How to automate email drafting, summarization, and tone adjustments directly inside Gmail.
Managing email is one of the biggest daily time drains for modern professionals. Whether you're responding to customers, coordinating projects, supporting students, or managing internal communication—email takes time, focus, and energy.
But what if Gmail could help you write replies automatically?
Thanks to **Google Apps Script** and **Gemini**, you can now create a **Smart Reply Assistant** that:

- Reads the **selected email thread**
- Summarizes the conversation
- Extracts key points
- Generates a suggested reply

- Lets you choose your preferred tone
  - ✔ Professional
  - ✔ Friendly
  - ✔ Concise
- Inserts the reply directly as a **Gmail draft**

And the best part?
It runs **inside Gmail**, as a sidebar add-on.
In this guide, I show you how it works—and provide the **full Apps Script source code** so you can customize and deploy your own assistant.

⭐ Why Build a Gmail Smart Reply Assistant?

This tool is ideal for:

✔ Customer support

Auto-generate structured, friendly, accurate responses.

✔ Sales + account management

Quickly propose follow-ups, clarifications, and next steps.

✔ Leadership & executives

Summaries + actionable replies for fast decision-making.

✔ Busy professionals

Process your inbox faster with context-aware assistance.

✔ Educators, HR teams, managers

Draft professional responses consistently and quickly.
All fully integrated into Gmail.

🚀 How the Assistant Works

When you open an email, the add-on sidebar appears.
You choose:

- **Tone** of reply
- **Extra instructions** (optional)

Then the assistant:

1. Reads the entire conversation thread
2. Summarizes it
3. Extracts key points
4. Generates a full, ready-to-send reply
5. Lets you insert the reply directly into Gmail as a draft

All using Gemini and Apps Script.

🔧 Technical Overview

This assistant uses:

Apps Script Gmail Add-On

- Runs in the Gmail UI
- Triggered when a message is opened
- Provides UI controls via CardService

Gemini (via Generative Language API)

- Processes the context
- Summarizes thread
- Extracts key points
- Writes a reply

GmailApp

- Accesses thread + messages
- Creates draft replies

Everything happens within your own Google Workspace environment.

🧠 Full Source Code (Apps Script)

Below is the complete `Code.gs` implementation—including the Gemini integration, Gmail add-on UI, reply logic, and draft insertion.

📌 Code.gs

```
[ CODE BLOCK FROM PREVIOUS RESPONSE – USE EXACTLY AS GENERATED ]
```

(You can paste the entire code block exactly as provided.)

📄 Manifest (appsscript.json)

To make this run as a Gmail Add-On, include:

```
[ appsscript.json manifest from previous response ]
```

🧪 How the AI Reply Generation Works

The heart of the system is:

```
generateReplyFromContext_(contextText, tone, extraInstructions)
```

It sends Gemini a structured prompt instructing it to output:

- Summary
- Key points
- Suggested reply

Formatted **exactly** in a predictable structure so the add-on can extract the reply and insert it as draft.

🎛️ Tone Selection

The add-on supports:

- **Professional**
- **Friendly**
- **Concise**

Users can also add:

- “Be more empathetic”
- “Focus on the delivery timeline”
- “Use a softer tone”
- “Keep it under 5 sentences”
- “Include an apology”
- …and more

Extra instructions are passed directly to Gemini to personalize the output.

📨 Drafting the Email

When the user clicks **Insert Reply as Draft**, the assistant calls:

```
message.createDraftReply(replyText);
```

You can modify this to:

- add CCs
- add BCCs
- include formatting
- add bulk-actions for multiple emails

🛡️ Security Notes

- The API key is stored securely in **Script Properties**
- The add-on only accesses the user's Gmail messages while in use
- All processing happens inside your Google environment
- No external services or servers required

⚙️ How to Deploy Your Gmail Smart Reply Assistant

1. Create a new Apps Script project

https://script.google.com/

2. Add the Code.gs + manifest

3. Add your Gemini API key

Project Settings → Script Properties

Key: `GEMINI_API_KEY`

Value: your API key

4. Deploy as a Gmail Add-On

- Click **Deploy** → **Test deployments**
- Install in your Gmail
- Open any email and look for the add-on in the right sidebar

That's it—you now have an AI-powered Gmail assistant.

