, 85, 23]. Being able to become\na state-of-the-art real-time object detector usually requires\nthe following characteristics: (1) a faster and stronger net-\nwork architecture; (2) a more effective feature integration\nmethod [22, 97, 37, 74, 59, 30, 9, 45]; (3) a more accurate\ndetection method [76, 77, 69]; (4) a more robust loss func-\ntion [96, 64, 6, 56, 95, 57]; (5) a more efﬁcient label assign-\nment method [99, 20, 17, 82, 42]; and (6) a more efﬁcient\ntraining method. In this paper, we do not intend to explore\nself-supervised learning or knowledge distillation methods\nthat require additional data or large model. Instead, we will\ndesign new trainable bag-of-freebies method for the issues\nderived from the state-of-the-art methods associated with\n(4), (5), and (6) mentioned above.2.2. Model re-parameterization\nModel re-parametrization techniques [71, 31, 75, 19, 33,\n11, 4, 24, 13, 12, 10, 29, 14, 78] merge multiple compu-\ntational modules into one at inference stage. The model\nre-parameterization technique can be regarded as an en-\nsemble technique, and we can divide it into two cate-\ngories, i.e., module-level ensemble and model-level ensem-\nble. There are two common practices for model-level re-\nparameterization to obtain the ﬁnal inference model. One\nis to train multiple identical models with different train-\ning data, and then average the weights of multiple trained\nmodels. The other is to perform a weighted average of the\nweights of models at different iteration number. Module-\nlevel re-parameterization is a more popular research issue\nrecently. This type of method splits a module into multi-\nple identical or different module branches during training\nand integrates multiple branched modules into a completely\nequivalent module during inference. However, not all pro-\nposed re-parameterized module can be perfectly applied to\ndifferent architectures. With this in mind, we have devel-\noped new re-parameterization module and designed related\napplication strategies for various architectures.\n2.3. Model scaling\nModel scaling [72, 60, 74, 73, 15, 16, 2, 51] is a way\nto scale up or down an already designed model and make\nit ﬁt in different computing devices. The model scaling\nmethod usually uses different scaling factors, such as reso-\nlution (size of input image), depth (number of layer), width\n(number of channel), and stage (number of feature pyra-\nmid), so as to achieve a good trade-off for the amount of\nnetwork parameters, computation, inference speed, and ac-\ncuracy. Network architecture search (NAS) is one of the\ncommonly used model scaling methods. NAS can automat-\nically search for suitable scaling factors from search space\nwithout deﬁning too complicated rules. The disadvantage\nof NAS is that it requires very expensive computation to\ncomplete the search for model scaling factors. In [15], the\nresearcher analyzes the relationship between scaling factors\nand the amount of parameters and operations, trying to di-\nrectly estimate some rules, and thereby obtain the scaling\nfactors required by model scaling. Checking the literature,\nwe found that almost all model scaling methods analyze in-\ndividual scaling factor independently, and even the methods\nin the compound scaling category also optimized scaling\nfactor independently. The reason for this is because most\npopular NAS architectures deal with scaling factors that are\nnot very correlated. We observed that all concatenation-\nbased models, such as DenseNet [32] or V oVNet [39], will\nchange the input width of some layers when the depth of\nsuch models is scaled. Since the proposed architecture is\nconcatenation-based, we have to design a new compound\nscaling method for this model.\n2', metadata={'source': 'papers/yolov7paper.pdf', 'page': 1}),

Document(page\_content='Figure 2: Extended efﬁcient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmis-\nsion path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the\nfeatures of different groups in a shufﬂe and merge cardinality manner. This way of operation can enhance the features learned by different\nfeature maps and improve the use of parameters and calculations.\n3. Architecture\n3.1. Extended efﬁcient layer aggregation networks\nIn most of the literature on designing the efﬁcient ar-\nchitectures, the main considerations are no more than the\nnumber of parameters, the amount of computation, and the\ncomputational density. Starting from the characteristics of\nmemory access cost, Ma et al. [55] also analyzed the in-\nﬂuence of the input/output channel ratio, the number of\nbranches of the architecture, and the element-wise opera-\ntion on the network inference speed. Doll ´aret al. [15] addi-\ntionally considered activation when performing model scal-\ning, that is, to put more consideration on the number of el-\nements in the output tensors of convolutional layers. The\ndesign of CSPV oVNet [79] in Figure 2 (b) is a variation of\nV oVNet [39]. In addition to considering the aforementioned\nbasic designing concerns, the architecture of CSPV oVNet\n[79] also analyzes the gradient path, in order to enable the\nweights of different layers to learn more diverse features.\nThe gradient analysis approach described above makes in-\nferences faster and more accurate. ELAN [1] in Figure 2 (c)\nconsiders the following design strategy – “How to design an\nefﬁcient network?.” They came out with a conclusion: By\ncontrolling the shortest longest gradient path, a deeper net-\nwork can learn and converge effectively. In this paper, we\npropose Extended-ELAN (E-ELAN) based on ELAN and\nits main architecture is shown in Figure 2 (d).\nRegardless of the gradient path length and the stacking\nnumber of computational blocks in large-scale ELAN, it has\nreached a stable state. If more computational blocks are\nstacked unlimitedly, this stable state may be destroyed, and\nthe parameter utilization rate will decrease. The proposedE-ELAN uses expand, shufﬂe, merge cardinality to achieve\nthe ability to continuously enhance the learning ability of\nthe network without destroying the original gradient path.\nIn terms of architecture, E-ELAN only changes the archi-\ntecture in computational block, while the architecture of\ntransition layer is completely unchanged. Our strategy is\nto use group convolution to expand the channel and car-\ndinality of computational blocks. We will apply the same\ngroup parameter and channel multiplier to all the compu-\ntational blocks of a computational layer. Then, the feature\nmap calculated by each computational block will be shuf-\nﬂed into ggroups according to the set group parameter g,\nand then concatenate them together. At this time, the num-\nber of channels in each group of feature map will be the\nsame as the number of channels in the original architec-\nture. Finally, we add ggroups of feature maps to perform\nmerge cardinality. In addition to maintaining the original\nELAN design architecture, E-ELAN can also guide differ-\nent groups of computational blocks to learn more diverse\nfeatures.\n3.2. Model scaling for concatenation-based models\nThe main purpose of model scaling is to adjust some at-\ntributes of the model and generate models of different scales\nto meet the needs of different inference speeds. For ex-\nample the scaling model of EfﬁcientNet [72] considers the\nwidth, depth, and resolution. As for the scaled-YOLOv4\n[79], its scaling model is to adjust the number of stages. In\n[15], Doll ´aret al. analyzed the inﬂuence of vanilla convolu-\ntion and group convolution on the amount of parameter and\ncomputation when performing width and depth scaling, and\nused this to design the corresponding model scaling method.\n3', metadata={'source': 'papers/yolov7paper.pdf', 'page': 2}),

Document(page\_content='Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on\nconcatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width\nof the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-\nbased models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with\ncorresponding width scaling.\nThe above methods are mainly used in architectures such as\nPlainNet or ResNet. When these architectures are in execut-\ning scaling up or scaling down, the in-degree and out-degree\nof each layer will not change, so we can independently an-\nalyze the impact of each scaling factor on the amount of\nparameters and computation. However, if these methods\nare applied to the concatenation-based architecture, we will\nﬁnd that when scaling up or scaling down is performed on\ndepth, the in-degree of a translation layer which is immedi-\nately after a concatenation-based computational block will\ndecrease or increase, as shown in Figure 3 (a) and (b).\nIt can be inferred from the above phenomenon that\nwe cannot analyze different scaling factors separately for\na concatenation-based model but must be considered to-\ngether. Take scaling-up depth as an example, such an ac-\ntion will cause a ratio change between the input channel and\noutput channel of a transition layer, which may lead to a de-\ncrease in the hardware usage of the model. Therefore, we\nmust propose the corresponding compound model scaling\nmethod for a concatenation-based model. When we scale\nthe depth factor of a computational block, we must also cal-\nculate the change of the output channel of that block. Then,\nwe will perform width factor scaling with the same amount\nof change on the transition layers, and the result is shown\nin Figure 3 (c). Our proposed compound scaling method\ncan maintain the properties that the model had at the initial\ndesign and maintains the optimal structure.\n4. Trainable bag-of-freebies\n4.1. Planned re-parameterized convolution\nAlthough RepConv [13] has achieved excellent perfor-\nmance on the VGG [68], when we directly apply it to\nResNet [26] and DenseNet [32] and other architectures,\nits accuracy will be signiﬁcantly reduced. We use gradi-\nent ﬂow propagation paths to analyze how re-parameterized\nconvolution should be combined with different network.\nWe also designed planned re-parameterized convolution ac-\ncordingly.\nFigure 4: Planned re-parameterized model. In the proposed\nplanned re-parameterized model, we found that a layer with resid-\nual or concatenation connections, its RepConv should not have\nidentity connection. Under these circumstances, it can be replaced\nby RepConvN that contains no identity connections.\nRepConv actually combines 3×3convolution, 1×1\nconvolution, and identity connection in one convolutional\nlayer. After analyzing the combination and correspond-\ning performance of RepConv and different architectures,\nwe ﬁnd that the identity connection in RepConv destroys\nthe residual in ResNet and the concatenation in DenseNet,\nwhich provides more diversity of gradients for different fea-\nture maps. For the above reasons, we use RepConv with-\nout identity connection (RepConvN) to design the architec-\nture of planned re-parameterized convolution. In our think-\ning, when a convolutional layer with residual or concate-\nnation is replaced by re-parameterized convolution, there\nshould be no identity connection. Figure 4 shows an exam-\nple of our designed “planned re-parameterized convolution”\nused in PlainNet and ResNet. As for the complete planned\nre-parameterized convolution experiment in residual-based\nmodel and concatenation-based model, it will be presented\nin the ablation study session.\n4', metadata={'source': 'papers/yolov7paper.pdf', 'page': 3}),

Document(page\_content='Figure 5: Coarse for auxiliary and ﬁne for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head.\nDifferent from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-ﬁne lead head\nguided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training\nlead head and auxiliary head at the same time. The detailed coarse-to-ﬁne implementation method and constraint design details will be\nelaborated in Apendix.\n4.2. Coarse for auxiliary and ﬁne for lead loss\nDeep supervision [38] is a technique that is often used\nin training deep networks. Its main concept is to add\nextra auxiliary head in the middle layers of the network,\nand the shallow network weights with assistant loss as the\nguide. Even for architectures such as ResNet [26] and\nDenseNet [32] which usually converge well, deep supervi-\nsion [70, 98, 67, 47, 82, 65, 86, 50] can still signiﬁcantly\nimprove the performance of the model on many tasks. Fig-\nure 5 (a) and (b) show, respectively, the object detector ar-\nchitecture “without” and “with” deep supervision. In this\npaper, we call the head responsible for the ﬁnal output as\nthe lead head, and the head used to assist training is called\nauxiliary head.\nNext we want to discuss the issue of label assignment. In\nthe past, in the training of deep network, label assignment\nusually refers directly to the ground truth and generate hard\nlabel according to the given rules. However, in recent years,\nif we take object detection as an example, researchers often\nuse the quality and distribution of prediction output by the\nnetwork, and then consider together with the ground truth to\nuse some calculation and optimization methods to generate\na reliable soft label [61, 8, 36, 99, 91, 44, 43, 90, 20, 17, 42].\nFor example, YOLO [61] use IoU of prediction of bounding\nbox regression and ground truth as the soft label of object-\nness. In this paper, we call the mechanism that considers\nthe network prediction results together with the ground truth\nand then assigns soft labels as “label assigner.”\nDeep supervision needs to be trained on the target ob-\njectives regardless of the circumstances of auxiliary head or\nlead head. During the development of soft label assigner re-\nlated techniques, we accidentally discovered a new deriva-\ntive issue, i.e., “How to assign soft label to auxiliary head\nand lead head ?” To the best of our knowledge, the relevant\nliterature has not explored this issue so far. The results of\nthe most popular method at present is as shown in Figure 5\n(c), which is to separate auxiliary head and lead head, and\nthen use their own prediction results and the ground truthto execute label assignment. The method proposed in this\npaper is a new label assignment method that guides both\nauxiliary head and lead head by the lead head prediction.\nIn other words, we use lead head prediction as guidance to\ngenerate coarse-to-ﬁne hierarchical labels, which are used\nfor auxiliary head and lead head learning, respectively. The\ntwo proposed deep supervision label assignment strategies\nare shown in Figure 5 (d) and (e), respectively.\nLead head guided label assigner is mainly calculated\nbased on the prediction result of the lead head and the\nground truth, and generate soft label through the optimiza-\ntion process. This set of soft labels will be used as the tar-\nget training model for both auxiliary head and lead head.\nThe reason to do this is because lead head has a relatively\nstrong learning capability, so the soft label generated from it\nshould be more representative of the distribution and corre-\nlation between the source data and the target. Furthermore,\nwe can view such learning as a kind of generalized residual\nlearning. By letting the shallower auxiliary head directly\nlearn the information that lead head has learned, lead head\nwill be more able to focus on learning residual information\nthat has not yet been learned.\nCoarse-to-ﬁne lead head guided label assigner also\nused the predicted result of the lead head and the ground\ntruth to generate soft label. However, in the process we gen-\nerate two different sets of soft label, i.e., coarse label and\nﬁne label, where ﬁne label is the same as the soft label gen-\nerated by lead head guided label assigner, and coarse label\nis generated by allowing more grids to be treated as posi-\ntive target by relaxing the constraints of the positive sample\nassignment process. The reason for this is that the learning\nability of an auxiliary head is not as strong as that of a lead\nhead, and in order to avoid losing the information that needs\nto be learned, we will focus on optimizing the recall of aux-\niliary head in the object detection task. As for the output\nof lead head, we can ﬁlter the high precision results from\nthe high recall results as the ﬁnal output. However, we must\nnote that if the additional weight of coarse label is close to\n5', metadata={'source': 'papers/yolov7paper.pdf', 'page': 4}),

Document(page\_content='Table 1: Comparison of baseline object detectors.\nModel #Param. FLOPs Size APvalAPval\n50APval\n75APval\nSAPval\nMAPval\nL\nYOLOv4 [3] 64.4M 142.8G 640 49.7% 68.2% 54.3% 32.9% 54.8% 63.7%\nYOLOR-u5 (r6.1) [81] 46.5M 109.1G 640 50.2% 68.7% 54.6% 33.2% 55.5% 63.7%\nYOLOv4-CSP [79] 52.9M 120.4G 640 50.3% 68.6% 54.9% 34.2% 55.6% 65.1%\nYOLOR-CSP [81] 52.9M 120.4G 640 50.8% 69.5% 55.3% 33.7% 56.0% 65.4%\nYOLOv7 36.9M 104.7G 640 51.2% 69.7% 55.5% 35.2% 56.0% 66.7%\nimprovement -43% -15% - +0.4 +0.2 +0.2 +1.5 = +1.3\nYOLOR-CSP-X [81] 96.9M 226.8G 640 52.7% 71.3% 57.4% 36.3% 57.5% 68.3%\nYOLOv7-X 71.3M 189.9G 640 52.9% 71.1% 57.5% 36.9% 57.7% 68.6%\nimprovement -36% -19% - +0.2 -0.2 +0.1 +0.6 +0.2 +0.3\nYOLOv4-tiny [79] 6.1 6.9 416 24.9% 42.1% 25.7% 8.7% 28.4% 39.2%\nYOLOv7-tiny 6.2 5.8 416 35.2% 52.8% 37.3% 15.7% 38.0% 53.4%\nimprovement +2% -19% - +10.3 +10.7 +11.6 +7.0 +9.6 +14.2\nYOLOv4-tiny-3l [79] 8.7 5.2 320 30.8% 47.3% 32.2% 10.9% 31.9% 51.5%\nYOLOv7-tiny 6.2 3.5 320 30.8% 47.3% 32.2% 10.0% 31.9% 52.2%\nimprovement -39% -49% - = = = -0.9 = +0.7\nYOLOR-E6 [81] 115.8M 683.2G 1280 55.7% 73.2% 60.7% 40.1% 60.4% 69.2%\nYOLOv7-E6 97.2M 515.2G 1280 55.9% 73.5% 61.1% 40.6% 60.3% 70.0%\nimprovement -19% -33% - +0.2 +0.3 +0.4 +0.5 -0.1 +0.8\nYOLOR-D6 [81] 151.7M 935.6G 1280 56.1% 73.9% 61.2% 42.4% 60.5% 69.9%\nYOLOv7-D6 154.7M 806.8G 1280 56.3% 73.8% 61.4% 41.3% 60.6% 70.1%\nYOLOv7-E6E 151.7M 843.2G 1280 56.8% 74.4% 62.1% 40.8% 62.1% 70.6%\nimprovement = -11% - +0.7 +0.5 +0.9 -1.6 +1.6 +0.7\nthat of ﬁne label, it may produce bad prior at ﬁnal predic-\ntion. Therefore, in order to make those extra coarse positive\ngrids have less impact, we put restrictions in the decoder,\nso that the extra coarse positive grids cannot produce soft\nlabel perfectly. The mechanism mentioned above allows\nthe importance of ﬁne label and coarse label to be dynam-\nically adjusted during the learning process, and makes the\noptimizable upper bound of ﬁne label always higher than\ncoarse label.\n4.3. Other trainable bag-of-freebies\nIn this section we will list some trainable bag-of-\nfreebies. These freebies are some of the tricks we used\nin training, but the original concepts were not proposed\nby us. The training details of these freebies will be elab-\norated in the Appendix, including (1) Batch normalization\nin conv-bn-activation topology: This part mainly connects\nbatch normalization layer directly to convolutional layer.\nThe purpose of this is to integrate the mean and variance\nof batch normalization into the bias and weight of convolu-\ntional layer at the inference stage. (2) Implicit knowledge\nin YOLOR [81] combined with convolution feature map in\naddition and multiplication manner: Implicit knowledge in\nYOLOR can be simpliﬁed to a vector by pre-computing at\nthe inference stage. This vector can be combined with the\nbias and weight of the previous or subsequent convolutional\nlayer. (3) EMA model: EMA is a technique used in mean\nteacher [75], and in our system we use EMA model purely\nas the ﬁnal inference model.5. Experiments\n5.1. Experimental setup\nWe use Microsoft COCO dataset to conduct experiments\nand validate our object detection method. All our experi-\nments did not use pre-trained models. That is, all models\nwere trained from scratch. During the development pro-\ncess, we used train 2017 set for training, and then used val\n2017 set for veriﬁcation and choosing hyperparameters. Fi-\nnally, we show the performance of object detection on the\ntest 2017 set and compare it with the state-of-the-art object\ndetection algorithms. Detailed training parameter settings\nare described in Appendix.\nWe designed basic model for edge GPU, normal GPU,\nand cloud GPU, and they are respectively called YOLOv7-\ntiny, YOLOv7, and YOLOv7-W6. At the same time, we\nalso use basic model for model scaling for different ser-\nvice requirements and get different types of models. For\nYOLOv7, we do stack scaling on neck, and use the pro-\nposed compound scaling method to perform scaling-up of\nthe depth and width of the entire model, and use this to ob-\ntain YOLOv7-X. As for YOLOv7-W6, we use the newly\nproposed compound scaling method to obtain YOLOv7-E6\nand YOLOv7-D6. In addition, we use the proposed E-\nELAN for YOLOv7-E6, and thereby complete YOLOv7-\nE6E. Since YOLOv7-tiny is an edge GPU-oriented archi-\ntecture, it will use leaky ReLU as activation function. As\nfor other models we use SiLU as activation function. We\nwill describe the scaling factor of each model in detail in\nAppendix.\n6', metadata={'source': 'papers/yolov7paper.pdf', 'page': 5}),

