```
from langchain_google_genai import GoogleGenerativeAIEmbeddings
```

```
c:\Users\hello\AppData\Local\Programs\Python\Python312\Lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please u
  from .autonotebook import tqdm as notebook_tqdm
```

```
from langchain_community.document_loaders import PyPDFLoader
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

```
from langchain.vectorstores import Pinecone
```

```
from dotenv import load_dotenv
```

```
from langchain_pinecone import PineconeVectorStore
```

```
load_dotenv()
```

```
True
```

```
loader=PyPDFLoader("C:/Users/hello/Downloads/yolov9_paper.pdf")
```

```
data=loader.load()
```

```
len(data)
```

```
18
```

```
data
```

[Document(metadata={'source': 'C:/Users/hello/Downloads/yolov9_paper.pdf', 'page': 0}, page_content='YOLOv9: Learning What You Want to Learn\nUsing Programmable Gradient Information\nChien-Yao Wang1,2, I-Hau Yeh2, and Hong-Yuan Mark Liao1,2,3\n1Institute of Information Science, Academia Sinica, Taiwan\n2National Taipei University of Technology, Taiwan\n3Department of Information and Computer Engineering, Chung Yuan Christian University, Taiwan\nkinyiu@iis.sinica.edu.tw, ihyeh@emc.com.tw, and liao@iis.sinica.edu.tw\nAbstract\nToday’s deep learning methods focus on how to design\nthe most appropriate objective functions so that the pre-\ndiction results of the model can be closest to the ground\ntruth. Meanwhile, an appropriate architecture that can\nfacilitate acquisition of enough information for prediction\nhas to be designed. Existing methods ignore a fact that\nwhen input data undergoes layer-by-layer feature extrac-\ntion and spatial transformation, large amount of informa-\ntion will be lost. This paper will delve into the important is-\nsues of data loss when data is transmitted through deep net-\nworks, namely information bottleneck and reversible func-\ntions. We proposed the concept of programmable gradi-\nent information (PGI) to cope with the various changes\nrequired by deep networks to achieve multiple objectives.\nPGI can provide complete input information for the tar-\nget task to calculate objective function, so that reliable\ngradient information can be obtained to update network\nweights. In addition, a new lightweight network architec-\nture – Generalized Efficient Layer Aggregation Network\n(GELAN), based on gradient path planning is designed.\nGELAN’s architecture confirms that PGI has gained su-\nperior results on lightweight models. We verified the pro-\nposed GELAN and PGI on MS COCO dataset based ob-\nject detection. The results show that GELAN only uses\nconventional convolution operators to achieve better pa-\nrameter utilization than the state-of-the-art methods devel-\noped based on depth-wise convolution. PGI can be used\nfor variety of models from lightweight to large. It can be\nused to obtain complete information, so that train-from-\nscratch models can achieve better results than state-of-the-\nart models pre-trained using large datasets, the compari-\nson results are shown in Figure 1. The source codes are at:\nhttps://github.com/WongKinYiu/yolov9.\n1. Introduction\nDeep learning-based models have demonstrated far bet-\nter performance than past artificial intelligence systems in\nvarious fields, such as computer vision, language process-\ning, and speech recognition. In recent years, researchers\nFigure 1. Comparisons of the real-time object detecors on MS\nCOCO dataset. The GELAN and PGI-based object detection\nmethod surpassed all previous train-from-scratch methods in terms\nof object detection performance. In terms of accuracy, the new\nmethod outperforms RT DETR [43] pre-trained with a large\ndataset, and it also outperforms depth-wise convolution-based de-\nsign YOLO MS [7] in terms of parameters utilization.\nin the field of deep learning have mainly focused on how\nto develop more powerful system architectures and learn-\ning methods, such as CNNs [21–23, 42, 55, 71, 72], Trans-\nformers [8, 9, 40, 41, 60, 69, 70], Perceivers [26, 26, 32, 52,\n56, 81, 81], and Mambas [17, 38, 80]. In addition, some\nresearchers have tried to develop more general objective\nfunctions, such as loss function [5, 45, 46, 50, 77, 78], la-\nbel assignment [10, 12, 33, 67, 79] and auxiliary supervi-\nsion [18, 20, 24, 28, 29, 51, 54, 68, 76]. The above studies\nall try to precisely find the mapping between input and tar-\nget tasks. However, most past approaches have ignored that\ninput data may have a non-negligible amount of informa-\ntion loss during the feedforward process. This loss of in-\nformation can lead to biased gradient flows, which are sub-\nsequently used to update the model. The above problems\ncan result in deep networks to establish incorrect associa-\ntions between targets and inputs, causing the trained model\nto produce incorrect predictions.\n1\narXiv:2402.13616v2 [cs.CV] 29 Feb 2024'),
 Document(metadata={'source': 'C:/Users/hello/Downloads/yolov9_paper.pdf', 'page': 1}, page_content='Figure 2. Visualization results of random initial weight output feature maps for different network architectures: (a) input image, (b)\nPlainNet, (c) ResNet, (d) CSPNet, and (e) proposed GELAN. From the figure, we can see that in different architectures, the information\nprovided to the objective function to calculate the loss is lost to varying degrees, and our architecture can retain the most complete\ninformation and provide the most reliable gradient information for calculating the objective function.\nIn deep networks, the phenomenon of input data losing\ninformation during the feedforward process is commonly\nknown as information bottleneck [59], and its schematic di-\nagram is as shown in Figure 2. At present, the main meth-\nods that can alleviate this phenomenon are as follows: (1)\nThe use of reversible architectures [3, 16, 19]: this method\nmainly uses repeated input data and maintains the informa-\ntion of the input data in an explicit way; (2) The use of\nmasked modeling [1, 6, 9, 27, 71, 73]: it mainly uses recon-\nstruction loss and adopts an implicit way to maximize the\nextracted features and retain the input information; and (3)\nIntroduction of the deep supervision concept [28,51,54,68]:\nit uses shallow features that have not lost too much impor-\ntant information to pre-establish a mapping from features\nto targets to ensure that important information can be trans-\nferred to deeper layers. However, the above methods have\ndifferent drawbacks in the training process and inference\nprocess. For example, a reversible architecture requires ad-\nditional layers to combine repeatedly fed input data, which\nwill significantly increase the inference cost. In addition,\nsince the input data layer to