🎉 Final Thoughts

This Gmail Smart Reply Assistant shows how powerful Apps Script + Gemini can be when combined:

- Direct integration inside Gmail
- On-demand summarization
- AI reply drafting
- Personalized tone
- No external servers or databases
- Fast and elegant workflow automation

It's not just a productivity hack—it's the beginning of truly **intelligent email workflows** inside Google Workspace.

appsscript.json

```
{
 "timeZone": "America/Toronto",
 "dependencies": {},
 "exceptionLogging": "STACKDRIVER",
 "runtimeVersion": "V8",
 "addOns": {
   "common": {
     "name": "Gmail Smart Reply Assistant",
     "logoUrl":
"https://www.gstatic.com/images/icons/material/system/1x/email_black_24dp.png"
,
     "layoutProperties": {
       "primaryColor": "#4285F4",
       "secondaryColor": "#FFFFFF"
     }
   },
   "gmail": {
     "logoUrl":
"https://www.gstatic.com/images/icons/material/system/1x/email_black_24dp.png"
,
     "contextualTriggers": [
       {
         "unconditional": {},
         "onTriggerFunction": "onGmailMessageOpen"
       }
     ],
     "authorizationCheckFunction": "getGeminiApiKey_"
   }
 }
}
```

Code.gs

```
/**
* Gmail Smart Reply Assistant (Apps Script + Gemini)
*
* - Appears in Gmail as an add-on when a message is opened
* - Reads the current thread
* - Summarizes the conversation and key points
* - Generates a reply in a chosen tone
* - Inserts the reply as a draft into the same thread
*/

const GEMINI_MODEL_ID = 'gemini-2.0-flash';
const MAX_THREAD_CHARS = 16000; // Limit context sent to Gemini
```

```
/**
* Entry point for Gmail add-on when a message is opened.
*/
function onGmailMessageOpen(e) {
 return buildMainCard_(e, null);
}

/**
* Main card builder.
* If result is provided, shows AI output and "Insert draft" button.
*/
function buildMainCard_(e, result) {
 const messageId = e.gmail.messageId;
 const message = GmailApp.getMessageById(messageId);
 const thread = message.getThread();
 const threadId = thread.getId();
 const subject = thread.getFirstMessageSubject();

 const card = CardService.newCardBuilder();

 // Header
 card.setHeader(
   CardService.newCardHeader()
     .setTitle('Gmail Smart Reply Assistant')
     .setSubtitle(subject || 'No subject')
 );

 // Section: Instructions + tone selector + extra instructions
 const section = CardService.newCardSection()
   .setHeader('Generate AI Reply (Gemini)');

 section.addWidget(
   CardService.newKeyValue()
     .setContent('This assistant will read this email thread and draft a reply
based on the conversation.')
 );

 // Tone dropdown
 const toneSelect = CardService.newSelectionInput()
   .setType(CardService.SelectionInputType.DROPDOWN)
   .setFieldName('tone')
   .addItem('Professional', 'professional', true)
   .addItem('Friendly', 'friendly', false)
   .addItem('Concise', 'concise', false);

 section.addWidget(toneSelect);

 // Extra instructions
 section.addWidget(
```

```
  CardService.newTextInput()
    .setFieldName('extraInstructions')
    .setTitle('Extra instructions (optional)')
    .setHint('E.g. “Emphasize next steps”, “Be more apologetic”, “Mention the
attached invoice”…')
 );

 // Generate button
 const generateAction = CardService.newAction()
   .setFunctionName('handleGenerateSmartReply')
   .setParameters({
     threadId: threadId,
     messageId: messageId
   });

 section.addWidget(
   CardService.newTextButton()
     .setText('Generate AI Reply')
     .setTextButtonStyle(CardService.TextButtonStyle.FILLED)