Document(page\_content='Table 2: Comparison of state-of-the-art real-time object detectors.\nModel #Param. FLOPs Size FPS APtest/APvalAPtest\n50APtest\n75APtest\nSAPtest\nMAPtest\nL\nYOLOX-S [21] 9.0M 26.8G 640 102 40.5% / 40.5% - - - - -\nYOLOX-M [21] 25.3M 73.8G 640 81 47.2% / 46.9% - - - - -\nYOLOX-L [21] 54.2M 155.6G 640 69 50.1% / 49.7% - - - - -\nYOLOX-X [21] 99.1M 281.9G 640 58 51.5% / 51.1% - - - - -\nPPYOLOE-S [85] 7.9M 17.4G 640 208 43.1% / 42.7% 60.5% 46.6% 23.2% 46.4% 56.9%\nPPYOLOE-M [85] 23.4M 49.9G 640 123 48.9% / 48.6% 66.5% 53.0% 28.6% 52.9% 63.8%\nPPYOLOE-L [85] 52.2M 110.1G 640 78 51.4% / 50.9% 68.9% 55.6% 31.4% 55.3% 66.1%\nPPYOLOE-X [85] 98.4M 206.6G 640 45 52.2% / 51.9% 69.9% 56.5% 33.3% 56.3% 66.4%\nYOLOv5-N (r6.1) [23] 1.9M 4.5G 640 159 - / 28.0% - - - - -\nYOLOv5-S (r6.1) [23] 7.2M 16.5G 640 156 - / 37.4% - - - - -\nYOLOv5-M (r6.1) [23] 21.2M 49.0G 640 122 - / 45.4% - - - - -\nYOLOv5-L (r6.1) [23] 46.5M 109.1G 640 99 - / 49.0% - - - - -\nYOLOv5-X (r6.1) [23] 86.7M 205.7G 640 83 - / 50.7% - - - - -\nYOLOR-CSP [81] 52.9M 120.4G 640 106 51.1% / 50.8% 69.6% 55.7% 31.7% 55.3% 64.7%\nYOLOR-CSP-X [81] 96.9M 226.8G 640 87 53.0% / 52.7% 71.4% 57.9% 33.7% 57.1% 66.8%\nYOLOv7-tiny-SiLU 6.2M 13.8G 640 286 38.7% / 38.7% 56.7% 41.7% 18.8% 42.4% 51.9%\nYOLOv7 36.9M 104.7G 640 161 51.4% / 51.2% 69.7% 55.9% 31.8% 55.5% 65.0%\nYOLOv7-X 71.3M 189.9G 640 114 53.1% / 52.9% 71.2% 57.8% 33.8% 57.1% 67.4%\nYOLOv5-N6 (r6.1) [23] 3.2M 18.4G 1280 123 - / 36.0% - - - - -\nYOLOv5-S6 (r6.1) [23] 12.6M 67.2G 1280 122 - / 44.8% - - - - -\nYOLOv5-M6 (r6.1) [23] 35.7M 200.0G 1280 90 - / 51.3% - - - - -\nYOLOv5-L6 (r6.1) [23] 76.8M 445.6G 1280 63 - / 53.7% - - - - -\nYOLOv5-X6 (r6.1) [23] 140.7M 839.2G 1280 38 - / 55.0% - - - - -\nYOLOR-P6 [81] 37.2M 325.6G 1280 76 53.9% / 53.5% 71.4% 58.9% 36.1% 57.7% 65.6%\nYOLOR-W6 [81] 79.8G 453.2G 1280 66 55.2% / 54.8% 72.7% 60.5% 37.7% 59.1% 67.1%\nYOLOR-E6 [81] 115.8M 683.2G 1280 45 55.8% / 55.7% 73.4% 61.1% 38.4% 59.7% 67.7%\nYOLOR-D6 [81] 151.7M 935.6G 1280 34 56.5% / 56.1% 74.1% 61.9% 38.9% 60.4% 68.7%\nYOLOv7-W6 70.4M 360.0G 1280 84 54.9% / 54.6% 72.6% 60.1% 37.3% 58.7% 67.1%\nYOLOv7-E6 97.2M 515.2G 1280 56 56.0% / 55.9% 73.5% 61.2% 38.0% 59.9% 68.4%\nYOLOv7-D6 154.7M 806.8G 1280 44 56.6% / 56.3% 74.0% 61.8% 38.8% 60.1% 69.5%\nYOLOv7-E6E 151.7M 843.2G 1280 36 56.8% / 56.8% 74.4% 62.1% 39.3% 60.5% 69.0%\n1Our FLOPs is calaculated by rectangle input resolution like 640 ×640 or 1280 ×1280.\n2Our inference time is estimated by using letterbox resize input image to make its long side equals to 640 or 1280.\n5.2. Baselines\nWe choose previous version of YOLO [3, 79] and state-\nof-the-art object detector YOLOR [81] as our baselines. Ta-\nble 1 shows the comparison of our proposed YOLOv7 mod-\nels and those baseline that are trained with the same settings.\nFrom the results we see that if compared with YOLOv4,\nYOLOv7 has 75% less parameters, 36% less computation,\nand brings 1.5% higher AP. If compared with state-of-the-\nart YOLOR-CSP, YOLOv7 has 43% fewer parameters, 15%\nless computation, and 0.4% higher AP. In the performance\nof tiny model, compared with YOLOv4-tiny-31, YOLOv7-\ntiny reduces the number of parameters by 39% and the\namount of computation by 49%, but maintains the same AP.\nOn the cloud GPU model, our model can still have a higher\nAP while reducing the number of parameters by 19% and\nthe amount of computation by 33%.5.3. Comparison with state-of-the-arts\nWe compare the proposed method with state-of-the-art\nobject detectors for general GPUs and Mobile GPUs, and\nthe results are shown in Table 2. From the results in\nTable 2 we know that the proposed method has the best\nspeed-accuracy trade-off comprehensively. If we compare\nYOLOv7-tiny-SiLU with YOLOv5-N (r6.1), our method\nis 127 fps faster and 10.7% more accurate on AP. In ad-\ndition, YOLOv7 has 51.4% AP at frame rate of 161 fps,\nwhile PPYOLOE-L with the same AP has only 78 fps frame\nrate. In terms of parameter usage, YOLOv7 is 41% less than\nPPYOLOE-L. If we compare YOLOv7-X with 114 fps in-\nference speed to YOLOv5-L (r6.1) with 99 fps inference\nspeed, YOLOv7-X can improve AP by 3.9%. If YOLOv7-\nX is compared with YOLOv5-X (r6.1) of similar scale, the\ninference speed of YOLOv7-X is 31 fps faster. In addi-\ntion, in terms of the amount of parameters and computation,\nYOLOv7-X reduces 22% of parameters and 8% of compu-\ntation compared to YOLOv5-X (r6.1), but improves AP by\n2.2%.\n7', metadata={'source': 'papers/yolov7paper.pdf', 'page': 6}),

Document(page\_content='If we compare YOLOv7 with YOLOR using the input\nresolution 1280, the inference speed of YOLOv7-W6 is 8\nfps faster than that of YOLOR-P6, and the detection rate is\nalso increased by 1% AP. As for the comparison between\nYOLOv7-E6 and YOLOv5-X6 (r6.1), the former has 0.9%\nAP gain than the latter, 45% less parameters and 63% less\ncomputation, and the inference speed is increased by 47%.\nYOLOv7-D6 has close inference speed to YOLOR-E6, but\nimproves AP by 0.8%. YOLOv7-E6E has close inference\nspeed to YOLOR-D6, but improves AP by 0.3%.\n5.4. Ablation study\n5.4.1 Proposed compound scaling method\nTable 3 shows the results obtained when using different\nmodel scaling strategies for scaling up. Among them, our\nproposed compound scaling method is to scale up the depth\nof computational block by 1.5 times and the width of tran-\nsition block by 1.25 times. If our method is compared with\nthe method that only scaled up the width, our method can\nimprove the AP by 0.5% with less parameters and amount\nof computation. If our method is compared with the method\nthat only scales up the depth, our method only needs to in-\ncrease the number of parameters by 2.9% and the amount of\ncomputation by 1.2%, which can improve the AP by 0.2%.\nIt can be seen from the results of Table 3 that our proposed\ncompound scaling strategy can utilize parameters and com-\nputation more efﬁciently.\nTable 3: Ablation study on proposed model scaling.\nModel #Param. FLOPs Size APvalAPval\n50APval\n75\nbase (v7-X light) 47.0M 125.5G 640 51.7% 70.1% 56.0%\nwidth only (1.25 w)73.4M 195.5G 640 52.4% 70.9% 57.1%\ndepth only (2.0 d) 69.3M 187.6G 640 52.7% 70.8% 57.3%\ncompound (v7-X) 71.3M 189.9G 640 52.9% 71.1% 57.5%\nimprovement - - - +1.2 +1.0 +1.5\n5.4.2 Proposed planned re-parameterized model\nIn order to verify the generality of our proposed planed\nre-parameterized model, we use it on concatenation-based\nmodel and residual-based model respectively for veriﬁca-\ntion. The concatenation-based model and residual-based\nmodel we chose for veriﬁcation are 3-stacked ELAN and\nCSPDarknet, respectively.\nIn the experiment of concatenation-based model, we re-\nplace the 3×3convolutional layers in different positions in\n3-stacked ELAN with RepConv, and the detailed conﬁgura-\ntion is shown in Figure 6. From the results shown in Table 4\nwe see that all higher AP values are present on our proposed\nplanned re-parameterized model.\nIn the experiment dealing with residual-based model,\nsince the original dark block does not have a 3×3con-\nFigure 6: Planned RepConv 3-stacked ELAN. Blue circles are the\nposition we replace Conv by RepConv.\nTable 4: Ablation study on planned RepConcatenation model.\nModel APvalAPval\n50APval\n75APval\nSAPval\nMAPval\nL\nbase (3-S ELAN) 52.26% 70.41% 56.77% 35.81% 57.00% 67.59%\nFigure 6 (a) 52.18% 70.34% 56.90% 35.71% 56.83% 67.51%\nFigure 6 (b) 52.30% 70.30% 56.92% 35.76% 56.95% 67.74%\nFigure 6 (c) 52.33% 70.56% 56.91% 35.90% 57.06% 67.50%\nFigure 6 (d) 52.17% 70.32% 56.82% 35.33% 57.06% 68.09%\nFigure 6 (e) 52.23% 70.20% 56.81% 35.34% 56.97% 66.88%\nvolution block that conforms to our design strategy, we ad-\nditionally design a reversed dark block for the experiment,\nwhose architecture is shown in Figure 7. Since the CSP-\nDarknet with dark block and reversed dark block has exactly\nthe same amount of parameters and operations, it is fair to\ncompare. The experiment results illustrated in Table 5 fully\nconﬁrm that the proposed planned re-parameterized model\nis equally effective on residual-based model. We ﬁnd that\nthe design of RepCSPResNet [85] also ﬁt our design pat-\ntern.\nFigure 7: Reversed CSPDarknet. We reverse the position of 1×1\nand3×3convolutional layer in dark block to ﬁt our planned re-\nparameterized model design strategy.\nTable 5: Ablation study on planned RepResidual model.\nModel APvalAPval\n50APval\n75APval\nSAPval\nMAPval\nL\nbase (YOLOR-W6) 54.82% 72.39% 59.95% 39.68% 59.38% 68.30%\nRepCSP 54.67% 72.50% 59.58% 40.22% 59.61% 67.87%\nRCSP 54.36% 71.95% 59.54% 40.15% 59.02% 67.44%\nRepRCSP 54.85% 72.51% 60.08% 40.53% 59.52% 68.06%\nbase (YOLOR-CSP) 50.81% 69.47% 55.28% 33.74% 56.01% 65.38%\nRepRCSP 50.91% 69.54% 55.55% 34.44% 55.74% 65.46%\n8', metadata={'source': 'papers/yolov7paper.pdf', 'page': 7}),

Document(page\_content='Figure 8: Objectness map predicted by different methods at auxiliary head and lead head.\n5.4.3 Proposed assistant loss for auxiliary head\nIn the assistant loss for auxiliary head experiments, we com-\npare the general independent label assignment for lead head\nand auxiliary head methods, and we also compare the two\nproposed lead guided label assignment methods. We show\nall comparison results in Table 6. From the results listed in\nTable 6, it is clear that any model that increases assistant\nloss can signiﬁcantly improve the overall performance. In\naddition, our proposed lead guided label assignment strat-\negy receives better performance than the general indepen-\ndent label assignment strategy in AP, AP 50, and AP 75. As\nfor our proposed coarse for assistant and ﬁne for lead label\nassignment strategy, it results in best results in all cases. In\nFigure 8 we show the objectness map predicted by different\nmethods at auxiliary head and lead head. From Figure 8 we\nﬁnd that if auxiliary head learns lead guided soft label, it\nwill indeed help lead head to extract the residual informa-\ntion from the consistant targets.\nTable 6: Ablation study on proposed auxiliary head.\nModel Size APvalAPval\n50APval\n75\nbase (v7-E6) 1280 55.6% 73.2% 60.7%\nindependent 1280 55.8% 73.4% 60.9%\nlead guided 1280 55.9% 73.5% 61.0%\ncoarse-to-ﬁne lead guided 1280 55.9% 73.5% 61.1%\nimprovement - +0.3 +0.3 +0.4\nIn Table 7 we further analyze the effect of the proposed\ncoarse-to-ﬁne lead guided label assignment method on the\ndecoder of auxiliary head. That is, we compared the results\nof with/without the introduction of upper bound constraint.\nJudging from the numbers in the Table, the method of con-\nstraining the upper bound of objectness by the distance from\nthe center of the object can achieve better performance.\nTable 7: Ablation study on constrained auxiliary head.\nModel Size APvalAPval\n50APval\n75\nbase (v7-E6) 1280 55.6% 73.2% 60.7%\naux without constraint 1280 55.9% 73.5% 61.0%\naux with constraint 1280 55.9% 73.5% 61.1%\nimprovement - +0.3 +0.3 +0.4Since the proposed YOLOv7 uses multiple pyramids to\njointly predict object detection results, we can directly con-\nnect auxiliary head to the pyramid in the middle layer for\ntraining. This type of training can make up for informa-\ntion that may be lost in the next level pyramid prediction.\nFor the above reasons, we designed partial auxiliary head\nin the proposed E-ELAN architecture. Our approach is to\nconnect auxiliary head after one of the sets of feature map\nbefore merging cardinality, and this connection can make\nthe weight of the newly generated set of feature map not\ndirectly updated by assistant loss. Our design allows each\npyramid of lead head to still get information from objects\nwith different sizes. Table 8 shows the results obtained us-\ning two different methods, i.e., coarse-to-ﬁne lead guided\nand partial coarse-to-ﬁne lead guided methods. Obviously,\nthe partial coarse-to-ﬁne lead guided method has a better\nauxiliary effect.\nTable 8: Ablation study on partial auxiliary head.\nModel Size APvalAPval\n50APval\n75\nbase (v7-E6E) 1280 56.3% 74.0% 61.5%\naux 1280 56.5% 74.0% 61.6%\npartial aux 1280 56.8% 74.4% 62.1%\nimprovement - +0.5 +0.4 +0.6\n6. Conclusions\nIn this paper we propose a new architecture of real-\ntime object detector and the corresponding model scaling\nmethod. Furthermore, we ﬁnd that the evolving process\nof object detection methods generates new research top-\nics. During the research process, we found the replace-\nment problem of re-parameterized module and the alloca-\ntion problem of dynamic label assignment. To solve the\nproblem, we propose the trainable bag-of-freebies method\nto enhance the accuracy of object detection. Based on the\nabove, we have developed the YOLOv7 series of object de-\ntection systems, which receives the state-of-the-art results.\n7. Acknowledgements\nThe authors wish to thank National Center for High-\nperformance Computing (NCHC) for providing computa-\ntional and storage resources.\n9', metadata={'source': 'papers/yolov7paper.pdf', 'page': 8}),

