```
// Main function to process screen information
function processScreenContext(screenData) {
  // Extract relevant information from the screen
  const currentApp = identifyCurrentApp(screenData);
  const currentActivity = identifyCurrentActivity(screenData,
currentApp);
  const contentType = identifyContentType(screenData,
currentApp, currentActivity);
  const screenElements = identifyScreenElements(screenData,
currentApp, currentActivity);
  const recognizedText = performSelectiveOCR(screenData,
screenElements);

  // Update current context
  updateCurrentContext(currentApp, currentActivity,
contentType, screenElements, recognizedText);

  // Determine relevant knowledge domains based on context
  const relevantDomains =
identifyRelevantDomainsFromContext(currentApp,
currentActivity, contentType);

  // Ensure relevant knowledge files are available
  for (const domain of relevantDomains) {
    accessFile(domain);
    accessFile(`${domain}_cache`);
  }

  // Generate contextual suggestions if appropriate
  if
```

```
(this.contextual_actions.suggestion_generation.based_on_scre
en) {
  const suggestions = generateContextualSuggestions(
    currentApp,
    currentActivity,
    contentType,
    screenElements,
    recognizedText,
    relevantDomains
  );

  if (suggestions.length > 0 &&
shouldPresentSuggestions(suggestions)) {
    queueSuggestionsForPresentation(suggestions);
  }
 }

 // Clean up any temporary data
 cleanupTemporaryScreenData();
}

// Function to identify which knowledge domains are relevant
based on screen context
function identifyRelevantDomainsFromContext(app, activity,
contentType) {
 const domains = [];

 // Add general vocabulary as a base
 domains.push("general_vocabulary");
```

```
  // Check app category
  if (isSocialMediaApp(app)) {
    domains.push("tiktok_terms"); // Even for other social apps,
this has relevant terminology
    domains.push("content_generation");

    if (activity === "content_creation" || activity === "editing") {
      domains.push("editorial_optimization");
      domains.push("video_terms");
    }

    if (contentType === "text_entry" || activity === "commenting"
|| activity === "captioning") {
      domains.push("text_analysis");
    }
  }

  if (isProductivityApp(app)) {
    domains.push("text_analysis");

    if (app.includes("code") || app.includes("develop")) {
      domains.push("code_generation");
    }

    if (contentType === "document" || contentType === "email") {
      domains.push("content_generation");
    }
  }

  if (isBrowsingApp(app)) {
```

```
    // Check recognized content if it's a browsing app
    if (contentHasFinancialTerms(recognizedText)) {
      domains.push("personal_finance");
    }

    if (contentHasTravelTerms(recognizedText)) {
      domains.push("travel_geography");
    }

    if (contentHasHealthTerms(recognizedText)) {
      domains.push("mental_health_terms");
      domains.push("first_aid");
    }

    if (contentHasAutomotiveTerms(recognizedText)) {
      domains.push("automotive_mechanics");
    }

    if (contentHasCookingTerms(recognizedText)) {
      domains.push("cooking_culinary");
    }
  }

  // Limit to most relevant domains to avoid unnecessary
processing
  if (domains.length > 5) {
    // Keep general_vocabulary and the 4 most specific domains
    domains.splice(5);
  }
```

```
  return domains;
}

// Generate contextual suggestions based on screen content
and knowledge
function generateContextualSuggestions(app, activity,
contentType, elements, text, knowledgeDomains) {
  const suggestions = [];

  // For TikTok content creation
  if (app === "TikTok" && activity === "content_creation") {
    // Suggest trending sounds/effects
    suggestions.push({
      type: "feature_suggestion",
      content: "Try adding trending sounds to increase visibility",
      confidence: 0.85,
      action: "open_sounds_panel"
    });

    // Suggest video structure
    suggestions.push({
      type: "content_suggestion",
      content: "Start with a hook question to increase viewer
retention",
      confidence: 0.8,
      action: null
    });
  }

  // For text entry on social media
```

```
 if (isSocialMediaApp(app) && contentType === "text_entry") {
  // Caption suggestions
  suggestions.push({
   type: "text_suggestion",
   content: "Add a question to encourage comments and
engagement",
   confidence: 0.75,
   action: null
  });

  // Hashtag suggestions based on content
  const relevantHashtags = identifyRelevantHashtags(text,
elements);
  if (relevantHashtags.length > 0) {
   suggestions.push({
    type: "hashtag_suggestion",
    content: `Try adding these hashtags: $
{relevantHashtags.join(", ")}`,
    confidence: 0.8,
    action: "insert_hashtags",
    data: relevantHashtags
   });
  }
 }

 // For email composition
 if (app.includes("mail") && activity === "composing") {
  // Formal language suggestion for professional emails
  if (isBusinessEmail(text)) {
   suggestions.push({
```

```
      type: "text_suggestion",
      content: "This appears to be a business email. Would you like help with formal language?",
      confidence: 0.75,
      action: "suggest_formal_phrasing"
    });
  }

  // Follow-up reminder suggestion
  suggestions.push({
    type: "productivity_suggestion",
    content: "Set a reminder to follow up if no response received?",
    confidence: 0.7,
    action: "set_followup_reminder"
  });
 }

 // For code editing
 if (isCodeEditingContext(app, activity, contentType)) {
  // Code completion suggestions would be handled differently
  // But we could suggest resources or debugging tips
  suggestions.push({
    type: "code_suggestion",
    content: "I notice you're working with JavaScript. Need help with any functions?",
    confidence: 0.7,
    action: "offer_code_help"
  });
 }
```

```
  // Filter suggestions based on confidence threshold
  const threshold =
this.contextual_actions.suggestion_generation.suggestion_thr
eshold;
  const filteredSuggestions = suggestions.filter(s =>
s.confidence >= threshold);

  // Limit to max allowed suggestions
  const maxSuggestions =
this.contextual_actions.suggestion_generation.max_suggestio
ns;
  if (filteredSuggestions.length > maxSuggestions) {
    filteredSuggestions.sort((a, b) => b.confidence - a.confidence);
    return filteredSuggestions.slice(0, maxSuggestions);
  }

  return filteredSuggestions;
}
```