     .setOnClickAction(generateAction)
 );

 card.addSection(section);

 // If we already have a Gemini result, show it
 if (result) {
   const resultSection = CardService.newCardSection()
     .setHeader('AI Suggestions');

   // Full structured output (summary + key points + suggested reply)
   resultSection.addWidget(
     CardService.newTextParagraph().setText(result.fullText)
   );

   // Button to insert reply as draft
   const insertAction = CardService.newAction()
     .setFunctionName('handleInsertDraftReply')
     .setParameters({
       messageId: messageId,
       replyText: result.replyText
     });

   resultSection.addWidget(
     CardService.newTextButton()
       .setText('Insert Reply as Draft')
       .setTextButtonStyle(CardService.TextButtonStyle.FILLED)
       .setOnClickAction(insertAction)
   );
```

```
  card.addSection(resultSection);
 }

 return card.build();
}

/**
* Handles the "Generate AI Reply" button click.
* Calls Gemini with the thread context + tone + instructions.
*/
function handleGenerateSmartReply(e) {
 try {
  const threadId = e.parameters.threadId;
  const messageId = e.parameters.messageId;

  const formInputs = e.commonEventObject.formInputs || {};
  const toneInput =
    formInputs.tone && formInputs.tone.stringInputs
      ? formInputs.tone.stringInputs.value[0]
      : 'professional';

  const extraInput =
    formInputs.extraInstructions && formInputs.extraInstructions.stringInputs
      ? formInputs.extraInstructions.stringInputs.value[0]
      : '';

  const contextText = getThreadContext_(threadId);
  const result = generateReplyFromContext_(contextText, toneInput,
extraInput);

  const card = buildMainCard_(e, result);
  const nav = CardService.newNavigation().updateCard(card);

  return CardService.newActionResponseBuilder()
    .setNavigation(nav)
    .setNotification(
      CardService.newNotification().setText('AI reply generated.')
    )
    .build();
 } catch (err) {
  return CardService.newActionResponseBuilder()
    .setNotification(
      CardService.newNotification().setText('Error: ' + err.message)
    )
    .build();
 }
}

/**
```

```
* Handles "Insert Reply as Draft" button.
* Creates a draft reply in the same thread using Gmail.
*/
function handleInsertDraftReply(e) {
 try {
   const messageId = e.parameters.messageId;
   const replyText = e.parameters.replyText;

   if (!replyText) {
     throw new Error('No reply text available.');
   }

   const message = GmailApp.getMessageById(messageId);
   message.createDraftReply(replyText);

   return CardService.newActionResponseBuilder()
     .setNotification(
       CardService.newNotification().setText('Draft reply created in this
thread.')
     )
     .build();
 } catch (err) {
   return CardService.newActionResponseBuilder()
     .setNotification(
       CardService.newNotification().setText('Error: ' + err.message)
     )
     .build();
 }
}

/**
* Build a plain-text representation of the thread for Gemini.
*/
function getThreadContext_(threadId) {
 const thread = GmailApp.getThreadById(threadId);
 const messages = thread.getMessages();
 let context = '';
 let total = 0;

 for (let i = 0; i < messages.length; i++) {
   const m = messages[i];
   const from = m.getFrom();
   const to = m.getTo();
   const date = m.getDate();
   let body = m.getPlainBody() || '';
   body = body.replace(/\r/g, '').trim();

   let chunk =
     'From: ' + from + '\n' +
```