Document(page\_content='Table 9: More comparison (batch=1, no-TRT, without extra object detection training data)\nModel #Param. FLOPs Size FPSV100APtest/APvalAPtest\n50APtest\n75\nYOLOv7-tiny-SiLU 6.2M 13.8G 640 286 38.7% /38.7% 56.7% 41.7%\nPPYOLOE-S [85] 7.9M 17.4G 640 208 43.1% /42.7% 60.5% 46.6%\nYOLOv7 36.9M 104.7G 640 161 51.4% /51.2% 69.7% 55.9%\nYOLOv5-N (r6.1) [23] 1.9M 4.5G 640 159 - / 28.0% - -\nYOLOv5-S (r6.1) [23] 7.2M 16.5G 640 156 - / 37.4% - -\nPPYOLOE-M [85] 23.4M 49.9G 640 123 48.9% / 48.6% 66.5% 53.0%\nYOLOv5-N6 (r6.1) [23] 3.2M 18.4G 1280 123 - / 36.0% - -\nYOLOv5-S6 (r6.1) [23] 12.6M 67.2G 1280 122 - / 44.8% - -\nYOLOv5-M (r6.1) [23] 21.2M 49.0G 640 122 - / 45.4% - -\nYOLOv7-X 71.3M 189.9G 640 114 53.1% /52.9% 71.2% 57.8%\nYOLOR-CSP [81] 52.9M 120.4G 640 106 51.1% / 50.8% 69.6% 55.7%\nYOLOX-S [21] 9.0M 26.8G 640 102 40.5% / 40.5% - -\nYOLOv5-L (r6.1) [23] 46.5M 109.1G 640 99 - / 49.0% - -\nYOLOv5-M6 (r6.1) [23] 35.7M 200.0G 1280 90 - / 51.3% - -\nYOLOR-CSP-X [81] 96.9M 226.8G 640 87 53.0% / 52.7% 71.4% 57.9%\nYOLOv7-W6 70.4M 360.0G 1280 84 54.9% /54.6% 72.6% 60.1%\nYOLOv5-X (r6.1) [23] 86.7M 205.7G 640 83 - / 50.7% - -\nYOLOX-M [21] 25.3M 73.8G 640 81 47.2% / 46.9% - -\nPPYOLOE-L [85] 52.2M 110.1G 640 78 51.4% / 50.9% 68.9% 55.6%\nYOLOR-P6 [81] 37.2M 325.6G 1280 76 53.9% / 53.5% 71.4% 58.9%\nYOLOX-L [21] 54.2M 155.6G 640 69 50.1% / 49.7% - -\nYOLOR-W6 [81] 79.8G 453.2G 1280 66 55.2% /54.8% 72.7% 60.5%\nYOLOv5-L6 (r6.1) [23] 76.8M 445.6G 1280 63 - / 53.7% - -\nYOLOX-X [21] 99.1M 281.9G 640 58 51.5% / 51.1% - -\nYOLOv7-E6 97.2M 515.2G 1280 56 56.0% /55.9% 73.5% 61.2%\nYOLOR-E6 [81] 115.8M 683.2G 1280 45 55.8% / 55.7% 73.4% 61.1%\nPPYOLOE-X [85] 98.4M 206.6G 640 45 52.2% / 51.9% 69.9% 56.5%\nYOLOv7-D6 154.7M 806.8G 1280 44 56.6% /56.3% 74.0% 61.8%\nYOLOv5-X6 (r6.1) [23] 140.7M 839.2G 1280 38 - / 55.0% - -\nYOLOv7-E6E 151.7M 843.2G 1280 36 56.8% /56.8% 74.4% 62.1%\nYOLOR-D6 [81] 151.7M 935.6G 1280 34 56.5% / 56.1% 74.1% 61.9%\nF-RCNN-R101-FPN+ [5] 60.0M 246.0G 1333 20 - / 44.0% - -\nDeformable DETR [100] 40.0M 173.0G - 19 - / 46.2% - -\nSwin-B (C-M-RCNN) [52] 145.0M 982.0G 1333 11.6 - / 51.9% - -\nDETR DC5-R101 [5] 60.0M 253.0G 1333 10 - / 44.9% - -\nEfﬁcientDet-D7x [74] 77.0M 410.0G 1536 6.5 55.1% / 54.4% 72.4% 58.4%\nDual-Swin-T (C-M-RCNN) [47] 113.8M 836.0G 1333 6.5 - / 53.6% - -\nViT-Adapter-B [7] 122.0M 997.0G - 4.4 - / 50.8% - -\nDual-Swin-B (HTC) [47] 235.0M - 1600 2.5 58.7% /58.4% - -\nDual-Swin-L (HTC) [47] 453.0M - 1600 1.5 59.4% /59.1% - -\nModel #Param. FLOPs Size FPSA100APtest/APvalAPtest\n50APtest\n75\nDN-Deformable-DETR [41] 48.0M 265.0G 1333 23.0 - / 48.6% - -\nConvNeXt-B (C-M-RCNN) [53] - 964.0G 1280 11.5 - / 54.0% 73.1% 58.8%\nSwin-B (C-M-RCNN) [52] - 982.0G 1280 10.7 - / 53.0% 71.8% 57.5%\nDINO-5scale (R50) [89] 47.0M 860.0G 1333 10.0 - / 51.0% - -\nConvNeXt-L (C-M-RCNN) [53] - 1354.0G 1280 10.0 - / 54.8% 73.8% 59.8%\nSwin-L (C-M-RCNN) [52] - 1382.0G 1280 9.2 - / 53.9% 72.4% 58.8%\nConvNeXt-XL (C-M-RCNN) [53] - 1898.0G 1280 8.6 - / 55.2% 74.2% 59.9%\n8. More comparison\nYOLOv7 surpasses all known object detectors in both\nspeed and accuracy in the range from 5 FPS to 160 FPS and\nhas the highest accuracy 56.8% AP test-dev / 56.8% AP\nmin-val among all known real-time object detectors with 30\nFPS or higher on GPU V100. YOLOv7-E6 object detector\n(56 FPS V100, 55.9% AP) outperforms both transformer-\nbased detector SWIN-L Cascade-Mask R-CNN (9.2 FPS\nA100, 53.9% AP) by 509% in speed and 2% in accuracy,and convolutional-based detector ConvNeXt-XL Cascade-\nMask R-CNN (8.6 FPS A100, 55.2% AP) by 551% in speed\nand 0.7% AP in accuracy, as well as YOLOv7 outperforms:\nYOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, De-\nformable DETR, DINO-5scale-R50, ViT-Adapter-B and\nmany other object detectors in speed and accuracy. More\nover, we train YOLOv7 only on MS COCO dataset from\nscratch without using any other datasets or pre-trained\nweights.\n10', metadata={'source': 'papers/yolov7paper.pdf', 'page': 9}),

Document(page\_content='Figure 9: Comparison with other object detectors.\nFigure 10: Comparison with other real-time object detectors.\nTable 10: Comparison of different setting.\nModel Presicion IoU threshold APval\nYOLOv7-X FP16 (default) 0.65 (default) 52.9%\nYOLOv7-X FP32 0.65 53.0%\nYOLOv7-X FP16 0.70 53.0%\nYOLOv7-X FP32 0.70 53.1%\nimprovement - - +0.2%\n\*Similar to meituan/YOLOv6 and PPYOLOE, our model could\nget higher AP when set higher IoU threshold.\nThe maximum accuracy of the YOLOv7-E6E (56.8%\nAP) real-time model is +13.7% AP higher than the cur-\nrent most accurate meituan/YOLOv6-s model (43.1% AP)\non COCO dataset. Our YOLOv7-tiny (35.2% AP, 0.4\nms) model is +25% faster and +0.2% AP higher than\nmeituan/YOLOv6-n (35.0% AP, 0.5 ms) under identical\nconditions on COCO dataset and V100 GPU with batch=32.\nFigure 11: Comparison with other real-time object detectors.\n11', metadata={'source': 'papers/yolov7paper.pdf', 'page': 10}),

Document(page\_content='References\n[1] anonymous. Designing network design strategies. anony-\nmous submission , 2022. 3\n[2] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus\nCubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens,\nand Barret Zoph. Revisiting ResNets: Improved training\nand scaling strategies. Advances in Neural Information Pro-\ncessing Systems (NeurIPS) , 34, 2021. 2\n[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-\nYuan Mark Liao. YOLOv4: Optimal speed and accuracy of\nobject detection. arXiv preprint arXiv:2004.10934 , 2020.\n2, 6, 7\n[4] Yue Cao, Thomas Andrew Geddes, Jean Yee Hwa Yang,\nand Pengyi Yang. Ensemble deep learning in bioinformat-\nics.Nature Machine Intelligence , 2(9):500–508, 2020. 2\n[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nico-\nlas Usunier, Alexander Kirillov, and Sergey Zagoruyko.\nEnd-to-end object detection with transformers. In Pro-\nceedings of the European Conference on Computer Vision\n(ECCV) , pages 213–229, 2020. 10\n[6] Kean Chen, Weiyao Lin, Jianguo Li, John See, Ji Wang, and\nJunni Zou. AP-loss for accurate one-stage object detection.\nIEEE Transactions on Pattern Analysis and Machine Intel-\nligence (TPAMI) , 43(11):3782–3798, 2020. 2\n[7] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong\nLu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for\ndense predictions. arXiv preprint arXiv:2205.08534 , 2022.\n10\n[8] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae\nLee. Gaussian YOLOv3: An accurate and fast object detec-\ntor using localization uncertainty for autonomous driving.\nInProceedings of the IEEE/CVF International Conference\non Computer Vision (ICCV) , pages 502–511, 2019. 5\n[9] Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen,\nMengchen Liu, Lu Yuan, and Lei Zhang. Dynamic head:\nUnifying object detection heads with attentions. In Pro-\nceedings of the IEEE/CVF Conference on Computer Vision\nand Pattern Recognition (CVPR) , pages 7373–7382, 2021.\n2\n[10] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi\nHuang, Jungong Han, and Guiguang Ding. Re-\nparameterizing your optimizers rather than architectures.\narXiv preprint arXiv:2205.15242 , 2022. 2\n[11] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong\nHan. ACNet: Strengthening the kernel skeletons for pow-\nerful CNN via asymmetric convolution blocks. In Proceed-\nings of the IEEE/CVF International Conference on Com-\nputer Vision (ICCV) , pages 1911–1920, 2019. 2\n[12] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and\nGuiguang Ding. Diverse branch block: Building a con-\nvolution as an inception-like unit. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 10886–10895, 2021. 2\n[13] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong\nHan, Guiguang Ding, and Jian Sun. RepVGG: Making\nVGG-style convnets great again. In Proceedings of theIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 13733–13742, 2021. 2, 4\n[14] Xiaohan Ding, Xiangyu Zhang, Yizhuang Zhou, Jungong\nHan, Guiguang Ding, and Jian Sun. Scaling up your ker-\nnels to 31x31: Revisiting large kernel design in CNNs. In\nProceedings of the IEEE/CVF Conference on Computer Vi-\nsion and Pattern Recognition (CVPR) , 2022. 2\n[15] Piotr Doll ´ar, Mannat Singh, and Ross Girshick. Fast and\naccurate model scaling. In Proceedings of the IEEE/CVF\nConference on Computer Vision and Pattern Recognition\n(CVPR) , pages 924–932, 2021. 2, 3\n[16] Xianzhi Du, Barret Zoph, Wei-Chih Hung, and Tsung-Yi\nLin. Simple training strategies and model scaling for object\ndetection. arXiv preprint arXiv:2107.00057 , 2021. 2\n[17] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott,\nand Weilin Huang. TOOD: Task-aligned one-stage object\ndetection. In Proceedings of the IEEE/CVF International\nConference on Computer Vision (ICCV) , pages 3490–3499,\n2021. 2, 5\n[18] Di Feng, Christian Haase-Sch ¨utz, Lars Rosenbaum, Heinz\nHertlein, Claudius Glaeser, Fabian Timm, Werner Wies-\nbeck, and Klaus Dietmayer. Deep multi-modal object de-\ntection and semantic segmentation for autonomous driv-\ning: Datasets, methods, and challenges. IEEE Transac-\ntions on Intelligent Transportation Systems , 22(3):1341–\n1360, 2020. 1\n[19] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin,\nDmitry P Vetrov, and Andrew G Wilson. Loss sur-\nfaces, mode connectivity, and fast ensembling of DNNs.\nAdvances in Neural Information Processing Systems\n(NeurIPS) , 31, 2018. 2\n[20] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and\nJian Sun. OTA: Optimal transport assignment for object\ndetection. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n303–312, 2021. 2, 5\n[21] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian\nSun. YOLOX: Exceeding YOLO series in 2021. arXiv\npreprint arXiv:2107.08430 , 2021. 1, 2, 7, 10\n[22] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. NAS-FPN:\nLearning scalable feature pyramid architecture for object\ndetection. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n7036–7045, 2019. 2\n[23] Jocher Glenn. YOLOv5 release v6.1. [https://github.com/\nultralytics/yolov5/releases/tag/v6.1](https://github.com/nultralytics/yolov5/releases/tag/v6.1), 2022. 2, 7, 10\n[24] Shuxuan Guo, Jose M Alvarez, and Mathieu Salzmann. Ex-\npandNets: Linear over-parameterization to train compact\nconvolutional networks. Advances in Neural Information\nProcessing Systems (NeurIPS) , 33:1298–1310, 2020. 2\n[25] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing\nXu, and Chang Xu. GhostNet: More features from cheap\noperations. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n1580–1589, 2020. 1\n[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.\nDeep residual learning for image recognition. In Proceed-\n12', metadata={'source': 'papers/yolov7paper.pdf', 'page': 11}),

