2toLEAD

Government AI Agent Playbook

Crafting Conversations with Agents

WRITTEN BY | Kanwal Khipple & Richard Plantt

E-BOOK

Love the way you work. Together.

TABLE OF CONTENTS

Introduction to Copilot Agents

What are Copilot Agents?

A Copilot Agent is an AI-powered digital assistant designed to help users' complete tasks, solve problems, and generate content through natural language interaction. Unlike traditional chat agents, Copilot Agents are conversational, goal-oriented, and context-aware, making them highly effective for a wide range of applications.

They leverage advanced language models to understand user intent, maintain coherent dialogue, and perform complex, multi-step tasks. These agents are often multimodal, capable of working with text, images, code, and more. These agents are typically integrated with tools, APIs, or internal systems to extend their functionality beyond simple conversation.

Real-world examples include Microsoft 365 Copilot, which assists with productivity tasks in Word, Excel, and Outlook; GitHub Copilot, which helps developers write and understand code; and custom-built Copilot Agents tailored for specific domains like customer support, education, or healthcare.

Building your own Copilot Agent allows you to tailor its capabilities to your unique workflows, integrate it with proprietary tools or data, and enhance productivity, creativity, and decision-making within your environment.

How Do Agents Differ from Traditional Chat Agents?

- **Smarter:** Uses advanced AI models (like GPT-4 or later) for deeper understanding and reasoning.
- **More Capable:** Can manage complex workflows, multi-step tasks, and dynamic user needs.
- **Context-Aware:** Maintains memory or session context to provide coherent, personalized assistance.

Why Build Your Own Agents?

- Tailor the agent to your unique workflows or business needs.
- Integrate with proprietary tools or data.
- Enhance productivity, creativity, and decision-making in your environment.

Privacy & Security

Copilot doesn't learn from individual user interactions in the way humans do. Instead, it relies on a vast amount of pre-existing data and continuous updates from its developers to improve its responses. This data includes books, articles, websites, and other text sources from trusted sources, ensuring the information provided is accurate and relevant.

While Copilot can remember the context of the current conversation to maintain coherence, it doesn't store personal data or learn from specific user interactions, ensuring your privacy and security. Additionally, users have the option to delete their Microsoft Copilot interaction history, which includes their prompts and the responses Copilot returns.

Integration with Organizational Data

Microsoft 365 Copilot's ability to leverage organizational data is a key differentiator. It integrates with various data sources within Microsoft 365, such as emails, Teams messages, SharePoint documents, and more. This integration allows Copilot to provide responses that are not only based on the user's prompt but also grounded in the data the user has access to within the organization. This ensures that the responses are relevant and tailored to the user's specific context.

For example, Copilot can:

- Summarize emails and documents from SharePoint and OneDrive.
- Retrieve information from Teams chats and meetings.
- Access calendar events to provide scheduling assistance.
- Utilize data from Dynamics 365 and other enterprise systems through Microsoft Graph connectors.

Enhancing Copilot with External Data

Copilot can also be extended to integrate with external data sources using Copilot connectors. These connectors allow organizations to bring in data from various systems, enhancing the AI-driven experience. For instance, you can use Microsoft Graph connectors to integrate data from non-Microsoft sources, providing a more comprehensive view and enabling more informed decision-making.

AI Use Cases for Government

The Role of AI in Public Relations and Government

Artificial intelligence (AI) agents integrated with Microsoft 365 are transforming how government agencies and public sector organizations deliver services and manage workflows. Across government operations, from citizen engagement to internal knowledge management, AI-powered “copilots” are accelerating processes, improving accuracy, and allowing staff to focus on higher-value tasks.

During the COVID-19 pandemic, for instance, nearly 75% of U.S. states deployed chatbots to handle surging public inquiries, proving the value of AI in scaling citizen communications.

Opportunities to Impact Government

Now, governments are expanding these capabilities into everyday use. This playbook outlines five key AI agent use cases for Government (and related public service functions), each presented in a consistent, easy-to-scan format. We focus on solutions built with Microsoft 365 native tools, such as Teams, SharePoint, Power Automate, Power Apps, Copilot Studio, Outlook, and Dataverse, ensuring these use cases are achievable for business users and “citizen developers” with full Microsoft 365 and Power Platform access.

Information Discovery & Summarization: Quickly surface relevant legislation, policy documents, or historical records.

Streamlined Public Communication: Draft press releases, social media posts, and public statements with tone and audience awareness.

Real-Time Sentiment & Feedback Analysis: Analyze public sentiment from social media, surveys, or emails.

Policy Development & Impact Simulation: Model potential outcomes of proposed policies using historical data and stakeholder input.

Automated Reporting & Compliance: Generate reports for internal audits, public disclosures, or regulatory compliance.

Relationship Management: Track interactions, generate personalized follow-ups, and manage outreach campaigns.

Public Relation Roles Using AI

Communications Manager

Description: Crafts press releases, manages media relations, and monitors public sentiment.

AI-Enhanced Productivity:

- Copilot can draft content, analyze media coverage, and suggest messaging strategies based on sentiment analysis.

Social Media Manager

Description: Handles digital engagement and brand presence.

AI-Enhanced Productivity:

- Agents can schedule posts, respond to comments using tone-aware templates, and generate performance analytics.

Crisis Communications Lead

Description: Manages communication during emergencies or sensitive situations.

AI-Enhanced Productivity:

- Copilot can help draft rapid-response statements, simulate public reaction scenarios, and coordinate internal updates.

