**1. Core Project: Enhanced Multi-Label Topic and Sub-Topic Classification**

This project directly addresses and expands your initial problem statement, aiming for the most accurate and fine-grained classification possible.

**Key Components**

* **Fine-Grained Classification:** Instead of just broad topic classification (e.g., 'Crop'), classify queries into both a **main topic** (e.g., *Pest Management, Fertilizer Use, Market Price, Government Scheme*) and a **sub-topic** (e.g., *Pest Management - Cotton Bollworm*, *Fertilizer Use - Nitrogen Deficiency*). This will likely be a **multi-label classification** task if a query can belong to more than one category.
* **Model Architecture:** Implement a **BERT-based classifier** with a custom top layer.
  + **BERT for Embeddings:** Use a pre-trained BERT model (or a domain-specific variant like **AgriBERT** if available/trainable) to generate high-quality contextual embeddings for the query text.
  + **Attention Mechanism Integration:** Add a **Self-Attention** layer or a simpler **Weighted-Attention** layer on top of BERT's output (specifically the [CLS] token's embedding) before the final classification layers. This mechanism can help the model focus on the most critical words in the query that determine its topic.
* **Evaluation and Analysis:**
  + Compare performance metrics (Accuracy, Precision, Recall, F1-score) of the BERT-Attention model against a simpler BERT-only model and a traditional machine learning baseline (e.g., Naive Bayes or SVM with TF-IDF features).
  + Generate **attention heatmaps** for a subset of queries to visualize which words the model focused on when making a decision. This provides valuable interpretability.

# In Easy Words

That's a great request! Focusing on the essence of each project will make it easier to choose and explain.

Here is a simplified explanation of the three Deep Learning project suggestions for the Kisan Call Centre data:

**1. Core Project: Enhanced Topic Classification**

**The Goal:**

To build a super-smart sorting machine that can read a farmer's query and instantly label *exactly* what the problem is (e.g., "Cotton pest" vs. "Wheat fertilizer").

**Analogy:**

Imagine you have a big pile of mail. This project is like having a person who not only sorts it into big bins ("Bills," "Junk Mail," "Magazines") but also puts sticky notes on each one for the sub-category ("Bills - Electricity," "Bills - Phone").

**Key Deep Learning Idea: BERT and Attention**

* **BERT:** The "query reader." It understands the meaning and context of the words, even if the farmer uses a local term or phrased the question oddly.
* **Attention:** The "focus mechanism." It tells BERT to pay extra close attention to the most important words in the query (e.g., if the query says "cotton leaf curl and spray," the attention focuses on "**cotton**" and "**leaf curl**" and ignores words like "and" and "the").

**Your Output:**

A model that is highly accurate at giving a farmer's query a precise, two-part label (e.g., **Topic: Pest Management, Sub-Topic: Mites**).

That's an excellent choice! **Google Colab** is the ideal environment for this project because it provides **free access to GPUs**, which are essential for fine-tuning large models like BERT.

Here is a refined, step-by-step implementation guide focused on using Google Colab's capabilities and libraries like **Hugging Face Transformers** and **PyTorch** (the most common framework in NLP research).

# Colab Implementation Steps for Project 1

Your project focuses on building a custom deep learning model. The steps below detail the coding and infrastructure tasks within a Colab environment.

**💻 Phase 1: Colab Setup & Data Import**

The first few cells in your notebook should focus here.

| Colab Task | Code/Action | Purpose |
| --- | --- | --- |
| **1.1 Enable GPU** | Go to **Runtime Change runtime type** and select **GPU** as the hardware accelerator. | Provides the necessary power for BERT fine-tuning (BERT is very slow on a CPU). |
| **1.2 Install Libraries** | Install Hugging Face Transformers and other required tools. | !pip install transformers torch scikit-learn pandas matplotlib |
| **1.3 Import Data** | Mount Google Drive or upload your 3 months of KCC data (.csv or .xlsx) to the Colab environment. | from google.colab import drive / drive.mount('/content/drive') then load data with pd.read\_csv(...). |
| **1.4 Data Preprocessing** | Clean query text and create your multi-label target vectors. | Use **pandas** to filter and clean text, and sklearn.preprocessing.MultiLabelBinarizer to encode your topic/sub-topic columns into the binary format (your labels). |

**🛠️ Phase 2: Building the BERT + Attention Architecture**

You will create a custom PyTorch model class to integrate your attention layer.

| Colab Task | Code/Action | Purpose |
| --- | --- | --- |
| **2.1 Load Tokenizer & Model** | Choose a multilingual model like bert-base-multilingual-cased (**mBERT**) or xlm-roberta-base. | tokenizer = AutoTokenizer.from\_pretrained(MODEL\_NAME) |
| **2.2 Define Custom Model Class** | Create a Python class that inherits from torch.nn.Module. | This class will contain the pre-trained BERT body and your new custom layers. |
| **2.3 Integrate Attention Layer** | You will add a **Weighted Attention Layer** on top of BERT's **sequence output**. | **Code Snippet Idea:** |
|  |  | 1. Get sequence\_output from BERT (vectors for every token). |
|  |  | 2. Apply a linear layer + tanh to get "attention energy." |
|  |  | 3. Apply softmax to normalize the energy into **attention weights** (vector ). |
|  |  | 4. Compute the final document representation (vector ) as a **weighted sum** of the sequence output: . |
| **2.4 Final Classification Layer** | Add the output layer to get your multi-label prediction. | nn.Linear(BERT\_hidden\_size, num\_labels) |

**🚀 Phase 3: Training and Fine-Tuning**

This is where you run the optimization process on the Colab GPU.

| Colab Task | Code/Action | Purpose |
| --- | --- | --- |
| **3.1 Define Optimizer & Scheduler** | Use the standard AdamW optimizer and a learning rate scheduler. | AdamW is the recommended optimizer for transformer fine-tuning. The scheduler helps prevent initial gradient explosions. |
| **3.2 Run Training Loop** | Iterate through your data, performing forward and backward passes. | Your loop will calculate the **BCEWithLogitsLoss**, perform backpropagation (loss.backward()), and update weights (optimizer.step()). |
| **3.3 Save the Best Model** | Use a callback to save the model weights only when performance on the **Validation Set** improves. | This ensures you have the best version of your model, which you will load for final testing. |

**📊 Phase 4: Analysis and Deliverables**

The last phase generates the required project documentation and proofs.

| Colab Task | Code/Action | Purpose |
| --- | --- | --- |
| **4.1 Test & Metric Calculation** | Load the best model and evaluate it on the untouched **Test Set**. | Calculate and report **Macro F1-score**, **Precision**, and **Recall**. |
| **4.2 Benchmark Plot** | Plot your final model's performance against your baseline models (e.g., Simple LSTM) to show the architectural advantage. | Use **Matplotlib** to generate a simple bar chart. |
| **4.3 Attention Heatmaps (The Interpretability Proof)** | This is your most important visual proof. You need to capture the attention weights from the layer you built (or the internal BERT layers). | Use **Matplotlib** and **Seaborn** to create a heatmap for 3-5 sample queries, showing which words drove the classification decision. For example: |
|  |  | **Query:** "My cotton is facing **whitefly** problem, what to spray?" |
|  |  | **Heatmap Output:** High attention weights on **cotton** and **whitefly**. |