Document(page\_content='ings of the IEEE/CVF Conference on Computer Vision and\nPattern Recognition (CVPR) , pages 770–778, 2016. 1, 4, 5\n[27] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh\nChen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu,\nRuoming Pang, Vijay Vasudevan, et al. Searching for Mo-\nbileNetV3. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n1314–1324, 2019. 1\n[28] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry\nKalenichenko, Weijun Wang, Tobias Weyand, Marco An-\ndreetto, and Hartwig Adam. MobileNets: Efﬁcient con-\nvolutional neural networks for mobile vision applications.\narXiv preprint arXiv:1704.04861 , 2017. 1\n[29] Mu Hu, Junyi Feng, Jiashen Hua, Baisheng Lai, Jian-\nqiang Huang, Xiaojin Gong, and Xiansheng Hua. On-\nline convolutional re-parameterization. In Proceedings of\nthe IEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , 2022. 2\n[30] Miao Hu, Yali Li, Lu Fang, and Shengjin Wang. A2-FPN:\nAttention aggregation based feature pyramid network for\ninstance segmentation. In Proceedings of the IEEE/CVF\nConference on Computer Vision and Pattern Recognition\n(CVPR) , pages 15343–15352, 2021. 2\n[31] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E\nHopcroft, and Kilian Q Weinberger. Snapshot ensembles:\nTrain 1, get m for free. International Conference on Learn-\ning Representations (ICLR) , 2017. 2\n[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kil-\nian Q Weinberger. Densely connected convolutional net-\nworks. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n4700–4708, 2017. 2, 4, 5\n[33] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov,\nDmitry Vetrov, and Andrew Gordon Wilson. Averaging\nweights leads to wider optima and better generalization. In\nConference on Uncertainty in Artiﬁcial Intelligence (UAI) ,\n2018. 2\n[34] Paul F Jaeger, Simon AA Kohl, Sebastian Bickel-\nhaupt, Fabian Isensee, Tristan Anselm Kuder, Heinz-Peter\nSchlemmer, and Klaus H Maier-Hein. Retina U-Net: Em-\nbarrassingly simple exploitation of segmentation supervi-\nsion for medical object detection. In Machine Learning for\nHealth Workshop , pages 171–183, 2020. 1\n[35] Hakan Karaoguz and Patric Jensfelt. Object detection ap-\nproach for robot grasp detection. In IEEE International\nConference on Robotics and Automation (ICRA) , pages\n4953–4959, 2019. 1\n[36] Kang Kim and Hee Seok Lee. Probabilistic anchor as-\nsignment with iou prediction for object detection. In Pro-\nceedings of the European conference on computer vision\n(ECCV) , pages 355–371, 2020. 5\n[37] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr\nDoll´ar. Panoptic feature pyramid networks. In Proceed-\nings of the IEEE/CVF Conference on Computer Vision and\nPattern Recognition (CVPR) , pages 6399–6408, 2019. 2\n[38] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou\nZhang, and Zhuowen Tu. Deeply-supervised nets. In Arti-\nﬁcial Intelligence and Statistics , pages 562–570, 2015. 5[39] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok\nBae, and Jongyoul Park. An energy and GPU-computation\nefﬁcient backbone network for real-time object detection.\nInProceedings of the IEEE/CVF Conference on Com-\nputer Vision and Pattern Recognition Workshops (CVPRW) ,\npages 0–0, 2019. 2, 3\n[40] Buyu Li, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and\nXiaogang Wang. GS3D: An efﬁcient 3d object detection\nframework for autonomous driving. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 1019–1028, 2019. 1\n[41] Feng Li, Hao Zhang, Shilong Liu, Jian Guo, Lionel M\nNi, and Lei Zhang. DN-DETR: Accelerate detr training\nby introducing query denoising. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 13619–13627, 2022. 10\n[42] Shuai Li, Chenhang He, Ruihuang Li, and Lei Zhang. A\ndual weighting label assignment scheme for object detec-\ntion. In Proceedings of the IEEE/CVF Conference on Com-\nputer Vision and Pattern Recognition (CVPR) , pages 9387–\n9396, 2022. 2, 5\n[43] Xiang Li, Wenhai Wang, Xiaolin Hu, Jun Li, Jinhui Tang,\nand Jian Yang. Generalized focal loss v2: Learning reliable\nlocalization quality estimation for dense object detection. In\nProceedings of the IEEE/CVF Conference on Computer Vi-\nsion and Pattern Recognition (CVPR) , pages 11632–11641,\n2021. 5\n[44] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin\nHu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal\nloss: Learning qualiﬁed and distributed bounding boxes for\ndense object detection. Advances in Neural Information\nProcessing Systems (NeurIPS) , 33:21002–21012, 2020. 5\n[45] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He.\nExploring plain vision transformer backbones for object de-\ntection. arXiv preprint arXiv:2203.16527 , 2022. 2\n[46] Zhuoling Li, Minghui Dong, Shiping Wen, Xiang Hu, Pan\nZhou, and Zhigang Zeng. CLU-CNNs: Object detection for\nmedical images. Neurocomputing , 350:53–59, 2019. 1\n[47] Tingting Liang, Xiaojie Chu, Yudong Liu, Yongtao Wang,\nZhi Tang, Wei Chu, Jingdong Chen, and Haibin Ling. CB-\nNetV2: A composite backbone network architecture for ob-\nject detection. arXiv preprint arXiv:2107.00420 , 2021. 5,\n10\n[48] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song\nHan. Memory-efﬁcient patch-based inference for tiny deep\nlearning. Advances in Neural Information Processing Sys-\ntems (NeurIPS) , 34:2346–2358, 2021. 1\n[49] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song\nHan, et al. MCUNet: Tiny deep learning on IoT de-\nvices. Advances in Neural Information Processing Systems\n(NeurIPS) , 33:11711–11722, 2020. 1\n[50] Yuxuan Liu, Lujia Wang, and Ming Liu. YOLOStereo3D:\nA step back to 2D for efﬁcient stereo 3D detection. In\nIEEE International Conference on Robotics and Automa-\ntion (ICRA) , pages 13018–13024, 2021. 5\n[51] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie,\nYixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong,\n13', metadata={'source': 'papers/yolov7paper.pdf', 'page': 12}),

Document(page\_content='et al. Swin transformer v2: Scaling up capacity and res-\nolution. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , 2022. 2\n[52] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng\nZhang, Stephen Lin, and Baining Guo. Swin transformer:\nHierarchical vision transformer using shifted windows. In\nProceedings of the IEEE/CVF International Conference on\nComputer Vision (ICCV) , pages 10012–10022, 2021. 10\n[53] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feicht-\nenhofer, Trevor Darrell, and Saining Xie. A ConvNet for\nthe 2020s. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n11976–11986, 2022. 10\n[54] Rangi Lyu. NanoDet-Plus. [https://github.com/RangiLyu/\nnanodet/releases/tag/v1.0.0-alpha-1](https://github.com/RangiLyu/nnanodet/releases/tag/v1.0.0-alpha-1), 2021. 1, 2\n[55] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian\nSun. ShufﬂeNet V2: Practical guidelines for efﬁcient CNN\narchitecture design. In Proceedings of the European Con-\nference on Computer Vision (ECCV) , pages 116–131, 2018.\n1, 3\n[56] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan\nKalkan. A ranking-based, balanced loss function unifying\nclassiﬁcation and localisation in object detection. Advances\nin Neural Information Processing Systems (NeurIPS) ,\n33:15534–15545, 2020. 2\n[57] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan\nKalkan. Rank & sort loss for object detection and in-\nstance segmentation. In Proceedings of the IEEE/CVF In-\nternational Conference on Computer Vision (ICCV) , pages\n3009–3018, 2021. 2\n[58] Shuvo Kumar Paul, Muhammed Tawﬁq Chowdhury,\nMircea Nicolescu, Monica Nicolescu, and David Feil-\nSeifer. Object detection and pose estimation from rgb and\ndepth data for real-time, adaptive robotic grasping. In Ad-\nvances in Computer Vision and Computational Biology ,\npages 121–142. 2021. 1\n[59] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. De-\ntectoRS: Detecting objects with recursive feature pyramid\nand switchable atrous convolution. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 10213–10224, 2021. 2\n[60] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick,\nKaiming He, and Piotr Doll ´ar. Designing network design\nspaces. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n10428–10436, 2020. 2\n[61] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali\nFarhadi. You only look once: Uniﬁed, real-time object de-\ntection. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n779–788, 2016. 2, 5\n[62] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster,\nstronger. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n7263–7271, 2017. 2\n[63] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental\nimprovement. arXiv preprint arXiv:1804.02767 , 2018. 1, 2[64] Hamid Rezatoﬁghi, Nathan Tsoi, JunYoung Gwak, Amir\nSadeghian, Ian Reid, and Silvio Savarese. Generalized in-\ntersection over union: A metric and a loss for bounding\nbox regression. In Proceedings of the IEEE/CVF Confer-\nence on Computer Vision and Pattern Recognition (CVPR) ,\npages 658–666, 2019. 2\n[65] Byungseok Roh, JaeWoong Shin, Wuhyun Shin, and\nSaehoon Kim. Sparse DETR: Efﬁcient end-to-end ob-\nject detection with learnable sparsity. arXiv preprint\narXiv:2111.14330 , 2021. 5\n[66] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey\nZhmoginov, and Liang-Chieh Chen. MobileNetV2: In-\nverted residuals and linear bottlenecks. In Proceedings of\nthe IEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 4510–4520, 2018. 1\n[67] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang,\nYurong Chen, and Xiangyang Xue. Object detection\nfrom scratch with deep supervision. IEEE Transactions\non Pattern Analysis and Machine Intelligence (TPAMI) ,\n42(2):398–412, 2019. 5\n[68] Karen Simonyan and Andrew Zisserman. Very deep convo-\nlutional networks for large-scale image recognition. arXiv\npreprint arXiv:1409.1556 , 2014. 4\n[69] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng\nXu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan,\nChanghu Wang, et al. Sparse R-CNN: End-to-end ob-\nject detection with learnable proposals. In Proceedings of\nthe IEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 14454–14463, 2021. 2\n[70] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet,\nScott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent\nVanhoucke, and Andrew Rabinovich. Going deeper with\nconvolutions. In Proceedings of the IEEE/CVF Confer-\nence on Computer Vision and Pattern Recognition (CVPR) ,\npages 1–9, 2015. 5\n[71] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon\nShlens, and Zbigniew Wojna. Rethinking the inception\narchitecture for computer vision. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 2818–2826, 2016. 2\n[72] Mingxing Tan and Quoc Le. EfﬁcientNet: Rethinking\nmodel scaling for convolutional neural networks. In Inter-\nnational Conference on Machine Learning (ICML) , pages\n6105–6114, 2019. 2, 3\n[73] Mingxing Tan and Quoc Le. EfﬁcientNetv2: Smaller mod-\nels and faster training. In International Conference on Ma-\nchine Learning (ICML) , pages 10096–10106, 2021. 2\n[74] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efﬁcient-\nDet: Scalable and efﬁcient object detection. In Proceedings\nof the IEEE/CVF Conference on Computer Vision and Pat-\ntern Recognition (CVPR) , pages 10781–10790, 2020. 2, 10\n[75] Antti Tarvainen and Harri Valpola. Mean teachers are better\nrole models: Weight-averaged consistency targets improve\nsemi-supervised deep learning results. Advances in Neural\nInformation Processing Systems (NeurIPS) , 30, 2017. 2, 6\n[76] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS:\nFully convolutional one-stage object detection. In Proceed-\n14', metadata={'source': 'papers/yolov7paper.pdf', 'page': 13}),

Document(page\_content='ings of the IEEE/CVF International Conference on Com-\nputer Vision (ICCV) , pages 9627–9636, 2019. 2\n[77] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS:\nA simple and strong anchor-free object detector. IEEE\nTransactions on Pattern Analysis and Machine Intelligence\n(TPAMI) , 44(4):1922–1933, 2022. 2\n[78] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff\nZhu, Oncel Tuzel, and Anurag Ranjan. An im-\nproved one millisecond mobile backbone. arXiv preprint\narXiv:2206.04040 , 2022. 2\n[79] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-\nYuan Mark Liao. Scaled-YOLOv4: Scaling cross stage\npartial network. In Proceedings of the IEEE/CVF Confer-\nence on Computer Vision and Pattern Recognition (CVPR) ,\npages 13029–13038, 2021. 2, 3, 6, 7\n[80] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu,\nPing-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSP-\nNet: A new backbone that can enhance learning capabil-\nity of CNN. In Proceedings of the IEEE/CVF Conference\non Computer Vision and Pattern Recognition Workshops\n(CVPRW) , pages 390–391, 2020. 1\n[81] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao.\nYou only learn one representation: Uniﬁed network for\nmultiple tasks. arXiv preprint arXiv:2105.04206 , 2021. 1,\n2, 6, 7, 10\n[82] Jianfeng Wang, Lin Song, Zeming Li, Hongbin Sun, Jian\nSun, and Nanning Zheng. End-to-end object detection\nwith fully convolutional network. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 15849–15858, 2021. 2, 5\n[83] Bichen Wu, Chaojian Li, Hang Zhang, Xiaoliang Dai,\nPeizhao Zhang, Matthew Yu, Jialiang Wang, Yingyan Lin,\nand Peter Vajda. FBNetv5: Neural architecture search for\nmultiple tasks in one run. arXiv preprint arXiv:2111.10007 ,\n2021. 1\n[84] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin,\nGabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans,\nMingxing Tan, Vikas Singh, and Bo Chen. MobileDets:\nSearching for object detection architectures for mobile ac-\ncelerators. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n3825–3834, 2021. 1\n[85] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao\nChang, Cheng Cui, Kaipeng Deng, Guanzhong Wang,\nQingqing Dang, Shengyu Wei, Yuning Du, et al. PP-\nYOLOE: An evolved version of YOLO. arXiv preprint\narXiv:2203.16250 , 2022. 2, 7, 8, 10\n[86] Zetong Yang, Yin Zhou, Zhifeng Chen, and Jiquan Ngiam.\n3D-MAN: 3D multi-frame attention network for object de-\ntection. In Proceedings of the IEEE/CVF Conference on\nComputer Vision and Pattern Recognition (CVPR) , pages\n1863–1872, 2021. 5\n[87] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor\nDarrell. Deep layer aggregation. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 2403–2412, 2018. 1\n[88] Guanghua Yu, Qinyao Chang, Wenyu Lv, Chang Xu, Cheng\nCui, Wei Ji, Qingqing Dang, Kaipeng Deng, GuanzhongWang, Yuning Du, et al. PP-PicoDet: A better real-\ntime object detector on mobile devices. arXiv preprint\narXiv:2111.00902 , 2021. 1\n[89] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun\nZhu, Lionel M Ni, and Heung-Yeung Shum. DINO: DETR\nwith improved denoising anchor boxes for end-to-end ob-\nject detection. arXiv preprint arXiv:2203.03605 , 2022. 10\n[90] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sun-\nderhauf. VarifocalNet: An IoU-aware dense object detector.\nInProceedings of the IEEE/CVF Conference on Computer\nVision and Pattern Recognition (CVPR) , pages 8514–8523,\n2021. 5\n[91] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and\nStan Z Li. Bridging the gap between anchor-based and\nanchor-free detection via adaptive training sample selec-\ntion. In Proceedings of the IEEE/CVF Conference on Com-\nputer Vision and Pattern Recognition (CVPR) , pages 9759–\n9768, 2020. 5\n[92] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian\nSun. ShufﬂeNet: An extremely efﬁcient convolutional neu-\nral network for mobile devices. In Proceedings of the\nIEEE/CVF Conference on Computer Vision and Pattern\nRecognition (CVPR) , pages 6848–6856, 2018. 1\n[93] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan\nYuan, Ping Luo, Wenyu Liu, and Xinggang Wang. BYTE-\nTrack: Multi-object tracking by associating every detection\nbox. arXiv preprint arXiv:2110.06864 , 2021. 1\n[94] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng,\nand Wenyu Liu. FAIRMOT: On the fairness of detec-\ntion and re-identiﬁcation in multiple object tracking. Inter-\nnational Journal of Computer Vision , 129(11):3069–3087,\n2021. 1\n[95] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang\nYe, and Dongwei Ren. Distance-IoU loss: Faster and bet-\nter learning for bounding box regression. In Proceedings\nof the AAAI Conference on Artiﬁcial Intelligence (AAAI) ,\nvolume 34, pages 12993–13000, 2020. 2\n[96] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo\nYin, Yuchao Dai, and Ruigang Yang. IoU loss for 2D/3D\nobject detection. In International Conference on 3D Vision\n(3DV) , pages 85–94, 2019. 2\n[97] Xingyi Zhou, Dequan Wang, and Philipp Kr ¨ahenb ¨uhl. Ob-\njects as points. arXiv preprint arXiv:1904.07850 , 2019. 1,\n2\n[98] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima\nTajbakhsh, and Jianming Liang. UNet++: A nested U-\nNet architecture for medical image segmentation. In\nDeep Learning in Medical Image Analysis and Multimodal\nLearning for Clinical Decision Support , 2018. 5\n[99] Benjin Zhu, Jianfeng Wang, Zhengkai Jiang, Fuhang Zong,\nSongtao Liu, Zeming Li, and Jian Sun. AutoAssign: Differ-\nentiable label assignment for dense object detection. arXiv\npreprint arXiv:2007.03496 , 2020. 2, 5\n[100] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang,\nand Jifeng Dai. Deformable DETR: Deformable trans-\nformers for end-to-end object detection. In Proceedings of\nthe International Conference on Learning Representations\n(ICLR) , 2021. 10\n15', metadata={'source': 'papers/yolov7paper.pdf', 'page': 14}),

Document(page\_content='DeepSeek-Coder: When the Large Language Model Meets\nProgramming - The Rise of Code Intelligence\nDaya Guo\*1, Qihao Zhu∗1,2, Dejian Yang1, Zhenda Xie1, Kai Dong1, Wentao Zhang1\nGuanting Chen1, Xiao Bi1, Y. Wu1, Y.K. Li1, Fuli Luo1, Yingfei Xiong2, Wenfeng Liang1\n1DeepSeek-AI\n2Key Lab of HCST (PKU), MOE; SCS, Peking University\n{zhuqh, [guodaya}@deepseek.com](mailto:guodaya%7D@deepseek.com)\nhttps://github.com/deepseek-ai/DeepSeek-Coder\nAbstract\nThe rapid development of large language models has revolutionized code intelligence in\nsoftware development. However, the predominance of closed-source models has restricted\nextensive research and development. To address this, we introduce the DeepSeek-Coder series,\na range of open-source code models with sizes from 1.3B to 33B, trained from scratch on 2\ntrillion tokens. These models are pre-trained on a high-quality project-level code corpus and\nemploy a fill-in-the-blank task with a 16K window to enhance code generation and infilling.\nOur extensive evaluations demonstrate that DeepSeek-Coder not only achieves state-of-the-art\nperformance among open-source code models across multiple benchmarks but also surpasses\nexisting closed-source models like Codex and GPT-3.5. Furthermore, DeepSeek-Coder models\nare under a permissive license that allows for both research and unrestricted commercial use.\nFigure 1|The Performance of DeepSeek-Coder\n\*Core contributors, ordered alphabetically by the name.arXiv:2401.14196v2 [cs.SE] 26 Jan 2024', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 0}),