Policy Analyst

Description: Uses Copilot to summarize legislation, compare policy proposals, and generate briefing notes.

AI-Enhanced Productivity:

- Agents can also track stakeholder sentiment and synthesize public feedback from surveys or social media.

Legislative Assistant

Description: Supports elected officials by drafting communications, preparing talking points, and managing schedules.

AI-Enhanced Productivity:

- Copilot can automate calendar coordination, generate constituent response templates, and surface relevant legislative updates.

Public Service Manager

Description: Oversees service delivery and internal operations.

AI-Enhanced Productivity:

- Agents can help with performance reporting, internal communications, and streamlining workflows across departments.

Data Governance Lead

Description: Ensures responsible data use and compliance.

AI-Enhanced Productivity:

- Agents can monitor data access patterns, flag anomalies, and generate reports aligned with privacy regulations.

Agent: Constituent Self-Service Chatbot

Agent Name: City Assist

Agent Description

A conversational AI chatbot that provides constituents with self-service support around the clock. Its purpose is to handle common citizen questions about government services, FAQs, and simple requests through natural language dialogue.

This frees up human staff to focus on complex or edge cases, while citizens get instant, consistent answers at any time. For example, during the pandemic many governments deployed chatbots to manage the surge in public inquiries; these bots answered FAQs on health guidelines and unemployment benefits 24/7, performing the work of multiple staff members. Connecticut's COVID-19 FAQ bot, for instance, handled nearly 40,000 interactions in 4 months, equivalent to the workload of 4 full-time employees during that period.

Tools Used

Power Virtual Agents (PVA): Now part of Microsoft Copilot Studio, which allows no-code creation of chatbots and integration of Azure OpenAI for GPT-based responses.

Microsoft Teams: Acts as a chat app and/or be embedded on a public website (via the PVA web chat widget or Direct Line channels).

Dataverse or SharePoint: Used to serve as the knowledge base for FAQ content, or PVA's built-in generative AI feature can be used to ingest existing FAQ documents and answer from them.

Power Automate: Used for flows extend the bot's capabilities by connecting to back-end systems (e.g., to fetch the status of an application or submit a service request). The chatbot leverages Natural Language Processing (NLP) to understand user questions, and it can integrate with Translation services for multilingual support (critical in diverse communities).

All components can run within Microsoft's secure cloud, PVA is available in Government Community Cloud environments, ensuring compliance with public sector security and privacy requirements.

Key Actions & Data Flow

When a citizen interacts with City Assist, the flow is:

- A user asks a question in natural language (e.g. via the chat interface on the city website or a Teams chat). The query might be, *"How do I apply for a business license?"*.
- The chatbot receives the utterance and uses its AI language understanding to identify the user's intent and relevant details. It matches the query to a pre-defined topic or uses the generative AI model to form an answer from a knowledge source. For common FAQs, the bot has a predefined answer or follows a guided conversation.
- If the question is straightforward (found in its knowledge base), the bot responds instantly with the information, perhaps including links to forms or web pages. For example: *"To apply for a business license, you need to fill out form X and submit it on the licensing portal. Here's the \[link]."*
- For more complex requests, the bot can ask follow-up questions. It might gather additional info through a guided dialog. (E.g., *"To check your application status, please provide your application ID."*)
- The bot can call Power Automate actions to fulfill requests that require backend data. For instance, if the user wants to know the status of a permit, the bot triggers a flow that looks up the permit status in a SharePoint list or database and returns the result: *"Your permit (ID 12345) is currently Approved and ready for pickup."*
- If the inquiry falls outside the bot's scope or the user requests a human, the bot can escalate gracefully. It might offer to connect to a live agent (handoff to a human via Teams or a support system) or collect the user's info for a callback. This ensures citizens aren't left without help when the AI can't handle

something. (The system recognizes when an issue is too complex or sensitive and routes it appropriately.)

- Throughout the interactions, the chatbot logs transcripts and user satisfaction (thumbs up/down feedback). These logs feed into analytics for continuous improvement of the bot's knowledge. The chatbot team can review unanswered questions and add those into the bot's repertoire over time.

How to Build the Agent

Assuming you have admin access to the Microsoft 365 environment (including Power Platform) and the necessary licenses:

1. **Plan the Knowledge Base:** Collect the frequently asked questions and answers from relevant departments (e.g., general city info, service office hours, procedures for common permits, etc.).

   Organize these into topics. For example: *"Trash & Recycling Pickup," "Business License Application," "COVID-19 Resources,"* etc. This content can come from existing FAQ pages or staff knowledge. Ensure answers are reviewed for accuracy and policy compliance.

2. **Create the Bot in Copilot Studio (Power Virtual Agents):** Go to the Power Virtual Agents interface (now integrated in Copilot Studio). Create a new bot for your government entity (you might call it *"City Assist"*). If available, choose the appropriate environment (for government, use GCC High if required). PVA will provision a bot with a default language (you can add more languages later).

   In the Topics section, create a topic for each FAQ or use PVA's suggested topics by uploading your FAQ documents. For instance, you can paste the text of an existing FAQ into PVA and use the AI to generate a topic from it. Define trigger phrases for each topic (e.g., *"business license," "apply for license," "start a business,"* etc. trigger the Business License topic).

3. **Leverage Generative AI for Answers:** Power Virtual Agents allows you to enable a "GPT model" to draft answers from a knowledge source. You can add your FAQ document or website URL to the bot's content. The bot will then use OpenAI GPT-4 behind the scenes (with Microsoft's controls) to answer questions that don't exactly match a predefined topic, by pulling information from those documents.