```
the output layer cannot have a\ntoo deep path, this limitation will make it difficult to model\nhigh-order semantic information
during the training pro-\ncess. As for masked modeling, its reconstruction loss some-\ntimes conflicts with the target loss. In
addition, most mask\nmechanisms also produce incorrect associations with data.\nFor the deep supervision mechanism, it will
produce error\naccumulation, and if the shallow supervision loses informa-\ntion during the training process, the subsequent
layers will\nnot be able to retrieve the required information. The above\nphenomenon will be more significant on difficult tasks
and\nsmall models.\nTo address the above-mentioned issues, we propose a\nnew concept, which is programmable gradient
information\n(PGI). The concept is to generate reliable gradients through\nauxiliary reversible branch, so that the deep
features can\nstill maintain key characteristics for executing target task.\nThe design of auxiliary reversible branch can avoid
the se-\nmantic loss that may be caused by a traditional deep super-\nvision process that integrates multi-path features. In
```

```
text_splitter=RecursiveCharacterTextSplitter(chunk_size=1000)
```

```
docs=text_splitter.split_documents(data)
```

```
print("TOTAL NUM OF DOCS",len(docs))
```

```
TOTAL NUM OF DOCS 96
```

```
docs[90]
```

```
Document(metadata={'source': 'C:/Users/hello/Downloads/yolov9_paper.pdf', 'page': 16}, page_content='RT DETR-R101 [43] (I) 76 259
54.3 72.7 58.6 36.0 58.8 72.1\nRTMDet-X [44] (I) 94.9 283.4 52.8 70.4 – – – –\nYOLOR-CSP-X [66] (C) 96.9 226.8 54.8 73.1 59.7 – – –
\nPPYOLOE+-X [74] (C) 98.4 206.6 54.7 72.0 59.9 37.9 59.3 70.4\nPPYOLOE-X [74] (I) 98.4 206.6 52.3 69.5 56.8 35.1 57.0 68.6\n1 (S),
(I), (D), (C) indicate train-from-scratch, ImageNet pretrained, knowledge distillation, and complex setting, respectively.\nTable 4
shows the performance of all models sorted by\nparameter size. Our proposed YOLOv9 is Pareto optimal\nin all models of different
sizes. Among them, we found no\nother method for Pareto optimal in models with more than\n20M parameters. The above experimental
data shows that\nour YOLOv9 has excellent parameter usage efficiency.\nShown in Table 5 is the performance of all participat-\ning
models sorted by the amount of computation. Our pro-\nposed YOLOv9 is Pareto optimal in all models with differ-\nent scales. Among
models with more than 60 GFLOPs, only')
```

```
import os
```

```
google_api_key = os.getenv("GOOGLE_API_KEY")

# Use the API key
print("Google API Key:", google_api_key)
```

```
Google API Key: AIzaSyAHf55HW8rRnWegYrgq9Og_lbGclbPX6mw
```

```
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001", api_key=google_api_key)
print(embeddings)
print("Embeddings initialized successfully!")
```

```
client=<google.ai.generativelanguage_v1beta.services.generative_service.client.GenerativeServiceClient object at 0x000001C8513D0380>
Embeddings initialized successfully!
```

```
vector=embeddings.embed_query("hello how are you")
vector
len(vector)
```

```
768
```

```
len(vector)
```

```
768
```

```
PINECONE_INDEX_NAME="firstproject"
os.environ['PINECONE_API_KEY']="pcsk_4u81nL_31P8DCTTWg8sh2o3QF4KPb54gjac1CHmaYBJWXWAFaKFjtuwNd4sTNfwckkYE8j"
```

```
docearch=Pinecone.from_existing_index(index_name=PINECONE_INDEX_NAME,embedding=embeddings)
print("index succeesfully created!")
```

```
index succeesfully created!
```

```
vectorstore_from_docs=PineconeVectorStore.from_documents(
    docs,
    index_name=PINECONE_INDEX_NAME,
    embedding=embeddings
)
```

```
docsearch=PineconeVectorStore.from_existing_index(PINECONE_INDEX_NAME,embeddings)
query="what is a parameter"
docs=docearch.similarity_search(query,k=3)
print(docs)
```

```
[Document(metadata={'page': 1.0, 'source': 'C:/Users/hello/Downloads/yolov9_paper.pdf'}, page_content='ditional layers to combine re
```

```
retriever=docsearch.as_retriever(search_type="similarity",search_kwargs={"k":10})
retrieved_docs=retriever.invoke("what is yolvo9 parameter?")
```

```
from langchain_google_genai import ChatGoogleGenerativeAI
llm=ChatGoogleGenerativeAI(model="gemini-1.5-pro",temparature=0.3,max_tokens=500)
```

```
print("Language model initialized successfully!")
```

```
Language model initialized successfully!
```

```
query = "Explain Yolov9 parameters in detail based on the documents."
retrieved_texts = [doc.page_content for doc in retrieved_docs]
```

```
context = "\n\n".join(retrieved_texts)
```

```
prompt = f"Based on the following information:\n\n{context}\n\nAnswer the question:\n{query}"
```

```
response = llm.generate(prompt)
print("LLM Response:")
print(response)
```

Start coding or generate with AI.