Document(page\_content='1. Introduction\nThe field of software development has been significantly transformed by the swift advancement\nof large language models (OpenAI, 2023; Touvron et al., 2023), which have brought about\na new era of code intelligence. These models have the potential to automate and streamline\nmany aspects of coding, from bug detection to code generation, thereby enhancing productivity\nand reducing the likelihood of human error. However, a major challenge in this field is the\nperformance gap between open-source models (Li et al., 2023; Nijkamp et al., 2022; Roziere\net al., 2023; Wang et al., 2021) and closed-source models (Gemini Team, 2023; OpenAI, 2023).\nThe giant closed-source models, while powerful, are often inaccessible to many researchers and\ndevelopers due to their proprietary nature.\nIn response to this challenge, we present the DeepSeek-Coder series. This series comprises\na range of open-source code models, varying in size from 1.3B to 33B, including the base\nversion and instructed version for each size. Each model in the series has been trained from\nscratch on 2 trillion tokens sourced from 87 programming languages, ensuring a comprehensive\nunderstanding of coding languages and syntax. Besides, we attempt to organize the pre-\ntraining data at the repository level to enhance the pre-trained model’s understanding capability\nwithin the context of cross-files within a repository. In addition to employing the next token\nprediction loss during pre-training, we have also incorporated the Fill-In-Middle (FIM) approach\n(Bavarian et al., 2022; Li et al., 2023). This approach is designed to further bolster the model’s\ncode completion capabilities. To meet the requirements of handling longer code inputs, we\nhave extended the context length to 16K. This adjustment allows our models to handle more\ncomplex and extensive coding tasks, thereby increasing their versatility and applicability in\nvarious coding scenarios.\nWe have carried out comprehensive experiments using a variety of public code-related\nbenchmarks. The findings reveal that among open-source models, DeepSeek-Coder-Base 33B\nconsistently delivers superior performance across all benchmarks. Furthermore, DeepSeek-\nCoder-Instruct 33B surpasses OpenAI GPT-3.5 Turbo in the majority of the evaluation benchmarks,\nsignificantly narrowing the performance gap between OpenAI GPT-4 and open-source models.\nRemarkably, despite having fewer parameters, DeepSeek-Coder-Base 7B demonstrates competi-\ntive performance when compared to models that are five times larger, such as CodeLlama-33B\n(Roziere et al., 2023). To summarize, our main contributions are:\n•We introduce DeepSeek-Coder-Base and DeepSeek-Coder-Instruct, our advanced code-\nfocused large language models (LLMs). Developed through extensive training on an\nexpansive code corpus, these models exhibit proficiency in understanding 87 programming\nlanguages. Additionally, they are available in various model scales to cater to a wide range\nof computational and application needs.\n•We make the first attempt to incorporate repository-level data construction during the\npre-training phase of our models. We find that it can significantly boost the capability of\ncross-file code generation.\n•Our analysis rigorously examines the impact of FIM training strategies on the pretraining\nphase of code models. The outcomes of these comprehensive studies shed light on intriguing\naspects of FIM configurations, offering valuable insights that significantly contribute to the\nenhancement and development of code pretrained models.\n•We conduct extensive evaluations of our code LLMs against a wide array of benchmarks en-\ncompassing numerous code-related tasks. The findings demonstrate that DeepSeek-Coder-\nBase surpasses all existing open-source code LLMs across these benchmarks. Furthermore,\n2', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 1}),

Document(page\_content='with meticulous fine-tuning using instructional data, DeepSeek-Coder-Instruct achieves\nbetter performance compared to the OpenAI GPT-3.5 Turbo model in code-related tasks.\n2. Data Collection\nThe training dataset of DeepSeek-Coder is composed of 87% source code, 10% English code-\nrelated natural language corpus, and 3% code-unrelated Chinese natural language corpus. The\nEnglish corpus consists of materials from GitHub’s Markdown and StackExchange1, which are\nused to enhance the model’s understanding of code-related concepts and improve its ability\nto handle tasks like library usage and bug fixing. Meanwhile, the Chinese corpus consists of\nhigh-quality articles aimed at improving the model’s proficiency in understanding the Chinese\nlanguage. In this section, we will provide an overview of how we construct the code training\ndata. This process involves data crawling, rule-based filtering, dependency parsing, repository-\nlevel deduplication, and quality screening, as illustrated in Figure 2. In the following, we will\ndescribe the data creation procedure step by step.\nRule\tFilteringDependency\tParsingRepo-level\tDeduplication\nQuality\tScreeningData\tCrawling\nFigure 2|The Procedure of Dataset Creation\n2.1. GitHub Data Crawling and Filtering\nWe collect public repositories created before February 2023 on GitHub and retain only 87\nprogramming languages, as listed in Table 1. To reduce the amount of data to be processed, we\napply filtering rules similar to those used in the StarCoder project (Li et al., 2023) to preliminarily\nfilter out lower-quality code. By applying these filtering rules, we reduce the total amount of\ndata to only 32.8% of its original size. To make the paper self-contained, we briefly describe the\nfilter rules used in the StarCoder Data project:\nFirstly, we filter out files with an average line length exceeding 100 characters or a maximum\nline length surpassing 1000 characters. Additionally, we remove files with fewer than 25%\nalphabetic characters. Except for the XSLT programming language, we further filter out files\nwhere the string "<?xml version=" appeared in the first 100 characters. For HTML files, we\nconsider the ratio of visible text to HTML code. We retain files where the visible text constitutes\nat least 20% of the code and is no less than 100 characters. For JSON and YAML files, which\ntypically contain more data, we only keep files that have a character count ranging from 50 to\n5000 characters. This effectively removes most data-heavy files.\n2.2. Dependency Parsing\nIn previous works (Chen et al., 2021; Li et al., 2023; Nijkamp et al., 2022; Roziere et al., 2023),\nlarge language models for code are mainly pre-trained on file-level source code, which ignores\nthe dependencies between different files in a project. However, in practical applications, such\nmodels struggle to effectively scale to handle entire project-level code scenarios. Therefore, we\n1https://stackexchange.com\n3', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 2}),

Document(page\_content='Algorithm 1 Topological Sort for Dependency Analysis\n1:procedure TO P O L O G I C A L SO R T (𝑓𝑖𝑙𝑒𝑠)\n2:𝑔𝑟𝑎𝑝ℎ𝑠←{} ⊲Initialize an empty adjacency list\n3:𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒←{} ⊲Initialize an empty dictionary for in-degrees\n4: for each𝑓𝑖𝑙𝑒in𝑓𝑖𝑙𝑒𝑠do\n5:𝑔𝑟𝑎𝑝ℎ𝑠[𝑓𝑖𝑙𝑒]←[]\n6:𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒[𝑓𝑖𝑙𝑒]← 0\n7: end for\n8:\n9: for each𝑓𝑖𝑙𝑒𝐴 in𝑓𝑖𝑙𝑒𝑠do\n10: for each𝑓𝑖𝑙𝑒𝐵 in𝑓𝑖𝑙𝑒𝑠do\n11: ifHA SDE P E N D E N C Y (𝑓𝑖𝑙𝑒𝐴 ,𝑓𝑖𝑙𝑒𝐵 )then ⊲If fileA depends on fileB\n12: 𝑔𝑟𝑎𝑝ℎ𝑠[𝑓𝑖𝑙𝑒𝐵].append(𝑓𝑖𝑙𝑒𝐴) ⊲Add edge from B to A\n13: 𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒[𝑓𝑖𝑙𝑒𝐴]←𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒[𝑓𝑖𝑙𝑒𝐴]+1 ⊲Increment in-degree of A\n14: end if\n15: end for\n16: end for\n17:\n18:𝑠𝑢𝑏𝑔𝑟𝑎𝑝ℎ𝑠←getDisconnectedSubgraphs (𝑔𝑟𝑎𝑝ℎ𝑠)⊲Identify disconnected subgraphs\n19:𝑎𝑙𝑙𝑅𝑒𝑠𝑢𝑙𝑡𝑠←[]\n20: for each𝑠𝑢𝑏𝑔𝑟𝑎𝑝ℎ in𝑠𝑢𝑏𝑔𝑟𝑎𝑝ℎ𝑠 do\n21: 𝑟𝑒𝑠𝑢𝑙𝑡𝑠←[]\n22: while length(𝑟𝑒𝑠𝑢𝑙𝑡𝑠)≠NumberOfNodes (𝑠𝑢𝑏𝑔𝑟𝑎𝑝ℎ)do\n23: 𝑓𝑖𝑙𝑒←argmin({𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒[𝑓𝑖𝑙𝑒]|𝑓𝑖𝑙𝑒∈𝑠𝑢𝑏𝑔𝑟𝑎𝑝ℎ and𝑓𝑖𝑙𝑒∉𝑟𝑒𝑠𝑢𝑙𝑡𝑠})\n24: for each𝑛𝑜𝑑𝑒 in𝑔𝑟𝑎𝑝ℎ𝑠[𝑓𝑖𝑙𝑒]do\n25: 𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒[𝑛𝑜𝑑𝑒]←𝑖𝑛𝐷𝑒𝑔𝑟𝑒𝑒[𝑛𝑜𝑑𝑒]−1\n26: end for\n27: 𝑟𝑒𝑠𝑢𝑙𝑡𝑠 .append(𝑓𝑖𝑙𝑒)\n28: end while\n29: 𝑎𝑙𝑙𝑅𝑒𝑠𝑢𝑙𝑡𝑠 .append(𝑟𝑒𝑠𝑢𝑙𝑡𝑠)\n30: end for\n31:\n32: return𝑎𝑙𝑙𝑅𝑒𝑠𝑢𝑙𝑡𝑠\n33:end procedure\nwill consider how to leverage the dependencies between files within the same repository in\nthis step. Specifically, we first parse the dependencies between files and then arrange these\nfiles in an order that ensures the context each file relies on is placed before that file in the\ninput sequence. By aligning the files in accordance with their dependencies, our dataset more\naccurately represents real coding practices and structures. This enhanced alignment not only\nmakes our dataset more relevant but also potentially increases the practicality and applicability\nof the model in handling project-level code scenarios. It’s worth noting that we only consider\nthe invocation relationships between files and use regular expressions to extract them, such as\n"import" in Python, "using" in C#, and "include" in C.\nThe algorithm 1 describes a topological sort for dependency analysis on a list of files within\nthe same project. Initially, it sets up two data structures: an empty adjacency list named "graphs"\nto represent dependencies between files and an empty dictionary called "inDegree" for storing\nthe in-degrees of each file. The algorithm then iterates over each file pair to identify depen-\n4', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 3}),

Document(page\_content='dencies, updating "graphs" and"inDegree" accordingly. Next, it identifies any disconnected\nsubgraphs within the overall dependency graph. For each subgraph, the algorithm employs a\nmodified topological sort. Unlike the standard approach that selects nodes with zero in-degrees,\nthis algorithm selects nodes with minimal in-degrees, which allows it to handle cycles within\nthe graph. Selected nodes are added to a "results" list, and the in-degrees of their connected\nnodes are decreased. This process continues until a topologically sorted sequence is generated\nfor each subgraph. The algorithm concludes by returning a list of these sorted sequences, and\neach sequence’s files are concatenated to form a single training sample. To incorporate file path\ninformation, a comment indicating the file’s path is added at the beginning of each file. This\nmethod ensures that the path information is preserved in the training data.\n2.3. Repo-Level Deduplication\nRecent studies have demonstrated the significant performance improvements that can be\nachieved by deduplicating training datasets for Large Language Models (LLMs). Lee et al. (2022)\nhave shown that language model training corpora often contain numerous near-duplicates, and\nthe performance of LLMs can be enhanced by removing long repetitive substrings. Kocetkov\net al. (2022) have applied a near-deduplication method to training data, resulting in dramatic\nimprovements, and they emphasize that near-deduplication is a crucial preprocessing step for\nachieving competitive performance on code benchmark tasks. In our dataset, we have also\nemployed near-deduplication. However, there is a distinction in our approach compared to\nprevious works. We perform deduplication at the repository level of code, rather than at the file\nlevel, as the latter approach may filter out certain files within a repository, potentially disrupting\nthe structure of the repository. Specifically, we treat the concatenated code from the repository\nlevel as a single sample and apply the same near-deduplication algorithm to ensure the integrity\nof the repository structure.\n2.4. Quality Screening and Decontamination\nIn addition to applying the filtering rules mentioned in Section 2.1, we also employ a compiler\nand a quality model, combined with heuristic rules, to further filter out low-quality data. This\nincludes code with syntax errors, poor readability, and low modularity. We provide the statistical\nsummary of source code in Table 1, which includes a total of 87 languages, detailing the disk\nsize, number of files, and percentage for each language. The total data volume is 798 GB with\n603 million files. To ensure that our code training data is not contaminated by information\nfrom the test set, which may be present on GitHub, we’ve implemented an n-gram filtering\nprocess. This process involves the removal of any code segments that match specific criteria.\nSpecifically, we filter out files containing docstrings, questions, and solutions from sources such\nas HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), GSM8K (Cobbe et al., 2021) and\nMATH (Hendrycks et al., 2021). For the filtering criteria, we apply the following rules: if a\npiece of code includes a 10-gram string identical to any in the test data, it is excluded from our\ntraining data. In cases where the test data comprises strings that are shorter than 10-grams but\nno less than 3-grams, we use an exact match approach for filtering.\n5', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 4}),

Document(page\_content='Language Size (GB) Files (k) Prop. (%) Language Size (GB) Files (k) Prop. (%)\nAda 0.91 126 0.11 Literate Haskell 0.16 20 0.02\nAgda 0.26 59 0.03 Lua 0.82 138 0.10\nAlloy 0.07 24 0.01 Makefile 0.92 460 0.12\nANTLR 0.19 38 0.02 Maple 0.03 6 0.00\nAppleScript 0.03 17 0.00 Mathematica 0.82 10 0.10\nAssembly 0.91 794 0.11 MATLAB 0.01 1 0.00\nAugeas 0.00 1 0.00 OCaml 0.91 139 0.11\nAWK 0.09 53 0.01 Pascal 0.79 470 0.10\nBatchfile 0.92 859 0.12 Perl 0.81 148 0.10\nBluespec 0.10 15 0.01 PHP 58.92 40,627 7.38\nC 28.64 27,111 3.59 PowerShell 0.91 236 0.11\nC# 58.56 53,739 7.34 Prolog 0.03 5 0.00\nClojure 0.90 295 0.11 Protocol Buffer 0.92 391 0.12\nCMake 0.90 359 0.11 Python 120.68 75,188 15.12\nCoffeeScript 0.92 361 0.12 R 0.92 158 0.11\nCommon Lisp 0.92 105 0.11 Racket 0.09 13 0.01\nC++ 90.87 36,006 11.39 RMarkdown 6.83 1,606 0.86\nCSS 5.63 11,638 0.71 Ruby 15.01 18,526 1.88\nCUDA 0.91 115 0.11 Rust 0.61 692 0.08\nDart 0.89 264 0.11 SAS 0.92 70 0.11\nDockerfile 0.04 48 0.00 Scala 0.81 971 0.10\nElixir 0.91 549 0.11 Scheme 0.92 216 0.12\nElm 0.92 232 0.12 Shell 13.92 10,890 1.74\nEmacs Lisp 0.91 148 0.11 Smalltalk 0.92 880 0.12\nErlang 0.92 145 0.12 Solidity 0.85 83 0.11\nF# 0.91 340 0.11 Sparql 0.10 88 0.01\nFortran 1.67 654 0.21 SQL 15.14 7,009 1.90\nGLSL 0.92 296 0.11 Stan 0.20 41 0.03\nGo 2.58 1,365 0.32 Standard ML 0.74 117 0.09\nGroovy 0.89 340 0.11 Stata 0.91 122 0.11\nHaskell 0.87 213 0.11 SystemVerilog 0.91 165 0.11\nHTML 30.05 14,998 3.77 TCL 0.90 110 0.11\nIdris 0.11 32 0.01 Tcsh 0.17 53 0.02\nIsabelle 0.74 39 0.09 Tex 20.46 2,867 2.56\nJava 148.66 134,367 18.63 Thrift 0.05 21 0.01\nJava Server Pages 0.86 1072 0.11 TypeScript 60.62 62,432 7.60\nJavaScript 53.84 71,895 6.75 Verilog 0.01 1 0.00\nJSON 4.61 11956 0.58 VHDL 0.85 392 0.11\nJulia 0.92 202 0.12 Visual Basic 0.75 73 0.09\nJupyter Notebook 14.38 2,555 1.80 XSLT 0.36 48 0.04\nKotlin 6.00 3,121 0.75 Yacc 0.72 67 0.09\nLean 0.52 68 0.07 YAML 0.74 890 0.09\nLiterate Agda 0.05 4 0.01 Zig 0.81 70 0.10\nLiterate CoffeeScript 0.01 3 0.00 Total 797.92 603,173 100.00\nTable 1|A summary of the cleaned training data for the selected programming languages.\n3. Training Policy\n3.1. Training Strategy\n3.1.1. Next Token Prediction\nThe first training objective for our model is known as next token prediction . In this process,\nvarious files are concatenated to form a fixed-length entry. Then, these entries are used to train\nthe model, enabling it to predict the subsequent token based on the provided context.\n6', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 5}),