   This is essentially Retrieval Augmented Generation: the bot retrieves relevant info and forms an answer with it. Enable this so the bot can handle variations of questions. Test it by asking something not worded exactly as in your topics to see if it still responds appropriately.

4. **Integrate Backend Systems with Power Automate:** Identify any tasks the bot should handle beyond static Q&A. Common examples: checking status (permit status, case status), scheduling an appointment, or submitting a service request. For each, create a Power Automate flow that the bot can call. In PVA, under "Actions," you can create new flows or use templates.

   For example, build a flow *"GetPermitStatus"* that takes an input (permit ID) and returns the status from a SharePoint list or SQL database. The flow might use a Dataverse table where new permit applications are tracked. Once the flow is tested to return the needed info, call this flow from the bot's topic using an Action node.

   In the conversation, the bot will ask the user for the permit ID, run the flow, then display the result to the user. This makes the bot interactive and transactional, not just informative.

5. **Enable Multilingual Support (if needed):** Government services often need to support multiple languages. Agents can be translated, or you can maintain separate language topics. If using the out-of-box translation, enable the feature so that if a user types Spanish, the bot auto-detects and responds in Spanish.

   Alternatively, integrate Azure Cognitive Services Translator in a flow to translate the user's question, then translate the answer back. Providing answers in the user's language broadens accessibility (for example, offering Spanish or Arabic answers to include non-English speakers). Also ensure the bot's interface is accessible (meeting ADA and Section 508 standards). PVA's web chat is compliant by default for screen readers etc.

6. **Test the Chatbot Thoroughly:** Before publishing, test the bot in the built-in test chat window. Try various phrasings of questions (*"I want a business license," "get business permit," "start company license"*) to ensure the bot picks the right topic.

   Test the error handling by asking something completely off-topic and see how it responds (you may craft a default fallback message like *"I'm sorry, I'm not sure about that. Please rephrase or type 'help' for options."*). If escalation to a human is part of scope, test that path (PVA can trigger an escalation event. For example, handoff to an operator on Live Assist or simply provide contact info). Make sure sensitive topics are either answered with approved messaging or escalated.

7. **Deploy to Channels (Web and Teams):** Once happy with performance, publish the bot. In PVA, publishing makes it available to connect to channels. Add the Web Chat channel to get an embed code or direct script you can add to your public website. For instance, embed the chatbot on the city's contact or help page so that users see a *"Chat with us"* bubble.

   Also deploy the bot to Microsoft Teams as a Teams app: this is useful if you want call center staff or internal teams to use the same bot or if you provide citizen support via Teams (some gov orgs may allow citizens to chat via Teams or more likely, this is for internal use, but in our case, mainly web is citizen-facing). You could also deploy the bot on other channels like Facebook Messenger or SMS if your constituents use those; PVA supports multiple channels out of the box.

8. **Monitor and Improve:** After go-live, use PVA's Analytics dashboard to monitor usage. Track metrics like number of sessions, resolution rate (how often users got their answer vs. abandoned or escalated), and what questions were asked.

   Pay particular attention to the "uncategorized" topics or where the bot responded with the fallback – those indicate gaps in the knowledge. Update the bot regularly to cover those. For example, if several people asked about a new program ("What is the green homes initiative?") that wasn't initially in the knowledge base, add a topic or content so the bot can answer next time.

   Continually improving the bot will increase its effectiveness. Also, gather user feedback. You can add a simple question at the end of chat like *"Did I answer your question?"* to collect thumbs up/down or a brief survey. This feedback loop, combined with usage analytics, creates a cycle of continuous improvement for the virtual assistant's performance over time.

Example Prompts

The Routine FAQs

A resident might ask, *"What are the hours of the public library?"* The chatbot looks up the library hours from its knowledge base and responds: *"All branch libraries are open 9am-6pm Mon-Sat, and 12pm-5pm on Sunday."* This saves a phone call to city offices.

Guiding to Services:

Someone types, *"I lost my job, what benefits can I get?"* The bot recognizes this is about unemployment assistance. It may not directly administer state unemployment, but it can provide guidance: *"I'm sorry to hear that. You may be eligible for provincial Unemployment Benefits. You can apply through the State portal. # for instructions and required documents. Also, our city offers job training programs – would you like information on those?"* For multi-turn, it could proceed to share job training info if the user says yes.

This kind of answer ensures the citizen gets accurate, helpful info instantly, which is crucial especially in crisis times (during COVID-19, such chatbots handled huge volumes of benefit questions, reducing wait times for users).

Transactional Query

A user asks, *"What's the status of my housing permit application?"* The chatbot might respond, *"Sure, to check that I need your application number."* The user enters it.