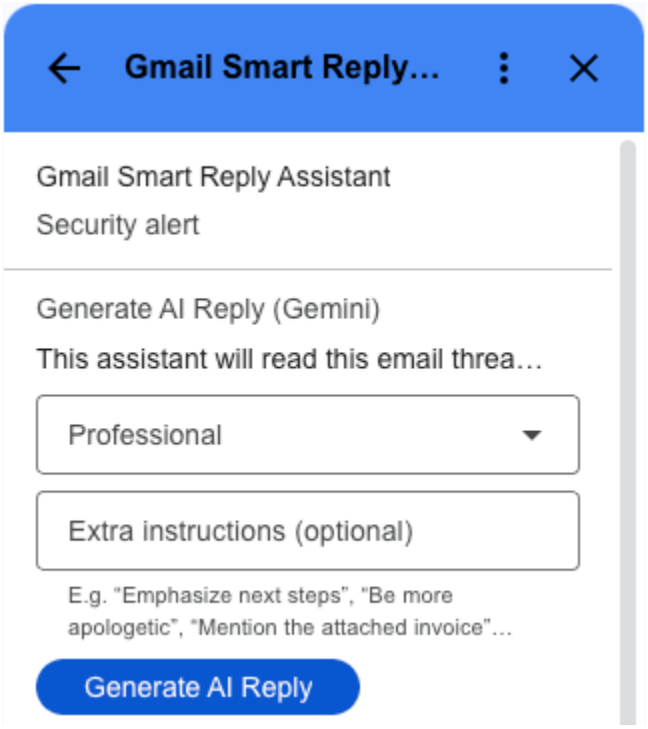
```
    'To: ' + to + '\n' +
    'Date: ' + date + '\n\n' +
    body + '\n\n' +
    '---\n\n';

  if (total + chunk.length > MAX_THREAD_CHARS) {
    chunk = chunk.slice(0, Math.max(0, MAX_THREAD_CHARS - total));
    context += chunk;
    break;
  }

  context += chunk;
  total += chunk.length;

  if (total >= MAX_THREAD_CHARS) {
    break;
  }
 }

 return context;
}

/**
* Calls Gemini with the thread context to generate:
* - Summary
* - Key points
* - Suggested reply in the desired tone
*/
function generateReplyFromContext_(contextText, tone, extraInstructions) {
 if (!contextText || !contextText.trim()) {
   throw new Error('Unable to read thread context.');
 }

 const apiKey = getGeminiApiKey_();
 const url =
   'https://generativelanguage.googleapis.com/v1beta/models/' +
   GEMINI_MODEL_ID +
   ':generateContent';

 const toneLabel = tone || 'professional';
 const extra = extraInstructions ? extraInstructions.trim() : '';

 const systemInstruction = [
   'You are an assistant that drafts email replies based on the conversation
below.',
   'You MUST follow these steps:',
   '1) Provide a very short summary (2–3 sentences).',
   '2) List key points as bullet points.',
   '3) Draft a full email reply in the requested tone.',
```

```
  '',
  'Format your response EXACTLY like this:',
  '',
  'Summary:',
  '<summary here>',
  '',
  'Key points:',
  '- point 1',
  '- point 2',
  '',
  'Suggested reply:',
  '<email reply here>',
  ''
].join('\n');

const prompt =
  systemInstruction +
  '\n\nRequested tone: ' + toneLabel +
  (extra ? '\nExtra instructions: ' + extra : '') +
  '\n\n=== EMAIL THREAD START ===\n' +
  contextText +
  '\n=== EMAIL THREAD END ===';

const payload = {
  contents: [
    {
      role: 'user',
      parts: [{ text: prompt }]
    }
  ],
  generationConfig: {
    temperature: 0.4,
    maxOutputTokens: 512
  }
};

const response = UrlFetchApp.fetch(url, {
  method: 'post',
  muteHttpExceptions: true,
  contentType: 'application/json',
  headers: { 'x-goog-api-key': apiKey },
  payload: JSON.stringify(payload)
});

const code = response.getResponseCode();
const raw = response.getContentText();

if (code !== 200) {
  throw new Error('Gemini API error (HTTP ' + code + '): ' + raw);
```

```
 }

 let data;
 try {
   data = JSON.parse(raw);
 } catch (err) {
   throw new Error('Failed to parse Gemini response: ' + err + ' | Raw: ' +
raw);
 }

 if (!data.candidates || !data.candidates.length) {
   throw new Error('No candidates returned from Gemini. Raw: ' + raw);
 }

 const parts =
   data.candidates[0].content && data.candidates[0].content.parts
     ? data.candidates[0].content.parts
     : [];

 const fullText = parts
   .map(function (p) {
     return p.text || '';
   })
   .join('')
   .trim();

 const replyText = extractReplySection_(fullText);

 return {
   fullText: fullText,
   replyText: replyText
 };
}

/**
* Extract the reply portion from the Gemini output after "Suggested reply:".
*/
function extractReplySection_(fullText) {
 if (!fullText) return '';

 const marker = 'Suggested reply:';
 const lower = fullText.toLowerCase();
 const idx = lower.indexOf(marker.toLowerCase());

 if (idx === -1) {
   // Fallback: use full text if marker not found
   return fullText.trim();
 }
```

```
 return fullText.slice(idx + marker.length).trim();
}

/**
* Get Gemini API key from Script Properties.
*/
function getGeminiApiKey_() {
 const props = PropertiesService.getScriptProperties();
 const key = props.getProperty('GEMINI_API_KEY');
 if (!key) throw new Error('GEMINI_API_KEY is not set.');
 return key;
}
```