Document(page\_content='3.1.2. Fill-in-the-Middle\nThe second training objective for our model is known as fill-in-the-middle. In the code pre-training\nscenario, it is often necessary to generate corresponding inserted content based on the given\ncontext and subsequent text. Due to specific dependencies in a programming language, relying\nsolely on next token prediction is insufficient to learn this fill-in-the-middle capability. Therefore,\nseveral approaches (Bavarian et al., 2022; Li et al., 2023) propose the pretraining method of\nFill-in-the-Midlle (FIM). This approach involves randomly dividing the text into three parts,\nthen shuffling the order of these parts and connecting them with special characters. This method\naims to incorporate a fill-in-the-blank pretraining task during the training process. Within the\nFIM methodology, two distinct modes are employed: PSM (Prefix-Suffix-Middle) and SPM\n(Suffix-Prefix-Middle). In the PSM mode, the training corpus is organized in the sequence\nof𝑃𝑟𝑒𝑓𝑖𝑥 ,𝑆𝑢𝑓𝑓𝑖𝑥 ,𝑀𝑖𝑑𝑑𝑙𝑒 , aligning the text in a way that the middle segment is flanked by the\nprefix and suffix. Conversely, the SPM mode arranges the segments as 𝑆𝑢𝑓𝑓𝑖𝑥 ,𝑃𝑟𝑒𝑓𝑖𝑥 ,𝑀𝑖𝑑𝑑𝑙𝑒 ,\npresenting a different structural challenge. These modes are instrumental in enhancing the\nmodel’s capability to handle various structural arrangements in code, providing a robust training\nframework for advanced code prediction tasks.\n0 2000 4000 6000 8000\nSteps0.0000.0250.0500.0750.1000.1250.1500.1750.200ValueBenchmark/HumanEval-Pass@1\n0 2000 4000 6000 8000\nSteps0.00.10.20.30.40.50.60.7ValueBenchmark/HumanFIM-Pass@1\n0 2000 4000 6000 8000\nSteps0.000.050.100.150.200.25ValueBenchmark/MBPP-Pass@1\nfim\_0\nfim\_0.5\nfim\_1.0\nmsp\_0.5\nFigure 3|The effectiveness of using FIM objective.\nTo determine the effectiveness of various hyperparameters within the FIM approach, we\nconducted a series of ablation experiments.\nExperiment Settings: In this experiment, we employ DeepSeek-Coder-Base 1.3B as our model\narchitecture. We focused on a Python subset from our training dataset to streamline the ex-\nperimental process. Our primary objective was to assess the efficacy of the Fill-in-the-Middle\n(FIM) technique, utilizing the HumanEval-FIM benchmark (Fried et al., 2022). This benchmark\nspecializes in a single-line FIM task for Python, in which one line of code from a HumanEval\nsolution is randomly obscured, testing the model’s proficiency in predicting the missing line.\nWe hypothesize that the PSM mode may exhibit subtle differences compared to the traditional\nnext-token prediction objective. This is primarily because PSM involves rearranging the order\nof the original text, potentially impacting the learning dynamics of the model. Therefore, we\nimplement the PSM mode for FIM across four distinct configurations: 0% FIM rate, 50% FIM\nrate, 100% FIM rate, and 50% MSP rate. The Masked Span Prediction (MSP) strategy, initially\nintroduced in T5 (Raffel et al., 2023), conceals multiple text spans and trains the model to recon-\nstruct these segments. According to CodeGen2.5 (Nijkamp et al., 2023), MSP may enhance FIM\nperformance compared to PSM. Thus, we include this method in our comparative analysis.\nResults: The outcomes of our experiment are illustrated in Figure 3. While the model demon-\nstrates peak performance on the HumanEval-FIM with a 100% FIM rate, this configuration also\nresults in the weakest code completion capability. This indicates a trade-off between FIM and\n7', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 6}),

Document(page\_content='code completion abilities. Moreover, we observe that with a 50% PSM rate, the model outper-\nforms the MSP strategy. To achieve a balance between FIM efficiency and code completion\nproficiency, we ultimately choose the 50% PSM rate as our preferred training policy.\nIn our implementation, we have introduced three sentinel tokens specifically for this task.\nFor each code file, we initially divide its content into three segments, denoted as 𝑓𝑝𝑟𝑒,𝑓𝑚𝑖𝑑𝑑𝑙𝑒 , and\n𝑓𝑠𝑢𝑓. Using the PSM mode, we construct the training example as follows:\n<｜fim\_start ｜>𝑓𝑝𝑟𝑒<｜fim\_hole ｜>𝑓𝑠𝑢𝑓<｜fim\_end ｜>𝑓𝑚𝑖𝑑𝑑𝑙𝑒<|eos\_token|>\nWe implement the Fill-in-the-Middle (FIM) method at the document level before the packing\nprocess, as proposed in the original work by Bavarian et al. (2022). This is done with an FIM\nrate of 0.5, following the PSM mode.\n3.2. Tokenizer\nFor the tokenization process, we employ the HuggingFace Tokenizer library2to train Byte Pair\nEncoding (BPE) tokenizers, as outlined in Sennrich et al. (2015) (Sennrich et al., 2015), on a\nsubset of our training corpus. Ultimately, we utilize a tokenizer configured with a vocabulary\nsize of 32,000.\n3.3. Model Architecture\nWe develop a range of models with varying parameters to cater to diverse applications, including\nmodels with 1.3B, 6.7B, and 33B parameters. These models are built upon the same framework\nas the DeepSeek Large Language Model (LLM) outlined by DeepSeek-AI (2024). Each model is a\ndecoder-only Transformer, incorporating Rotary Position Embedding (RoPE) as described by Su\net al. (2023). Notably, the DeepSeek 33B model integrates Grouped-Query-Attention (GQA) with\na group size of 8, enhancing both training and inference efficiency. Additionally, we employ\nFlashAttention v2 (Dao, 2023) to expedite the computation involved in the attention mechanism.\nThe architectural details of our models are summarized in Table 2.\n3.4. Optimization\nFollowing DeepSeek LLM (DeepSeek-AI, 2024), we use AdamW (Loshchilov and Hutter, 2019)\nas the optimizer with 𝛽1and𝛽2values of 0.9 and 0.95. We adapt batch sizes and learning rates by\nthe scaling laws suggested in DeepSeek LLM. For the learning rate scheduling, we implement a\nthree-stage policy, which includes 2000 warm-up steps, and set the final learning rate to 10% of\nthe initial rate. Notably, the learning rate at each stage is scaled down to√︃\n1\n10of the preceding\nstage’s rate, following the guidelines established in DeepSeek LLM (DeepSeek-AI, 2024).\n3.5. Environments\nOur experiments are conducted using the HAI-LLM (High-Flyer, 2023) framework, known for\nits efficiency and lightweight approach in training large language models. This framework\nincorporates a variety of parallelism strategies to optimize computational efficiency. These in-\nclude tensor parallelism (Korthikanti et al., 2023), alongside ZeRO data parallelism (Rajbhandari\net al., 2020) and PipeDream pipeline parallelism (Narayanan et al., 2019). Our experiments\n2https://github.com/huggingface/tokenizers\n8', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 7}),

Document(page\_content='Hyperparameter DeepSeek-Coder 1.3B DeepSeek-Coder 6.7B DeepSeek-Coder 33B\nHidden Activation SwiGLU SwiGLU SwiGLU\nHidden size 2048 4096 7168\nIntermediate size 5504 11008 19200\nHidden layers number 24 32 62\nAttention heads number 16 32 56\nAttention Multi-head Multi-head Grouped-query (8)\nBatch Size 1024 2304 3840\nMax Learning Rate 5.3e-4 4.2e-4 3.5e-4\nTable 2|Hyperparameters of DeepSeek-Coder.\nutilize clusters outfitted with NVIDIA A100 and H800 GPUs. In the A100 cluster, each node\nis configured with 8 GPUs, interconnected in pairs using NVLink bridges. The H800 cluster is\nsimilarly arranged, with each node containing 8 GPUs. These GPUs are interconnected using\na combination of NVLink and NVSwitch technologies, ensuring efficient data transfer within\nnodes. To facilitate seamless communication between nodes in both A100 and H800 clusters, we\nemploy InfiniBand interconnects, known for their high throughput and low latency. This setup\nprovides a robust and efficient infrastructure for our computational experiments.\n3.6. Long Context\nTo enhance the capabilities of DeepSeek-Coder in handling extended contexts, particularly for\nscenarios like repository-level code processing, we have reconfigured the RoPE (Su et al., 2023)\nparameters to extend the default context window. Following previous practices (Chen et al.,\n2023; kaiokendev, 2023), we employed a linear scaling strategy, increasing the scaling factor from\n1to4and altering the base frequency from 10000 to100000 . The model underwent an additional\n1000 steps of training, using a batch size of 512and a sequence length of 16K. The learning rate\nwas maintained as in the final pre-training phase. Theoretically, these modifications enable our\nmodel to process up to 64K tokens in context. However, empirical observations suggest that\nthe model delivers its most reliable outputs within a 16K token range. Future research will\ncontinue to refine and evaluate the long-context adaptation methodology, aiming to further\nenhance DeepSeek-Coder’s efficiency and user-friendliness in processing extended contexts.\n3.7. Instruction Tuning\nWe develop DeepSeek-Coder-Instruct by enhancing the DeepSeek-Coder-Base through instruction-\nbased fine-tuning using high-quality data. This data comprises helpful and impartial human\ninstructions, structured by the Alpaca Instruction format (Taori et al., 2023). To demarcate each\ndialogue turn, we employed a unique delimiter token <|EOT|> to signify the conclusion of each\nsegment. For training, we use a cosine schedule with 100 warm-up steps and an initial learning\nrate 1e-5. We also use a batch size of 4M tokens and 2B tokens in total.\nAn example of using DeepSeek-Coder-Instruct 34B is depicted in Figure 4. This example is\na multi-turn dialogue scenario for building a snake game. Initially, we ask the model to write\na game snake using pygame. The model successfully creates a basic snake game that can run\nwithout bugs. To improve the game, we further request adding a scoring system in the top left\ncorner. The model then introduces a "score" variable and a "display\_score" function, along with\nan explanation of how to integrate these features. This example illustrates DeepSeek-Coder-\nInstruct’s ability to provide complete solutions in multi-turn dialogue settings. More cases can\nbe found in the Appendix A.\n9', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 8}),

Document(page\_content="Q1: Write a game snake using pygame\nSure, here is a simple implementation of the classic Snake game using Pygame. This code will create a window with a snake that can be controlled using the arrow keys. The snake will grow in length each time it eats the food. The game will end if the snake hits the edge of the window or its own tail.```pythonimport pygame import random…running = TrueWhile running:…# Draw everything screen.fill(BLACK) snake.draw() food.draw() pygame.display.flip() # Cap the frame rate clock.tick(SPEED) # Quit Pygame pygame.quit()```This code sets up a basic game loop where the snake moves in response to key presses, and the game ends if the snake hits the edge of the window or its own tail. The snake grows in length each time it eats the food. The game continues until the player closes the window.\nRun CodeQ2: Add a scoring system in the top left corner\nTo add a scoring system to the top left corner of the game window, you can create a function to display the score and increment it each time the snake eats the food. Here's how you can modify the code to include a scoring system:```python…running = TrueWhile running:…# Draw everything screen.fill(BLACK) snake.draw() food.draw()display\_score(score) # Display the scorepygame.display.flip() # Cap the frame rate clock.tick(SPEED) # Quit Pygame pygame.quit()```In this updated code, we've added a`score`variable to keep track of the player's score, which is incremented each time the snake eats the food. The`display\_score`function is used to render the score text and display it in the top left corner of the window. The`display\_score`function is called after the snake has moved and the screen has been filled with the black color, but before the`pygame.display.flip()`call, so that the score is drawn on top of everything else.Run Code\nFigure 4|An example of responses from DeepSeek-Coder-Instruct 33B in a multi-turn setting.\n4. Experimental Results\nIn this section, we evaluate DeepSeek-Coder on four tasks, including code generation (§4.1), FIM\ncode completion (§4.2), cross-file code completion (§4.3) and program-based math reasoning\n(§4.4). We compare DeepSeek-Coder with the previous state-of-the-art large language models:\n•CodeGeeX2 (Zheng et al., 2023) represents the second generation of the multilingual\ncode generation model CodeGeeX. It is developed using the ChatGLM2 (Du et al., 2022)\narchitecture and is enhanced with an extensive dataset of coding examples.\n•StarCoder (Li et al., 2023) is a publicly accessible model with a substantial parameter count\nof 15 billion. It is specifically trained on a meticulously curated subset of the Stack dataset\n(Kocetkov et al., 2022), covering 86 programming languages, ensuring its proficiency\nacross a wide range of coding tasks.\n•CodeLlama (Roziere et al., 2023) encompasses a series of code-centric Large Language\nModels (LLMs) that are derivatives of LLaMA2 (Touvron et al., 2023). Available in three\nsizes — 7B, 13B, and 34B — these models undergo continued training on a vast 500 billion\ntoken code corpus, building upon the foundational LLaMA2 architecture.\n•code-cushman-001 Chen et al. (2021) is a 12 billion parameter model developed by OpenAI\nand served as the initial model for Github Copilot.\n•GPT-3.5 and GPT-4 (OpenAI, 2023) are advanced generative AI models developed by\nOpenAI. While they are not explicitly trained for code generation, they also demonstrate\n10", metadata={'source': 'papers/rachelgreecv.pdf', 'page': 9}),

Document(page\_content='notable performance in this domain. Their effectiveness in handling code generation tasks\nis largely attributed to their massive scale in terms of parameter count.\n4.1. Code Generation\nHumanEval and MBPP Benchmarks The HumanEval (Chen et al., 2021) and MBPP (Austin\net al., 2021) benchmarks are widely used for evaluating code LLMs. HumanEval consists of 164\nhand-written Python problems that are validated using test cases to assess the code generated\nby a Code LLM in a zero-shot setting, while the MBPP benchmark includes 500 problems\nin a few-shot setting. To evaluate the model’s multilingual capabilities, we expanded the\nPython problems of Humaneval Benchmark to seven additional commonly used programming\nlanguages, namely C++, Java, PHP , TypeScript (TS), C#, Bash, and JavaScript (JS) (Cassano et al.,\n2023). For both benchmarks, We adopted a greedy search approach and re-implemented the\nbaseline results using the same script and environment for fair comparison.\nModel Size Python C++ Java PHP TS C# Bash JS Avg MBPP\nMultilingual Base Models\ncode-cushman-001 12B 33.5% 31.9% 30.6% 28.9% 31.3% 22.1% 11.7% - - -\nCodeGeeX2 6B 36.0% 29.2% 25.9% 23.6% 20.8% 29.7% 6.3% 24.8% 24.5% 36.2%\nStarCoderBase 16B 31.7% 31.1% 28.5% 25.4% 34.0% 34.8% 8.9% 29.8% 28.0% 42.8%\nCodeLlama 7B 31.7% 29.8% 34.2% 23.6% 36.5% 36.7% 12.0% 29.2% 29.2% 38.6%\nCodeLlama 13B 36.0% 37.9% 38.0% 34.2% 45.2% 43.0% 16.5% 32.3% 35.4% 48.4%\nCodeLlama 34B 48.2% 44.7% 44.9% 41.0% 42.1% 48.7% 15.8% 42.2% 41.0% 55.2%\nDeepSeek-Coder-Base 1.3B 34.8% 31.1% 32.3% 24.2% 28.9% 36.7% 10.1% 28.6% 28.3% 46.2%\nDeepSeek-Coder-Base 6.7B 49.4% 50.3% 43.0% 38.5% 49.7% 50.0% 28.5% 48.4% 44.7% 60.6%\nDeepSeek-Coder-Base 33B 56.1% 58.4% 51.9% 44.1% 52.8% 51.3% 32.3% 55.3% 50.3% 66.0%\nInstruction-Tuned Models\nGPT-3.5-Turbo - 76.2% 63.4% 69.2% 60.9% 69.1% 70.8% 42.4% 67.1% 64.9% 70.8%\nGPT-4 - 84.1% 76.4% 81.6% 77.2% 77.4% 79.1% 58.2% 78.0% 76.5% 80.0%\nDeepSeek-Coder-Instruct 1.3B 65.2% 45.3% 51.9% 45.3% 59.7% 55.1% 12.7% 52.2% 48.4% 49.4%\nDeepSeek-Coder-Instruct 6.7B 78.6% 63.4% 68.4% 68.9% 67.2% 72.8% 36.7% 72.7% 66.1% 65.4%\nDeepSeek-Coder-Instruct 33B 79.3% 68.9% 73.4% 72.7% 67.9% 74.1% 43.0% 73.9% 69.2% 70.0%\nTable 3|Performance of approaches on the Multilingual HumanEval and MBPP Benchmarks.\nThe results are presented in Table 3. As we can see, DeepSeek-Coder-Base achieves state-\nof-the-art performance with an average accuracy of 50.3% on HumanEval and 66.0% on MBPP .\nIn comparison to the similarly sized open-source model CodeLlama-Base 34B, our model has\ndemonstrated a notable improvement of 9% and 11% in accuracy, respectively. It’s worth\nnoting that even our smaller model, DeepSeek-Coder-Base 6.7B, surpasses the performance\nof CodeLlama-Base 34B. After instruction fine-tuning, our model surpasses the closed-source\nGPT-3.5-Turbo model in HumanEval benchmark, significantly reducing the performance gap\nbetween OpenAI GPT-4 and open-source models.\nDS-1000 Benchmark HumanEval and MBPP have a significant drawback in that they rely\nheavily on straightforward programming tasks that may not accurately represent the kind of\ncode most programmers typically write. In contrast, the DS-1000 benchmark, as introduced in\nthe work by Lai et al. (2023), offers a comprehensive collection of 1,000 practical and realistic data\nscience workflows across seven different libraries. This benchmark evaluates code generation by\nexecuting it against specific test cases. What sets DS-1000 apart is its categorization of problems\nbased on the libraries involved, which encompass Matplotlib, NumPy, Pandas, SciPy, Scikit-\n11', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 10}),