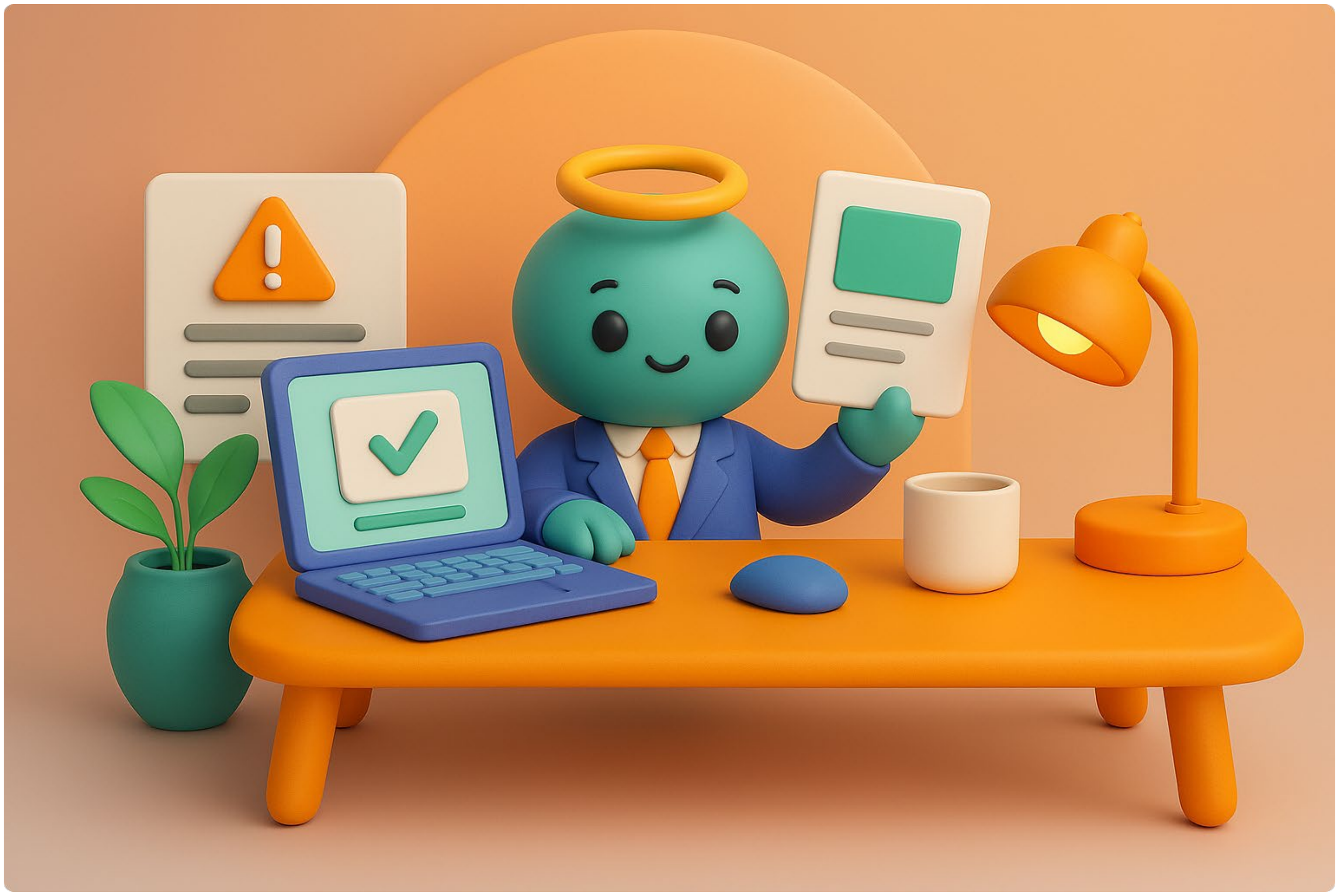
The bot invokes the flow, gets the status ("Approved on Oct 10, pending pickup"), then replies: *"Your application #HOU-2025-000123 is Approved. You will receive an official email soon. Congratulations!"* Optionally, it could even ask, *"Would you like me to mail the permit, or will you pick it up?"* if that's within scope, guiding the next steps.

Escalation to a Human

If a user says, *"I need to speak to someone about my tax assessment,"* the bot can recognize this as a request for human assistance. It might respond, *"I'll get you to a live agent. Please hold..."* and then create a ticket or transfer to a live chat queue (depending on integration).

Or if after hours, it could say, *"Our staff will call you back on the next business day."* The key is the user is not left without an answer. (The system's design ensures complex cases are seamlessly handed off.)

Agent: Intelligent Permit Processing Assistant



Agent Name: Permit Pal

Agent Description

An AI agent that assists with processing forms, applications, and documents in government workflows. Public sector agencies handle large volumes of paperwork, business license applications, permit forms, grant applications, case reports, you name it.

This copilot's purpose is to speed up and streamline document-centric processes by automatically extracting information, checking for completeness and compliance, and even drafting initial responses or decisions. It acts like a junior analyst that can read and summarize documents in seconds.

For example, a city might deploy a "Licensing Copilot" to auto-screen permit applications: it could cross-check each application against criteria (are all required fields filled? any red flags?), then recommend an approval or forward it to a human for further review. This reduces manual data entry and ensures nothing gets overlooked.

Tools Used

Document Processing model: AI Builder can be trained to extract structured data from forms (like PDFs or scanned images of applications), leveraging AI to recognize fields like names, addresses, checkboxes, etc.

Power Automate: Used to orchestrate the workflow. When a new form is received, it triggers the AI Builder model to run, then routes the outputs accordingly.

Microsoft Dataverse: Data can be stored and managed for a robust database with business logic. For simpler scenarios, a SharePoint list or Excel file can be used to manage the data.

Copilot Studio or Power Apps: Used to provide a user interface for human reviewers to view the AI-extracted data and intervene when needed (for example, a simple canvas app where an officer sees what the AI extracted and clicks "Approve" or "Needs Correction").

Microsoft Word (Optional): Used to utilized to draft documents (like an approval letter or a summary of the application) using generative AI based on the extracted info.

Outlook (Optional): May be involved if the agent is intended to send out emails (like requesting missing info from the applicant).