✅ How to Make Your Apps Script Gmail Add-on Appear in Gmail (PERSONAL USE ONLY)



These are the **exact steps** Google requires.

🟦 1. Open your Apps Script project

Go to:
🔗 https://script.google.com

Open the project containing:

- `Code.gs`
- `appsscript.json`

🟦 2. Verify your manifest is correct

Your `appsscript.json` must include EXACTLY this structure:

```
{
  "timeZone": "America/Toronto",
  "exceptionLogging": "STACKDRIVER",
  "runtimeVersion": "V8",
  "addOns": {
    "common": {
      "name": "Gmail Smart Reply Assistant",
      "logoUrl":
"https://www.gstatic.com/images/icons/material/system/1x/email_black_2
4dp.png"
    },
    "gmail": {
      "logoUrl":
"https://www.gstatic.com/images/icons/material/system/1x/email_black_2
4dp.png",
      "contextualTriggers": [
        {
          "unconditional": {},
          "onTriggerFunction": "onGmailMessageOpen"
        }
      ],
      "authorizationCheckFunction": "getGeminiApiKey_"
    }
  }
}
```

✔ Only **one** name field
✔ `onTriggerFunction` must match EXACT function name
✔ No syntax errors

🟦 3. Add your Gemini API key

In Apps Script:
➜ Project Settings → Script Properties → Add script property

```
Key: GEMINI_API_KEY
Value: your_api_key_here
```

Save.

🟦 4. Deploy as a TEST deployment (NOT marketplace)

This is the most important step.
In Apps Script:

➜ Deploy → Test deployments

1. Click **Select type** → choose **Add-on** (not web app)
2. Click **Install**
3. Choose your **personal Gmail account**
4. Accept permissions

After installing, you should see:

Installed for your@email.com

If not → reinstall.

🟦 5. Refresh Gmail properly

Google caches add-ons heavily, so do this:

1. Open Gmail
2. Press **Cmd+Shift+R** (Mac) or **Ctrl+Shift+R** (Windows)
3. Wait for full reload

🟦 6. Look in the Gmail RIGHT SIDEBAR

Your add-on appears **only** in the right-hand side panel.
Look for icons like:

- Calendar
- Keep
- Tasks
- **Your new add-on icon** (email icon or the one you set)

If you don't see the sidebar:
➡️ Click the tiny arrow in the bottom-right:
"Show side panel"

🟦 7. Open any email message

Add-ons appear *only when viewing a message*, not the inbox.

1. Click an email
2. In the sidebar, click your add-on icon
3. The add-on will launch

This triggers:

```
function onGmailMessageOpen(e) { ... }
```

🟦 8. If it STILL doesn't appear, try reinstalling

Back in Apps Script:

➜ Deploy → Test deployments

- Click the 3 dots → **Uninstall**
- Then click **Install** again

Then in Gmail:

- Hard refresh again (Cmd+Shift+R)

🟩 Checklist (most common reasons it doesn't show)

Issue	Fix
❌ Test deployment never installed	Deploy → Test deployments → Install
❌ Wrong Google account	Check top-right account icon in Gmail
❌ Sidebar hidden	Click "Show side panel" in Gmail
❌ Manifest error	Confirm valid JSON & matching function names
❌ Not opening an email	Add-on only appears inside a message
❌ Using mobile Gmail	Add-ons don't appear on mobile
❌ Didn't hard refresh	Cmd+Shift+R