Document(page\_content='Learn, PyTorch, and TensorFlow. The benchmark assesses the performance of base models in the\ncode completion setting and we provide pass@1 results for each library, as well as overall score.\nThe results of DS-1000 benchmark are shown in Table 4. As can be seen from the table, the\nDeepSeek-Coder model achieves relatively high accuracy in all libraries, demonstrating that our\nmodel is not only capable of generating good code but also of using libraries more accurately in\nreal data science workflows.\nModel Size Matplotlib Numpy Pandas Pytorch Scipy Scikit-Learn Tensorflow Avg\nCodeGeeX2 6B 38.7% 26.8% 14.4% 11.8% 19.8% 27.0% 17.8% 22.9%\nStarCoder-Base 16B 43.2% 29.1% 11.0% 20.6% 23.6% 32.2% 15.6% 24.6%\nCodeLlama-Base 7B 41.9% 24.6% 14.8% 16.2% 18.9% 17.4% 17.8% 22.1%\nCodeLlama-Base 13B 46.5% 28.6% 18.2% 19.1% 18.9% 27.8% 33.3% 26.8%\nCodeLlama-Base 34B 50.3% 42.7% 23.0% 25.0% 28.3% 33.9% 40.0% 34.3%\nDeepSeek-Coder-Base 1.3B 32.3% 21.4% 9.3% 8.8% 8.5% 16.5% 8.9% 16.2%\nDeepSeek-Coder-Base 6.7B 48.4% 35.5% 20.6% 19.1% 22.6% 38.3% 24.4% 30.5%\nDeepSeek-Coder-Base 33B 56.1% 49.6% 25.8% 36.8% 36.8% 40.0% 46.7% 40.2%\nTable 4|Performance of different approaches on the DS-1000-Tasks.\nLeetCode Contest Benchmark To further validate the model’s capability in real-world pro-\ngramming problems, we construct the LeetCode Contest benchmark3. LeetCode4presents\ncompetition-level problems, offering significant challenges that test the model’s problem under-\nstanding and code generation skills. We collected the latest problems from LeetCode Contests to\nprevent the appearance of both the problems or their solutions in our pre-training data. A total\nof 180 problems were collected from July 2023 to January 2024. For each problem, we collected\n100 test cases to ensure the test coverage. We use the template "{problem\_description}\\nPlease\ncomplete the code below to solve the above problem:\\n ```python\\n{code\_template}\\n ```" to\nbuild the instruction prompt.\nThe evaluation results are shown in Table 5. In our evaluation, the DeepSeek-Coder models\ndemonstrate remarkable performance over current open-source coding models. Specifically, the\nDeepSeek-Coder-Instruct 6.7B and 33B achieve Pass@1 scores of 19.4% and 27.8% respectively\nin this benchmark. This performance notably surpasses existing open-sourced models such as\nCode-Llama-33B. The DeepSeek-Coder-Instruct 33B is the only open-sourced model that outper-\nforms OpenAI’s GPT-3.5-Turbo in this task. However, there remains a substantial performance\ngap when compared to the more advanced GPT-4-Turbo.\nOur analysis indicates that the implementation of Chain-of-Thought (CoT) prompting no-\ntably enhances the capabilities of DeepSeek-Coder-Instruct models. This improvement becomes\nparticularly evident in the more challenging subsets of tasks. By adding the directive, "You\nneed first to write a step-by-step outline and then write the code." following the initial prompt,\nwe have observed enhancements in performance. This observation leads us to believe that the\nprocess of first crafting detailed code descriptions assists the model in more effectively under-\nstanding and addressing the intricacies of logic and dependencies in coding tasks, particularly\nthose of higher complexity. Therefore, we strongly recommend employing CoT prompting strate-\ngies when utilizing DeepSeek-Coder-Instruct models for complex coding challenges. Such an\napproach promotes a more methodical and logical framework for problem-solving, potentially\nresulting in more precise and efficient outcomes in code generation tasks.\n3We have published this benchmark in [https://github.com/deepseek-ai/DeepSeek-Coder/tree/main\n/Evaluation/LeetCode](https://github.com/deepseek-ai/DeepSeek-Coder/tree/main/n/Evaluation/LeetCode) .\n4https://leetcode.com/\n12', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 11}),

Document(page\_content='Model Size Easy (45) Medium (91) Hard (44) Overall(180)\nWizardCoder-V1.0 15B 17.8% 1.1% 0.0% 5.0%\nCodeLlama-Instruct 34B 24.4% 4.4% 4.5% 9.4%\nPhind-CodeLlama-V2 34B 26.7% 8.8% 9.1% 13.3%\nGPT-3.5-Turbo - 46.7% 15.4 % 15.9% 23.3%\nGPT-3.5-Turbo + CoT - 42.2% 15.4% 20.5% 23.3%\nGPT-4-Turbo - 73.3% 31.9% 25.0% 40.6%\nGPT-4-Turbo + CoT - 71.1% 35.2% 25.0% 41.8%\nDeepSeek-Coder-Instruct 1.3B 22.2% 1.1% 4.5% 7.2%\nDeepSeek-Coder-Instruct + CoT 1.3B 22.2% 2.2% 2.3% 7.2%\nDeepSeek-Coder-Instruct 6.7B 44.4% 12.1% 9.1% 19.4%\nDeepSeek-Coder-Instruct + CoT 6.7B 44.4% 17.6% 4.5% 21.1%\nDeepSeek-Coder-Instruct 33B 57.8% 22.0% 9.1% 27.8%\nDeepSeek-Coder-Instruct + CoT 33B 53.3% 25.3% 11.4% 28.9%\nTable 5|Performance of different models on the LeetCode Contest Benchmark.\nIt is important to acknowledge that despite our diligent efforts to gather the most recent\ncode questions for model testing, the possibility of data contamination cannot be entirely ruled\nout. We observed that the GPT-4-Turbo and DeepSeek-Coder models achieved higher scores\nin the LeetCode Contest held in July and August. We encourage the research community to\nconsider the potential issue of data contamination when evaluating models in future studies\nusing our released LeetCode data.\n4.2. Fill-in-the-Middle Code Completion\nDeepSeek-Coder models are trained with a 0.5 FIM (Fill-In-the-Middle) rate during their pre-\ntraining phase. This specialized training strategy empowers the model to proficiently generate\ncode by filling in blanks based on the surrounding context, both prefix and suffix, of the given\ncode snippet. This capability is particularly advantageous in the realm of code completion\ntools. Several open-source models have emerged with similar capabilities. Notable among these\nare SantaCoder (Allal et al., 2023), StarCoder (Li et al., 2023), and CodeLlama (Roziere et al.,\n2023). These models have set a precedent in the field of code generation and completion. In\nevaluating the performance DeepSeek-Coder models, we conducted a comparative analysis with\nthe aforementioned models. The benchmark for this comparison was the Single-Line Infilling\nbenchmarks, encompassing three different programming languages, as proposed by Allal et al.\n(2023). This benchmark uses the line exact match accuracy as the evaluation metric.\nModel Size python java javascript Mean\nSantaCoder 1.1B 44.0% 62.0% 74.0% 69.0%\nStarCoder 16B 62.0% 73.0% 74.0% 69.7%\nCodeLlama-Base 7B 67.6% 74.3% 80.2% 69.7%\nCodeLlama-Base 13B 68.3% 77.6% 80.7% 75.5%\nDeepSeek-Coder-Base 1B 57.4% 82.2% 71.7% 70.4%\nDeepSeek-Coder-Base 7B 66.6% 88.1% 79.7% 80.7%\nDeepSeek-Coder-Base 33B 65.4% 86.6% 82.5% 81.2%\nTable 6|Performance of different approaches on the FIM-Tasks.\nThe evaluation results are shown in Table 6. Despite being the smallest model with a capacity\n13', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 12}),

Document(page\_content='of 1.3 billion parameters, DeepSeek-Coder outperforms its larger counterparts, StarCoder and\nCodeLlama, in these benchmarks. This superior performance can be attributed to the high\nquality of the pre-trained data utilized by DeepSeek-Coder. Furthermore, a notable trend\nobserved is the correlation between the size of the model and its performance. As the model\nsize increases, there is a corresponding and responsible enhancement in performance. This\ntrend underscores the importance of model capacity in achieving higher accuracy in code\ncompletion tasks. Based on these findings, we recommend the deployment of the DeepSeek-\nCoder-Base 6.7B model in code completion tools. This recommendation is grounded in the\nmodel’s demonstrated balance between efficiency and accuracy. The DeepSeek-Coder-Base 6.7B\nmodel, with its substantial parameter size, has proven to be highly effective in the context of\ncode completion, making it an ideal choice for integrating advanced computational capabilities\ninto coding environments.\n4.3. Cross-File Code Completion\nIn this section, we will evaluate the performance of existing open-source models in cross-file\ncode completion tasks. Unlike code generation discussed in the previous section, cross-file code\ncompletion requires the model to access and understand repositories that span multiple files\nwith numerous cross-file dependencies. We use CrossCodeEval (Ding et al., 2023) to evaluate the\ncapabilities of currently available open-source code models of 7B scale in cross-file completion\ntasks. This dataset is constructed on a diverse set of real-world, open-sourced, permissively\nlicensed repositories in four popular programming languages: Python, Java, TypeScript, and C#.\nThe dataset is specifically designed to strictly require cross-file context for accurate completion.\nNotably, this dataset was constructed from repositories created between March and June 2023,\nwhile our pre-training data only includes code created before February 2023, which ensures that\nthis dataset was not present in our pre-training data, thus avoiding data leakage.\nModel SizePython Java TypeScript C#\nEM ES EM ES EM ES EM ES\nCodeGeex2 6B 8.11% 59.55% 7.34% 59.60% 6.14% 55.50% 1.70% 51.66%\n+ Retrieval 10.73% 61.76% 10.10% 59.56% 7.72% 55.17% 4.64% 52.30%\nStarCoder-Base 7B 6.68% 59.55% 8.65% 62.57% 5.01% 48.83% 4.75% 59.53%\n+ Retrieval 13.06% 64.24% 15.61% 64.78% 7.54% 42.06% 14.20% 65.03%\nCodeLlama-Base 7B 7.32% 59.66% 9.68% 62.64% 8.19% 58.50% 4.07% 59.19%\n+ Retrieval 13.02% 64.30% 16.41% 64.64% 12.34% 60.64% 13.19% 63.04%\nDeepSeek-Coder-Base 6.7B 9.53% 61.65% 10.80% 61.77% 9.59% 60.17% 5.26% 61.32%\n+ Retrieval 16.14% 66.51% 17.72% 63.18% 14.03% 61.77% 16.23% 63.42%\n+ Retrieval w/o Repo Pre-training 16.02% 66.65% 16.64% 61.88% 13.23% 60.92% 14.48% 62.38%\nTable 7|Performance of different models on cross-file code completion.\nIn our evaluation of various models, we set the maximum sequence length to 2048 tokens,\nthe maximum output length to 50 tokens, and a limit of 512 tokens for the cross-file context. For\nthe cross-file context, we utilize the official BM25 search results provided by Ding et al. (2023).\nEvaluation metrics include exact match and edit similarity. The results, presented in Table 7,\ndemonstrate that DeepSeek-Coder consistently outperforms other models in cross-file comple-\ntion tasks across multiple languages, showcasing its superior practical application capabilities.\nWhen only utilizing file-level code corpus ( w/o Repo Pre-training ) to pre-train DeepSeek-Coder,\nwe observe a decrease in performance in the Java, TypeScript, and C# languages, indicating the\neffectiveness of the repository-level pre-training.\n14', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 13}),

Document(page\_content='4.4. Program-based Math Reasoning\nProgram-based math reasoning involves evaluating a model’s ability to understand and solve\nmathematical problems through programming. This type of reasoning is critical in fields\nsuch as data analysis and scientific computing. To conduct this assessment, we utilize the\nProgram-Aided Math Reasoning (PAL) method as outlined in Gao et al. (2023). This approach\nis applied across seven distinct benchmarks, each offering unique challenges and contexts.\nThese benchmarks includes GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), GSM-\nHard (Gao et al., 2023), SVAMP (Patel et al., 2021), TabMWP (Lu et al., 2022), ASDiv (Miao\net al., 2020) and MAWPS (Gou et al., 2023). In each of these benchmarks, the model is prompted\nto alternately describe a solution step in natural language and then execute that step with\ncode. As seen in Table 8, DeepSeek-Coder models achieve a remarkable performance across all\nbenchmarks, especially the 33B variant, which demonstrates the potential of using such models\nin applications that require complex mathematical computations and problem-solving abilities.\nModel Size GSM8k MATH GSM-Hard SVAMP TabMWP ASDiv MAWPS Avg\nMultilingual Base Models\nCodeGeex-2 7B 22.2% 9.7% 23.6% 39.0% 44.6% 48.5% 66.0% 36.2%\nStarCoder-Base 16B 23.4% 10.3% 23.0% 42.4% 45.0% 54.9% 81.1% 40.0%\nCodeLlama-Base 7B 31.2% 12.1% 30.2% 54.2% 52.9% 59.6% 82.6% 46.1%\nCodeLlama-Base 13B 43.1% 14.4% 40.2% 59.2% 60.3% 63.6% 85.3% 52.3%\nCodeLlama-Base 34B 58.2% 21.2% 51.8% 70.3% 69.8% 70.7% 91.8% 62.0%\nDeepSeek-Coder-Base 1.3B 14.6% 16.8% 14.5% 36.7% 30.0% 48.2% 62.3% 31.9%\nDeepSeek-Coder-Base 6.7B 43.2% 19.2% 40.3% 58.4% 67.9% 67.2% 87.0% 54.7%\nDeepSeek-Coder-Base 33B 60.7% 29.1% 54.1% 71.6% 75.3% 76.7% 93.3% 65.8%\nTable 8|Performance of different approaches on the program-aid math reasoning tasks.\n5. Continue Pre-Training From General LLM\nTo further enhance the natural language understanding and mathematical reasoning abilities\nof the DeepSeek-Coder model, we perform additional pre-training from the general language\nmodel DeepSeek-LLM-7B Base (DeepSeek-AI, 2024) on 2 trillion tokens, resulting in DeepSeek-\nCoder-v1.5 7B. For this pre-training, we specifically use the data sources listed in Table 9. Unlike\nDeepSeek-Coder, DeepSeek-Coder-v1.5 employs solely a next token prediction objective with a\n4K context length during its pre-training phase.\nData Source Percentage\nSource Code 70%\nMarkdown and StackExchange 10%\nNatural language related to code 7%\nNatural language related to math 7%\nBilingual (Chinese-English) natural language 6%\nTable 9|Data sources for DeepSeek-Coder-v1.5 7B pre-training\nWe conduct a comparison between DeepSeek-Coder-v1.5 7B and DeepSeek-Coder 6.7B, and\nre-run all benchmarks using our evaluation pipeline to ensure a fair comparison. We evaluate\nperformance across a wide range of tasks, which can be categorized as follows:\n15', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 14}),