All these tools combined create an end-to-end solution where AI does the heavy lifting of reading and initial evaluation, while humans handle exceptions.

Key Actions & Data Flow

A typical flow for the Permit Pal Assistant:

Intake: An application or form is submitted. This could come via email (e.g., someone emails a PDF) or an online form (perhaps a Microsoft Form or Power Apps portal) or scanned paper converted to PDF. For example, a contractor uploads a building permit application PDF to a SharePoint folder or emails it to a specific address.

Trigger: The arrival of the document triggers a Power Automate flow. For instance, a flow watches a SharePoint library "New Permit Applications." When a file is added, the flow kicks off.

AI Extraction: The flow calls the AI Builder Form Processing model to extract key fields from the document. The model outputs structured data: e.g., Applicant Name: Jane Doe; Address: 123 Main St; Permit Type: Electrical; Project Valuation: $50,000; Required Documents Provided: Yes/No for each… etc. It can also extract tables or checkboxes if needed. This turns an unstructured form into a set of data the system can understand.

Data Storage: The extracted data is then stored in a Dataverse table (or SharePoint list).

For example, a new record *"Permit Application #1234"* is created with all these fields populated. The original file can be linked or attached to this record as well for reference.

Automated Checks: The Power Automate flow (or Dataverse logic) performs business rule validation on the data. For example: Are all required fields present (no nulls)? Are any required attachments missing (maybe the form had a checklist of attachments like proof of insurance, site plan)?

The agent can flag missing items: e.g., if "Site Plan Attached" is false, mark the application as Incomplete. It can also cross-verify data: maybe call another service to verify the applicant's license number is valid or ensure the project value doesn't exceed a threshold for that permit type.

If everything looks good, the application might be tentatively marked Complete. If something's off, it's marked Needs Attention. Essentially, the AI does a triage: separating straightforward cases from those that need human intervention.

Initial Decision Recommendation: The agent can go further to even recommend an action. For instance, if an application meets all criteria (no missing info, passes automated rules), it could mark it *"Recommended for Approval"*.

If it finds issues (missing data, or conditions that require review, like an unusually high project value or an expired contractor license), it flags for manual review or *"Recommend Escalation"*. This is a rule-based or AI-based decision: more advanced implementations could use a machine learning model trained on past approvals to predict an outcome, but even simple condition logic yields value.

Drafting Response: Based on the above, the agent can draft a response document or email. For a straightforward case, it might draft an Approval Letter: using a template in Word with placeholders filled in (applicant name, permit number, any conditions of approval, etc.).

For a problematic application, it might draft an email to the applicant listing what's missing or asking for clarification on certain points. Generative AI (Copilot) can help here, for example, use a prompt to Draft a polite email: *"Dear Jane Doe, we reviewed your application for an Electrical Permit. It appears that the site plan document is missing. Please submit the site plan so we can proceed. Thank you."*

The AI can pull the list of missing items and insert into a pre-defined email style.

Human Review & Approval: At this stage, a human officer is involved but their job is much easier. They open a Power Apps dashboard (or maybe they get an email notification with a summary). They see a queue of new applications where some are marked "Recommended Approval" with all data present, and some are *"Needs Review"* with flags.

The officer can click on an application, see the data and the AI's notes (e.g., *"Missing site plan"* or *"All criteria met"*), and see any drafted outputs (an approval letter ready, or an email asking for info). The officer verifies and with one click, confirms the action.

For the complete ones, maybe they just hit Approve and the system will send out the approval letter email via Outlook and update the status in the database. For the incomplete one, the officer might edit the draft email if needed and send it or add a comment and mark it as needing applicant follow-up.

Feedback Loop & Tracking: Each application record now has an outcome (approved, denied, info requested) and the time taken. The system can track how many were auto-approved vs needed human input.

This data can be used to refine the rules or train better AI models. All interactions are logged for audit (important in government decisions).

If the AI made an incorrect extraction or recommendation, the human could correct it, and those corrections can be fed back into AI Builder to retrain the form model or improve the logic. Example, if the AI often misreads a certain field on the form, you provide more samples or adjust the form design.

How to Build the Agent

1. **Prepare a Structured Data Store:** First, define where you'll keep the application data and what fields you need. Using Dataverse is ideal because you can enforce required fields and relationships.

   Create a Dataverse table called *"Permit Applications"* (or use an existing case management entity if one exists). Add columns for each piece of info you want to capture: Applicant Name, Contact, Permit Type, Submission Date, each required attachment or field, etc., and columns for status, and AI's recommendation.

   Alternatively, a SharePoint Online list can suffice for simpler needs, but Dataverse provides more scalability and integration with model-driven apps.

2. **Train an AI Builder Form Processing Model:** In Power Apps (make sure you're in an environment with AI Builder credits available), create a new Form Processing model. *Upload at least 5 sample forms* of the same type (e.g., past permit application PDFs or scans).

   Label the fields on each form in the training interface – for example, draw boxes around *"Name"*, *"Address"*, *"Permit Type"*, *"Description of Work"*, etc., and assign those labels. AI Builder will train a machine learning model to recognize those fields in new forms. If you have different types of forms (say residential vs commercial permits), you might train multiple models or a single composite model if they're similar.

   Publish the model. Now you have a model that can take an incoming file and output a JSON of extracted field values.

3. **Build the Ingestion Flow in Power Automate:** Go to Power Automate and create a new automated cloud flow, trigger: *"When a file is created in a folder"* (point it to the SharePoint or OneDrive folder where incoming applications land, or *"When a new email arrives"* if intake is email-based).

   For each new file, add an AI Builder action *"Extract information from forms"* (this action connects to your trained model). Point it to the file (you might need a SharePoint "Get file content" action before it, then feed the file content into AI Builder). The output will be all the fields the model identified, along with confidence scores.

4. **Add Data Handling to the Flow:** Next, add steps to the flow to interpret and store the extracted data. For Dataverse: use the *"Add a new row"* action to create a record in the Permit Applications table, mapping the fields from the AI Builder output to the table columns.

   For example, map *"ApplicantName"* extracted by AI to the Dataverse *"Applicant Name"* column. Also record metadata like who submitted (if known) and attach the original file (Dataverse allows file columns, or you can save a link to SharePoint). As you create the record, you can set an initial status, e.g. "Received" or "Under Review".

5. **Implement Business Rule Checks:** Still in the flow (or you could use Dataverse business rules if comfortable), add conditions to simulate an initial review.

   For instance, after saving the data, use a Condition action: "If 'SitePlan' field is blank (or false) OR 'SafetyChecklist' is incomplete, then…".

   If criteria fail, update the Dataverse record's "AI_Recommendation" field to "Needs Info" and maybe populate a "MissingItems" text field with a list of what's missing. If all criteria are met, set "AI_Recommendation" to "Approve" (meaning it found no issues).

   You can get creative. If the application is for a certain neighborhood that requires extra approval, flag that; or if cost > $100k and needs additional fee, mark that. The idea is to encode whatever decision rules a human would normally apply by eyeballing the form.

   Over time, you might incorporate more advanced AI, but start rule based. At this step, you might also generate a draft decision letter: for example, use a Word template (stored in SharePoint) with placeholders and the Populate Word Document action to fill it with the applicant's name, permit number, etc., saving the draft to a folder or attaching to Dataverse. Alternatively, you can defer document generation until a human approves, depending on your process.

6. **Notify or Queue for Human Review:** After the automated checks, have the flow notify the appropriate staff or move the case to the next stage.

   For example, if AI_Recommendation is *"Approve"*, you might still require a human sign-off, so send a Teams message or Adaptive Card to the permitting team channel: *"New Permit #1234 ready for approval. AI suggests all looks good. \[Open in app]"*.

   If it's *"Needs Info"*, notify the case owner or create a task in Planner to follow up with applicant. You could also send an email directly to the applicant at this point if you trust the AI (e.g., a form letter saying, *"we received your application, however we noticed X is missing"*. Perhaps better to let a human confirm before sending.

   A common approach: create a Power Apps (model-driven or canvas) app for permit officers to manage applications. They can open the app to see all new submissions, filter those recommended for quick approval vs those needing attention. Building that app would involve forms and views tied to the Dataverse table, which Power Apps can auto-generate.

7. **Human-in-the-loop Decision:** The permit officers use the interface to review.

   For a model-driven app: configure views like *"All New Applications – Recommended for Approval"* and *"Applications – Needs Info"* using the fields set by AI. The officer opens a record, sees an automatically filled form. They verify against the original document (which they can open if needed, e.g., to double-check something the AI extracted incorrectly).

   If everything's fine, they click an Approve button (which could trigger a second Power Automate flow or Dataverse workflow). That button action might simply update the record status to *"Approved"* and trigger an email to applicant with the attached approval letter.

   If something's missing, maybe an Email Applicant button that pops up an email draft (pre-filled with the MissingItems that AI identified) for the officer to review and send. The design here can vary, but the main point is the human doesn't have to start from scratch, they have all info neatly presented and often just need to confirm and send.

8. **Automate the Outputs:** Finalize the automation by sending out notifications. For approved cases, integrate an Outlook step to send the permit approval email (with the permit document or certificate attached if applicable). For incomplete cases, send the request-for-info email.

   These communications can be standardized and generated automatically to ensure consistency. Every time an email is sent or status changed, update the Dataverse record (for audit trail: e.g., *"ApprovalEmailSentOn"* timestamp etc.).

9. **Test the End-to-End Process:** Do a dry run with sample applications. Take a few historical permit forms (with varying quality: one complete, one missing something, one with a tricky detail) and feed them through the system.

   Evaluate: Did AI Builder correctly extract the fields? (Check the Dataverse entries vs the actual forms.) Tweak the AI model if some fields were mis-read (maybe add more training samples or adjust the document format).

   Check the logic: Did it correctly flag the missing info case? Did it erroneously flag something that was fine? Adjust the flow conditions accordingly. Ensure the notifications to humans are working and that they have the info needed to make decisions. Have a staff member use the app to process a case and gather their feedback. Maybe they want an extra field shown, or the wording of the draft email changed to be clearer, etc. Refine iteratively.

10. **Deploy and Train Staff:** Once confident, deploy the solution in production for live cases. Train the relevant staff on the new process: show them how the copilot works, reassure them that they still have control (the AI is assisting, not finalizing anything without them).

    Explain that over time, as they trust it, they might lean on its recommendations more. Also train them to flag if the AI misses something, so the team can update the system. Possibly run old cases through it to demonstrate how it would have worked, building trust.

Example Usage

Permit Application Screening

A small business owner submits a café permit application online. Normally, an admin might take 30 minutes to review it for completeness. Instead, within a minute of submission, the owner receives an email: *"Your application has been received and is under review."*

Meanwhile, internally, the Permit Pal AI has extracted all the details and found everything in order. By the time a human opens it, the AI has marked it *"Complete – OK to approve"*. The officer spends 2 minutes glancing over it and clicks Approve.