Document(page\_content='•Programming : This category includes evaluations in a multilingual setting using the\nHumanEval dataset by Chen et al. (2021), as well as evaluations in a Python setting using\nthe MBPP dataset by Austin et al. (2021)\n•Math Reasoning : We assess performance on math reasoning tasks using the GSM8K\nbenchmark (Cobbe et al., 2021) and the MATH (Hendrycks et al., 2021) benchmark [4].\nThese tasks involve solving math problems by generating programs.\n•Natural Language Our evaluation in natural language tasks includes MMLU (Hendrycks\net al., 2020), BBH (Suzgun et al., 2022), HellaSwag (Zellers et al., 2019), Winogrande (Sak-\naguchi et al., 2021), and ARC-Challenge (Clark et al., 2018) benchmarks.\nThe results for the Base and Instruct models are presented in Table 10. It is observed\nthat the DeepSeek-Coder-Base-v1.5 model, despite a slight decrease in coding performance,\nshows marked improvements across most tasks when compared to the DeepSeek-Coder-Base\nmodel. In particular, in the Math Reasoning and Natural Language categories, DeepSeek-\nCoder-Base-v1.5 significantly outperforms its predecessor across all benchmarks, which also\ndemonstrates significant improvements in its mathematical reasoning and natural language\nprocessing capabilities.\nProgramming Math Reasoning Natural Language\nModels Size HumanEval MBPP GSM8K MATH MMLU BBH HellaSwag WinoG ARC-C\nDeepSeek-Coder-Base 6.7B 44.7% 60.6% 43.2% 19.2% 36.6% 44.3% 53.8% 57.1% 32.5%\nDeepSeek-Coder-Base-v1.5 6.9B 43.2% 60.4% 62.4% 24.7% 49.1% 55.2% 69.9% 63.8% 47.2%\nDeepSeek-Coder-Instruct 6.7B 66.1% 65.4% 62.8% 28.6% 37.2% 46.9% 55.0% 57.6% 37.4%\nDeepSeek-Coder-Instruct-v1.5 6.9B 64.1% 64.6% 72.6% 34.1% 49.5% 53.3% 72.2% 63.4% 48.1%\nTable 10|Comparative analysis of performance between DeepSeek-Coder-Base and\nDeepSeek-Coder-Base-v1.5. Math tasks are solved through programming.\n6. Conclusion\nIn this technical report, we introduce a series of specialized Large Language Models (LLMs) for\ncoding, named DeepSeek-Coder, available in three distinct scales: 1.3B, 6.7B, and 33B parameters.\nThese models are uniquely trained on a meticulously curated project-level code corpus, utilizing\na "fill-in-the-blank" pre-training objective to enhance code infilling capabilities. A significant\nadvancement is the extension of the models’ context window to 16,384 tokens, thereby greatly\nimproving their effectiveness in handling extensive code generation tasks. Our evaluations\nreveal that the most advanced model in our series, DeepSeek-Coder-Base 33B surpasses existing\nopen-source code models across a variety of standard tests. Impressively, the DeepSeek-Coder-\nBase 6.7B model, despite its smaller scale, delivers performance on par with the 34B parameter\nCodeLlama, a testament to the high quality of our pretraining corpus.\nTo augment the zero-shot instruction capabilities of the DeepSeek-Coder-Base models, we\nhave fine-tuned them with high-quality instructional data. This has led to the DeepSeek-Coder-\nInstruct 33B model outperforming OpenAI’s GPT-3.5 Turbo in a range of coding-related tasks,\nshowcasing its exceptional proficiency in code generation and understanding.\nTo further improve the natural language understanding capabilities of the DeepSeek-Coder-\nBase models, we have conducted additional pretraining based on the DeepSeek-LLM 7B check-\npoint. This additional training involved processing a diverse dataset comprising 2 billion tokens,\nincluding natural language, code, and mathematical data. The result is the creation of a new\n16', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 15}),

Document(page\_content='and improved code model, DeepSeek-Coder-v1.5. Our observations indicate that DeepSeek-\nCoder-v1.5 not only maintains its predecessor’s high-level coding performance but also exhibits\nenhanced natural language comprehension. This advancement underscores our belief that the\nmost effective code-focused Large Language Models (LLMs) are those built upon robust general\nLLMs. The reason is evident: to effectively interpret and execute coding tasks, these models\nmust also possess a deep understanding of human instructions, which often come in various\nforms of natural language. Looking ahead, our commitment is to develop and openly share\neven more powerful code-focused LLMs based on larger-scale general LLMs.\nAcknowledgements\nWe would like to express our gratitude to Bo Liu, Chengqi Deng, Chong Ruan, Damai Dai,\nJiashi Li, Kang Guan, Mingchuan Zhang, Panpan Huang, Shuiping Yu, Shirong Ma, Yaofeng\nSun, Yishi Piao, Zhihong Shao, and Zhewen Hao for their invaluable discussions and assistance\nduring training DeepSeek-Coder models.\nReferences\nL. B. Allal, R. Li, D. Kocetkov, C. Mou, C. Akiki, C. M. Ferrandis, N. Muennighoff, M. Mishra,\nA. Gu, M. Dey, et al. Santacoder: don’t reach for the stars! arXiv preprint arXiv:2301.03988 ,\n2023.\nJ. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry,\nQ. Le, and C. Sutton. Program synthesis with large language models, 2021.\nM. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen. Efficient\ntraining of language models to fill in the middle. arXiv preprint arXiv:2207.14255, 2022.\nF. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee,\nY. Zi, C. J. Anderson, M. Q. Feldman, et al. Multipl-e: a scalable and polyglot approach to\nbenchmarking neural code generation. IEEE Transactions onSoftware Engineering, 2023.\nM. Chen, J. Tworek, H. Jun, Q. Yuan, H. P . d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda,\nN. Joseph, G. Brockman, et al. Evaluating large language models trained on code. arXiv\npreprint arXiv:2107.03374, 2021.\nS. Chen, S. Wong, L. Chen, and Y. Tian. Extending context window of large language models via\npositional interpolation. arXiv preprint arXiv:2306.15595, 2023.\nP . Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think\nyou have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint\narXiv:1803.05457, 2018.\nK. Cobbe, V . Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek,\nJ. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. arXiv preprint\narXiv:2110.14168, 2021.\nT. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023.\nDeepSeek-AI. Deepseek llm: Scaling open-source language models with longtermism. arXiv\npreprint arXiv:2401.02954, 2024.\n17', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 16}),

Document(page\_content='Y. Ding, Z. Wang, W. U. Ahmad, H. Ding, M. Tan, N. Jain, M. K. Ramanathan, R. Nallapati,\nP . Bhatia, D. Roth, et al. Crosscodeeval: A diverse and multilingual benchmark for cross-file\ncode completion. In Thirty-seventh Conference onNeural Information Processing Systems\nDatasets and Benchmarks Track, 2023.\nZ. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang. Glm: General language model\npretraining with autoregressive blank infilling. In Proceedings ofthe60th Annual Meeting\noftheAssociation forComputational Linguistics (Volume 1:Long Papers) , pages 320–335,\n2022.\nD. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-t. Yih, L. Zettlemoyer,\nand M. Lewis. Incoder: A generative model for code infilling and synthesis. arXiv preprint\narXiv:2204.05999, 2022.\nL. Gao, A. Madaan, S. Zhou, U. Alon, P . Liu, Y. Yang, J. Callan, and G. Neubig. Pal: Program-\naided language models. In International Conference onMachine Learning , pages 10764–\n10799. PMLR, 2023.\nG. Gemini Team. Gemini: A family of highly capable multimodal models, 2023. URL https:\n//goo.gle/GeminiPaper .\nZ. Gou, Z. Shao, Y. Gong, Y. Yang, M. Huang, N. Duan, W. Chen, et al. Tora: A tool-integrated\nreasoning agent for mathematical problem solving. arXiv preprint arXiv:2309.17452, 2023.\nD. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measur-\ning massive multitask language understanding. In International Conference onLearning\nRepresentations, 2020.\nD. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Mea-\nsuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874 ,\n2021.\nHigh-Flyer. Hai-llm: An efficient and lightweight tool for training large models. 2023. URL\nhttps://[www.high-flyer.cn/en/blog/hai-llm](http://www.high-flyer.cn/en/blog/hai-llm) .\nkaiokendev. Things i’m learning while training superhot. [https://kaiokendev.github.io\n/til#extending-context-to-8k](https://kaiokendev.github.io/n/til#extending-context-to-8k) , 2023.\nD. Kocetkov, R. Li, L. Jia, C. Mou, Y. Jernite, M. Mitchell, C. M. Ferrandis, S. Hughes, T. Wolf,\nD. Bahdanau, et al. The stack: 3 tb of permissively licensed source code. Transactions on\nMachine Learning Research, 2022.\nV . A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro.\nReducing activation recomputation in large transformer models. Proceedings ofMachine\nLearning and Systems, 5, 2023.\nY. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu.\nDs-1000: A natural and reliable benchmark for data science code generation. In International\nConference onMachine Learning, pages 18319–18345. PMLR, 2023.\nK. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini. Deduplicat-\ning training data makes language models better. In Proceedings ofthe60th Annual Meeting\noftheAssociation forComputational Linguistics (Volume 1:Long Papers) , pages 8424–8445,\n2022.\n18', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 17}),

Document(page\_content='R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim,\net al. Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161, 2023.\nI. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019.\nP . Lu, L. Qiu, K.-W. Chang, Y. N. Wu, S.-C. Zhu, T. Rajpurohit, P . Clark, and A. Kalyan. Dynamic\nprompt learning via policy gradient for semi-structured mathematical reasoning. In The\nEleventh International Conference onLearning Representations, 2022.\nS.-Y. Miao, C.-C. Liang, and K.-Y. Su. A diverse corpus for evaluating and developing english\nmath word problem solvers. In Proceedings ofthe58th Annual Meeting oftheAssociation\nforComputational Linguistics, pages 975–984, 2020.\nD. Narayanan, A. Harlap, A. Phanishayee, V . Seshadri, N. R. Devanur, G. R. Ganger, P . B.\nGibbons, and M. Zaharia. Pipedream: Generalized pipeline parallelism for dnn training.\nInProceedings ofthe27th ACM Symposium onOperating Systems Principles , pages 1–15,\n2019.\nE. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong. Codegen:\nAn open large language model for code with multi-turn program synthesis. arXiv preprint\narXiv:2203.13474, 2022.\nE. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou. Codegen2: Lessons for training\nllms on programming and natural languages, 2023.\nOpenAI. Gpt-4 technical report, 2023.\nA. Patel, S. Bhattamishra, and N. Goyal. Are nlp models really able to solve simple math\nword problems? In Proceedings ofthe2021 Conference oftheNorth American Chapter of\ntheAssociation forComputational Linguistics: Human Language Technologies , pages 2080–\n2094, 2021.\nC. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P . J. Liu.\nExploring the limits of transfer learning with a unified text-to-text transformer, 2023.\nS. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: Memory optimizations toward training tril-\nlion parameter models. In SC20: International Conference forHigh Performance Computing,\nNetworking, Storage and Analysis, pages 1–16. IEEE, 2020.\nB. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin,\net al. Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950, 2023.\nK. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd\nschema challenge at scale. Communications oftheACM, 64(9):99–106, 2021.\nR. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword\nunits. arXiv preprint arXiv:1508.07909, 2015.\nJ. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with\nrotary position embedding, 2023.\nM. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V . Le,\nE. H. Chi, D. Zhou, , and J. Wei. Challenging big-bench tasks and whether chain-of-thought\ncan solve them. arXiv preprint arXiv:2210.09261, 2022.\n19', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 18}),

Document(page\_content='R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P . Liang, and T. B. Hashimoto.\nStanford alpaca: An instruction-following llama model. [https://github.com/tatsu-lab\n/stanford\_alpaca](https://github.com/tatsu-lab/n/stanford_alpaca) , 2023.\nH. Touvron, L. Martin, K. Stone, P . Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra,\nP . Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv\npreprint arXiv:2307.09288, 2023.\nY. Wang, W. Wang, S. Joty, and S. C. Hoi. Codet5: Identifier-aware unified pre-trained encoder-\ndecoder models for code understanding and generation. arXiv preprint arXiv:2109.00859 ,\n2021.\nR. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really\nfinish your sentence? In Proceedings ofthe57th Annual Meeting oftheAssociation for\nComputational Linguistics, pages 4791–4800, 2019.\nQ. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, L. Shen, Z. Wang, A. Wang, Y. Li, et al.\nCodegeex: A pre-trained model for code generation with multilingual benchmarking on\nhumaneval-x. In Proceedings ofthe29th ACM SIGKDD Conference onKnowledge Discovery\nand Data Mining, pages 5673–5684, 2023.\n20', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 19}),

Document(page\_content='A. Cases of Chatting with DeepSeek-Coder-Instruct\nWe will present two cases of interactions with DeepSeek-Coder-Instruct, with one involving a\nmulti-turn conversation about creating a database and performing data analysis, and the other\ncentered around using a model to solve a sample problem from LeetCode.\nIn the first scenario, depicted in Figure 5, we instruct the model to build a student database\nusing Python and randomly insert 10 pieces of information. Subsequently, in the second round\nof the conversation, we continue to ask the model by analyzing the age distribution of the\nstudents. From Figure 5, it’s evident that the model can generate bug-free and comprehensive\ncode, accompanied by explanatory details. In the second scenario, as illustrated in Figure 6,\nwe further assess the model’s capabilities by testing it on an out-of-domain LeetCode contest\nproblem. This particular problem was released in November 2023, after our data collection, and\nthus, isn’t part of our model’s training data. The results show that our model excels at solving\nproblems that extend beyond its training distribution.\nQ1: Build a database in Python that includes various pieces of information about students, insert ten random records and display.\nSure, here is a simple example of how you can do this using SQLite in Python.```pythonimport sqlite3import random# Connect to SQLite databaseconn = sqlite3.connect(\'students.db\')# Create a cursor objectcur = conn.cursor()…# Create tablecur.execute(\'\'’CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY,name TEXT,age INTEGER,grade TEXT) \'\'\')…# Display all recordscur.execute(\'SELECT \* FROM students\')…```This script will create a SQLite database named \'students.db\' and a table named \'students\' with four columns: \'id\', \'name\', \'age\', and \'grade\'. It will then insert 10 random records into the table and display all records. …Run CodeQ2: create a graph depicting the age distribution.\nTo create a graph depicting the age distribution of the students, you can use thematplotliblibrary in Python. Here\'s how you can do it:```pythonimport sqlite3import pandas as pdimport matplotlib.pyplotas plt# Connect to SQLite databaseconn = sqlite3.connect(\'students.db\')# Read the data from the database into a pandas DataFramedf= pd.read\_sql\_query("SELECT \* from students", conn)# Create a histogram of the age distributionplt.hist(df[\'age\'], bins=range(10, 30, 2), edgecolor=\'black\')plt.title(\'Age Distribution of Students\')plt.xlabel(\'Age\')plt.ylabel(\'Number of Students\')plt.show()# Close the connectionconn.close()```This script will connect to the SQLite database, read the data into a pandas DataFrame, then create a histogram of the age distribution. The histogram will show the number of students in each age range.Run Code\nFigure 5|An example of building database and data analysis.\n21', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 20}),

Document(page\_content='Q: There arenteams numbered from0ton -1in a tournament; each team is also a node in aDAG.You are given the integernand a0-indexed2D integer arrayedgesof lengthmrepresenting theDAG, whereedges[i] = [ui, vi]indicates that there is a directed edge from teamuito teamviin the graph.A directed edge fromatobin the graph means that teamaisstrongerthan teamband teambisweakerthan teama.Teamawill be thechampionof the tournament if there is no teambthat isstrongerthan teama.Returnthe team that will be thechampionof the tournament if there is auniquechampion, otherwise, return-1.Notes•Acycleis a series of nodesa1, a2, ..., an, an+1such that nodea1is the same node as nodean+1, the nodesa1, a2, ..., anare distinct, and there is a directed edge from the nodeaito nodeai+1for everyiin the range[1, n].•ADAGis a directed graph that does not have anycycle.•Example 1:…•Example 2:…Constraints:…Please follow the template below to solve the problem:```pythonclass Solution:deffindChampion(self,n: int, edges: List[List[int]]) -> int: ```\nTo solve this problem, we can use a topological sorting algorithm. The idea is to …```pythonfrom typing import Listclass Solution:deffindChampion(self, n: int, edges: List[List[int]]) -> int:# Create an adjacency list to represent the DAGadj\_list= {i: [] for iin range(n)}for u, v in edges:adj\_list[u].append(v)#Initialize a list to keep track of the number of …in\_degree= [0] \* n# Calculate the in-degree for each teamfor u in adj\_list:for v in adj\_list[u]:in\_degree[v] += 1# Initialize a list to keep track of the teams with no incoming edgesno\_incoming\_edges= [ifor iin range(n) if in\_degree[i] == 0]# If there is more than one team with no incoming edges, there is no unique championif len(no\_incoming\_edges) != 1:return -1# Return the team with no incoming edges, which is the championreturn no\_incoming\_edges[0]```This code first creates an adjacency list to represent the DAG. Then, …\nSubmit CodeFigure 6|An example of solving LeetCode Problem.\nB. Benchmark curves during training of DeepSeek-Coder-Base\nIn Figure 7, we present the benchmark curves illustrating the performance of DeepSeek-Coder-\nBase models during their training phase. For validation, a carefully curated subset of the training\ncorpus was employed, consisting of 8,000 code files. This subset was deliberately chosen to\nensure a diverse and representative sample, critical for an accurate assessment of the models’\ncapabilities. The performance metrics of these models are specifically detailed in the final two\nsub-figures of Figure 7, offering a clear visual representation of their efficacy throughout the\ntraining process.\n22', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 21}),

Document(page\_content='0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.10.20.30.40.5ValueHumanEval-Pass@1\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.10.20.30.40.50.6ValueHumanEval-cpp-Pass@1\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.10.20.30.40.50.6ValueHumanEval-java-Pass@1\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.500.550.600.650.700.750.800.85ValueHumanEvalFIM-EM\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.20.40.6ValueMBPP-Pass@1\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.10.20.30.40.5ValueMeanHumanEval\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.10.20.30.4ValueValidation-Completion-EM\n0 250 500 750 1000 1250 1500 1750 2000\nT okens (Billions)0.00.10.20.30.40.5ValueValidation-CompletionFIM-EMDeepSeek-Coder-Base-1.3B DeepSeek-Coder-Base-6.7B DeepSeek-Coder-Base-33BFigure 7|Benchmark curves during training of DeepSeek-Coder-Base.\n23', metadata={'source': 'papers/rachelgreecv.pdf', 'page': 22})]