The owner then gets an automatic approval email say an hour after submission, a process that used to take 2 weeks. The owner is pleasantly surprised by the speed. The officer, on the other hand, managed to approve 50 such applications that day instead of 5, because each took minutes, not hours, to process.

Missing Information Case for Building Permits

A contractor submits a building permit but forgets to include the required site plan diagram. The AI copilot flags this immediately: it noticed the *"Site Plan Attached"* checkbox was empty on the form.

It creates a draft email: *"Dear \[Applicant Name], we noticed your submission is missing the site plan document. Please email us the site plan or upload it to continue processing your permit."* A human reviewer double-checks the finding (indeed no site plan pages were in the PDF) and with one click sends the request.

What could have been back-and-forth over weeks (or an outright rejection after long delay) now happens on day 1. The contractor sends the missing document the next day, and since the system had everything else, it goes through. This leads to faster completion rather than sitting in limbo.

Auto Drafting a Response

A government grant program closes applications and now staff must send either acceptance or rejection letters to each applicant. The Document Copilot can help by reading each application and scoring it against criteria. Suppose an applicant didn't meet an eligibility criterion (maybe their proposal didn't include a required component).

The AI can generate a tailored rejection letter that says: *"We regret to inform you that your grant application was not selected for funding. The review noted that the budget section was incomplete, which made the proposal ineligible."*

Meanwhile, for those that passed, it drafts acceptance letters. Staff then just review and send these in bulk. This saves them from copy-pasting boilerplate text 100 times and ensures each letter references the right details from the application (which the AI pulled out).

Case Triage in Social Services

Think beyond permits. A social services department receives many assistance request forms. An AI copilot could extract key info (household income, family size, urgency of need from a narrative) and assign a risk score or priority.

It might then route *"high urgency"* cases to be handled within 24 hours (alerting a supervisor), while low-priority ones are scheduled later. This ensures critical cases don't get buried in the pile.

The staff doesn't have to read every detail of every incoming form to prioritize; the AI sifts through and surfaces the ones that match certain triggers (like mentions of lack of food or imminent eviction in an open-ended response could flag as emergency). This scenario shows how document AI plus a bit of NLP can support decision support in public welfare contexts.

Agent: Policy Research & Decision Agent

Agent Name: Policy Pro Research Assistant

Agent Description

A generative AI-powered agent that serves as a research assistant for policymakers, analysts, or public affairs teams. Government officials and analysts often need to sift through vast amounts of information, legislation text, policy papers, economic reports, historical data – to make informed decisions or craft policy proposals.

This AI agent's purpose is to rapidly aggregate and summarize insights from a large body of documents and data, answering natural-language questions and even providing source citations for fact-checking. It's like having a smart research aide who can read every report in the archives and highlight the key points relevant to your query in seconds.

Tools Used

The core of this assistant is Microsoft 365 Copilot's generative AI capabilities combined with the organization's knowledge base:

Microsoft Teams (Copilot Chat): Microsoft 365 Copilot includes a chat interface (often accessed in Teams or via a special app) where users can ask questions in natural language and the AI will respond with information drawn from emails, documents, sites, etc., that the user has access to.

Copilot in Word: Within Word, an official could use Copilot to draft or analyze documents. For instance, they can ask Copilot in a Word doc *"Insert a summary of key points from the 2022 Urban Transport Report"* and it will fetch from that report and draft a section.

SharePoint and OneDrive: These serve as the repository of policy documents, research PDFs, meeting minutes, past proposals, etc. The assistant relies on having access to a well-maintained library of knowledge. Microsoft Search (and underlying Graph connectors) index this content so Copilot can retrieve it.

Power BI (with Copilot): If the question involves data (like *"What are the trends in population growth in our city over the last 5 years?"*), integration with Power BI could allow the agent to pull relevant stats or even generate charts on the fly.

Copilot Studio / Custom Plugins: Optionally, one could extend the agent with custom plugins or connectors (for example, to query an external legislation database or a news API). Within the Microsoft 365 environment, preview features allow connecting additional data sources securely.

Outlook with Copilot: For decision support, sometimes relevant info is in emails or attachments from experts. Copilot can also draw from the user's Outlook (e.g., summarizing feedback from stakeholders or pulling a statistic someone emailed).

Power Virtual Agents (as an alternative): If an organization doesn't have Microsoft 365 Copilot fully deployed, they might create a Power Virtual Agent chatbot specifically trained on their policy documents (using the ability to connect to external data or upload files for the bot to use). This could be a standalone *"Policy Q\&A Bot"* accessible in Teams.

In summary, the tools involve using Generative AI within Microsoft's ecosystem, combined with enterprise search across internal knowledge. The assistant uses retrieval-augmented generation. It finds relevant passages and then generates a synthesized answer, with citations linking back to sources for verification.

Key Actions & Data Flow

How the Policy Pro Agent works in practice:

Question Input: A user (e.g., a policy analyst) asks a question in natural language. They might be in a Teams chat with the Copilot, or in Word using the Copilot side panel. For example: *"Summarize the key economic impacts of Policy X as identified in past studies."*

Natural Language Understanding: The Copilot interprets the question using NLP. It doesn't require explicit keywords. You can ask it like you would a colleague.

Retrieval of Information: The system behind Copilot will break down the query and search the organization's content for relevant information. This could mean querying SharePoint sites, documents in OneDrive, emails, Teams wikis, or even external sources if configured.
It uses Microsoft Graph Search which can leverage semantic understanding (not just keyword matching). For example, it might find 3 research reports on *"Policy X"* and some email threads discussing *"Policy X impacts"* in the last year.

Generative Answer Formation: Copilot (powered by GPT-4) then reads the retrieved sources (or the most relevant parts of them) and composes an answer. Importantly, it uses a technique to ground the answer in actual content from the sources (to minimize hallucination).
It will draft a response that synthesizes the findings – e.g., "Past reports indicate that Policy X led to positive employment effects in two cities (sources A, B) but had negligible impact in another (source C). Key factors cited include the local cost of living and business compliance rates."

Source Citation: The assistant provides citations (in the form of footnote links or reference numbers) pointing to the documents or specific sections of text it used.
If the user is in Teams, it might show the answer with small numbers that correspond to document titles that can be clicked. In Word, it might insert the content with a citation. This is crucial for government work because decisions need to be backed by evidence, the user can click and read the original study or memo to verify the AI's summary.

Iterative Q\&A (Drill-down): The user can then follow up with more questions, refining the inquiry. For example, they might say *"Give me the source for the city that saw negligible impact."*
The assistant can then highlight the specific report (and even quote the exact line or paragraph from it). Or the user might ask, *"Compare Policy X's impact on employment vs Policy Y."* The agent would search for info on Policy Y and produce a comparative summary if possible.

Document Drafting: Another key action is drafting documents. Say the user is writing a policy brief. They can instruct the AI: *"Draft a background section on the history of Policy X adoption in our state, including any major outcomes, with references."*
The assistant will search internal archives for when this policy was adopted, any evaluation reports, etc., and compose a paragraph or two, again citing the sources (like a legislative history from the records). The user can then edit or expand this, but the heavy lifting of gathering facts is done.

Data and Chart Integration: If the query involves data that is in spreadsheets or Power BI, Copilot can attempt to fetch that.
Microsoft 365 Copilot has integration with Power BI; a user could ask *"What has been the trend in traffic fatalities since we lowered speed limits?"* If a Power BI report exists for traffic data, Copilot could retrieve the relevant figure (e.g., "Fatalities dropped 10% in the year after the speed limit reduction") or even generate a chart. This turns raw data into narrative insights on the fly.

Collaboration & Sharing: If this assistant is in Teams, the user might be able to use it during meetings or collaborative chats. For instance, in a meeting about a new policy, someone could quickly query the assistant: *"Has any other county implemented something similar?"* and share the findings live with the group. It speeds up discussions by injecting data instantly when needed.

Continuous Learning: Over time, as new content is added (new reports, new outcome data), the assistant becomes more knowledgeable. If integrated properly, whenever new documents are saved on SharePoint (like a consultant's study or a new policy evaluation), those are indexed and available to the AI.
Additionally, user feedback (if they correct or refine an answer) can indirectly help the system via improved search relevancy or updated content.

How to Build the Agent

1. **Prepare the Knowledge Base:** Gather relevant policy documents, research papers, and data. Store them in SharePoint Online libraries (e.g. a *"Policy Research"* site) so they are indexed for Microsoft Search.
    a. Enable metadata and use descriptive titles (this improves search results).

2. **Enable Microsoft 365 Copilot/Teams Chat:** Ensure your organization has Microsoft 365 Copilot enabled with access to SharePoint content. In Teams Admin, turn on Teams AI chat features for users (in 2025, Copilot in Teams supports enterprise search of SharePoint by default).

3. **Configure Grounding & Permissions:** In Microsoft 365 Copilot Studio (if available), configure the Copilot's grounding to include your SharePoint sites and any external data sources (through Graph Connectors). Set up content filters or DLP policies via Purview to protect any sensitive data from leaking.

4. **Build Automation (if needed):** Use Power Automate flows for any specialized retrieval.
   For example, a flow could call an external open data API (like a government economic indicator) when the query mentions it and returns the data to the Copilot. This flow can be triggered via a custom plugin for Copilot (Copilot Studio allows registering Power Automate flows as actions).

5. **Test with Example Queries:** Pilot the agent with sample questions. In Teams, start a chat with Copilot and ask something like *"Summarize the findings of the 2022 Urban Development Report on housing."* Verify it finds the document and produces a useful summary with a citation. Iterate on tuning search scope or adding content until answers are accurate.

6. **Deploy and Educate Users:** Roll out to a group of policy analysts. Train them (via a short demo) on how to invoke the Copilot in Teams or Word, and how to phrase queries for best results (e.g. specifying a year or program name to narrow focus). Encourage feedback to continuously refine the agent's knowledge (you might add new documents or adjust how it retrieves information).

Example Usage

A financial analyst in the government asks in Teams Copilot, *"What have been the economic impacts of the Affordable Housing Policy enacted in 2018?"*.

The Policy Insights Copilot searches the internal SharePoint library and finds a 2019 Economic Impact Study and a 2021 legislative review. In seconds, it replies in the chat with a concise analysis: *"According to the 2019 Economic Impact Study, the policy led to a 5% increase in housing construction and a $2 billion increase in economic output within two years. A 2021 review by the Budget Office noted these benefits were concentrated in urban areas." It provides brief excerpts and inserts footnote-style citations linking to those documents. The analyst can then click the citation to read more, or ask the Copilot, "Break down impacts by year and region,"* upon which it might generate a quick table or follow-up summary if data is available.

2toLEAD